AN14542

NETC Virtualization on IMX95LPD5EVK-19

Rev. 3.0 — 20 April 2026 Application note

Document information

Information	Content
Keywords	AN14542, i.MX 95, NETC virtualization, Ethernet, IMX95LPD5EVK-19
Abstract	This application note describes the framework of NETC virtualization on IMX95LPD5EVK-19.

1 Introduction

This document describes the framework of the Net Controller (NETC) virtualization on IMX95LPD5EVK-19. It provides the details to set up and run the `netc_share` demo. This demo tests the NETC virtualization use case between the Cortex-M and the Cortex-A cores.

The hardware environment is:

- The IMX95LPD5EVK-19 board
- The Linux kernel v6.6.36_2.1.0
- The MCUXpresso Software Development Kit (SDK) v2.16.001

2 NETC virtualization

Each NETC port on the IMX95LPD5EVK-19 board supports up to two Virtual Station Interfaces (VSI). The VSI 0 and the VSI 1 are assigned to the Linux running on the Cortex-A core. The Physical Station Interface (PSI) is assigned to the Cortex-M7 SDK.

These interfaces have independent MAC addresses, which support transmitting and receiving network packets.

2.1 Overview

The NETC peripheral has standalone drivers for OS running on the Cortex-A core, and Real Time Operating System (RTOS) running on the Cortex-M core. The drivers restrict the use of a single Ethernet port across both cores, which is one of the advantages of NETC virtualization.

Figure 1 shows the NETC virtualization architecture divided into the Cortex-M7 SDK and the Linux. The Cortex-M7 SDK and Linux communicate with each other through Remote Processor Messaging (RPMSG) so that Linux can complete the initial configuration of NETC.

The Cortex-M7 SDK owns the Ethernet Controller (ENETC) port, PSI, and configuration space. It performs preinitialization for the Virtual Functions (VF) and creates a shared memory as a virtual configuration space of 1 MB. The Linux generic Peripheral Component Interconnect (PCI) driver performs enumeration and discovers basic configurations through the virtual configuration space, which the Cortex-M7 SDK creates.

The Linux owns the VSI0 and VSI1 of ENETC2, and uses them for the Linux network stack and the Data Plane Development Kit (DPDK) respectively. To write both the virtual and real configuration spaces for the reuse of the generic PCI driver, a new PCIe ECAM driver requests the Cortex-M7 SDK.

To handle the operational requests of the Cortex-A on the virtual configuration space, the Simplified Real-Time Messaging (SRTM) NETC service driver based on the RPMSG is created on the Cortex-M7 SDK. On Linux, the VSI driver completes the probe process, which allows the VSI to send the relevant request data to the PSI through VSI-to-PSI messaging using a mailbox within the NETC.

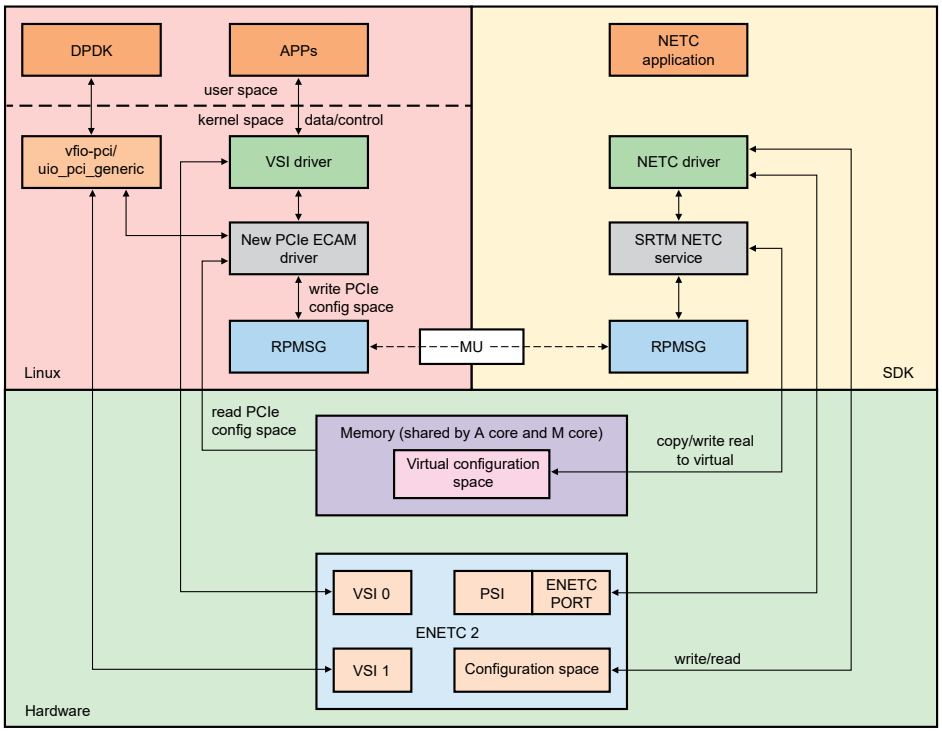

Figure 1. NETC virtualization architecture

2.2 Cortex-M7 preinitialization

The system Manager (SM) assigns the ENETC hardware resources to the Cortex-M7 before the Linux initialization. To discover PCI devices dynamically, the Cortex-M7 must perform preinitialization for the Linux. Figure 2 shows the Cortex-M7 VF preinitialization.

The Cortex-M7 preinitializes the VSI 0 and VSI 1 as follows:

1. Set the Local Domain Identifiers (LDID) for the VF as follows:

```
ENETC2 VF0: *((volatile uint32_t *)IERB_V4FAUXR) = 5;
ENETC2 VF1: *((volatile uint32_t *)IERB_V5FAUXR) = 6;
```

   The VSI function auxiliary registers are part of an IERB memory map, which starts at the address 0x4CDE0000H. These VSI function auxiliary registers contain semiopaque information. This function transmits the semiopaque information for all system transactions. The lower 4 bits specify the LDID. In this use case, choose the ENETC2, which owns the IERB_V4FAUXR and IERB_V5FAUXR.
   Ensure that the value of the LDID is consistent with the configuration in the SM. In this use case, SM assigns the sixth and seventh LDID to the Cortex-A as the VF. Set it to five and six, starting from zero.
2. Set the MAC addresses of the VF as follows:

```
ENETC2 VF0: PSI1PMAR0~ PSI1PMAR1
ENETC2 VF1: PSI2PMAR0~ PSI2PMAR1
```

   Each VF has its own MAC address and works as a standalone port. These registers set the MAC address. For example, in ENETC2 VF 0, setting registers `PSI1PMAR0` and `PSI1PMAR1`, sets a primary MAC

address. The width of the registers is 32 bits. The registers are defined in the network byte order (big-endian). The register `PSI1PMAR1` uses only the lower 2 bytes for the left MAC address.

3. Assign the Tx/Rx ring number to the VF as follows:

```
Tx: PSI1/2CFGR0[NUM_TX_BDR]
Rx: PSI1/2CFGR0[NUM_RX_BDR]
```

   - `NUM_TX_BDR`, the field of `PSI1CFGR0` and `PSI2CFGR0`, determines the number of transmit buffer descriptor rings assigned to the Station Interface (SI)
   - `NUM_RX_BDR` determines the number of receive buffer descriptor rings assigned to the SI

4. Assign the MSI-X interrupt number to the VF as follows:

```
PSI1CFRG2 AND PSI2CFGR2
```

5. Enable the VSI 0 and the VSI 1 as follows:
   `PMR[SI1EN]` AND `PMR[SI2EN]`
   `PMR` is the ENETC mode register for the port. To enable each SI, it has the following three fields:
   - `SI2EN`, where SI2 is VF1
   - `SI1EN`, where SI1 is VF0
   - `SI0EN`, where SI0 is PSI
6. Configure other supported configurations, such as receive side scaling.

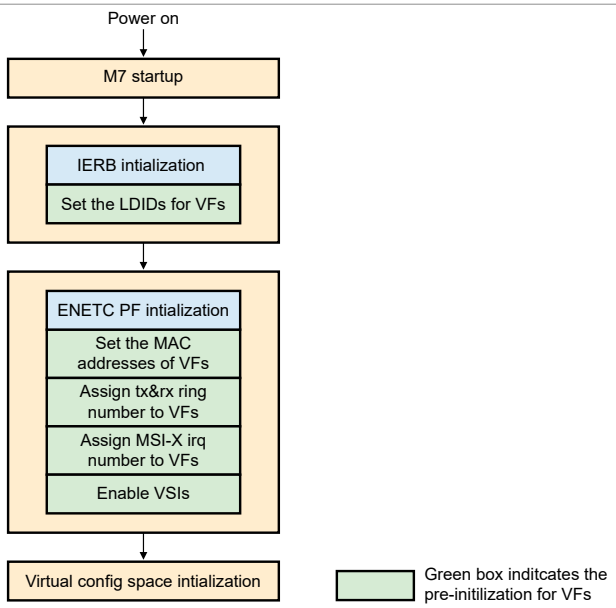

Figure 2. Cortex-M7 preinitialization

The Cortex-M7 must create a shared memory for the virtual configuration space like the PCIe configuration space. This memory block is 1 MB, which matches the real configuration space of the NETC. The address range is from 0x88224000 to 0x88323FFF. Figure 3 shows the virtual configuration space initialization.

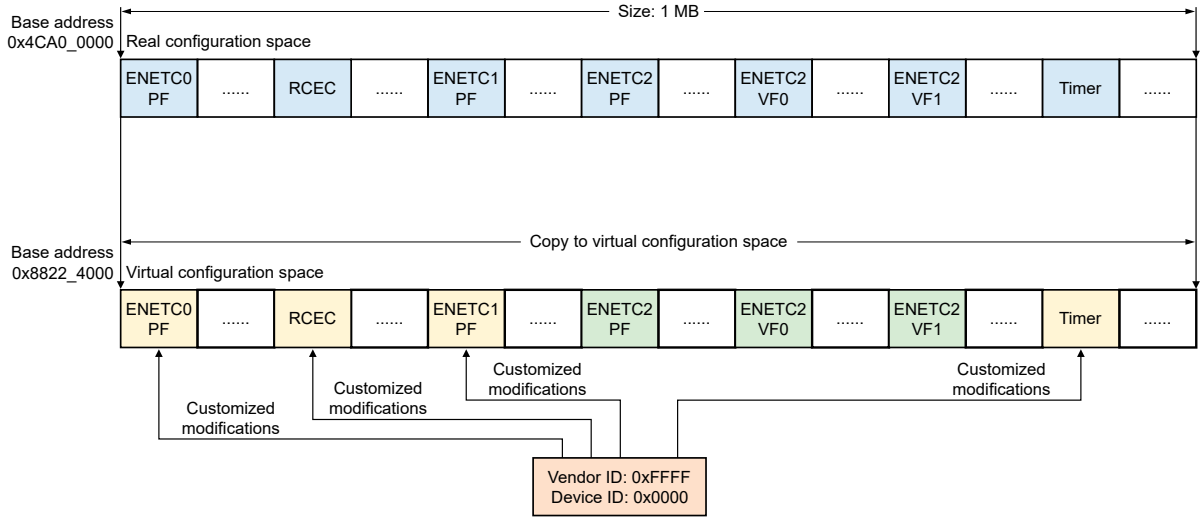

Figure 3. Cortex-M7 virtual configuration space initialization

Note: *Modify the vendor ID to 0xFFFF and the device ID to 0x0000. The generic PCI driver (Linux/Cortex-A) considers these functions as not existing when traversing the configuration space. This change can reduce the RPMSG communications.*

2.3 Linux enable VF

To communicate with the Cortex-M, the Linux kernel creates an RPMSG channel 7, after startup. Figure 4 shows the RPMSG format and Table 1 lists the values of the fields.

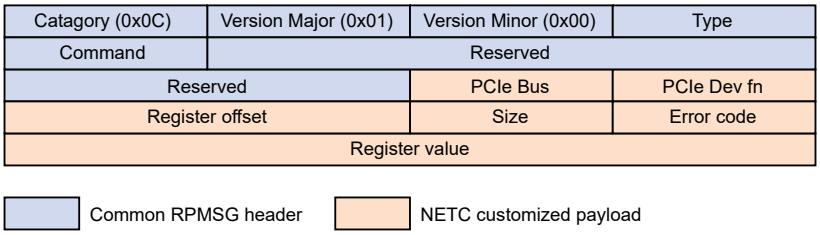

Figure 4. NETC RPMSG format

Table 1. RPMSG field values

Fields	Values
Type	0x00: Request filled by the Cortex-A
	0x01: Response filled by the Cortex-M7
Command	0x01: Write
Size	0x01: Register width is 1 byte
	0x02: Register width is 2 bytes
	0x04: Register width is 4 bytes
Error code	0x00: Command executed successfully
	0x01: Command executed failed
Other fields	Other

To enable the network devices, the Linux performs a generic PCI driver probes and enumeration process. Based on preinitialization, the Linux discovers and configures the VF of NETC whose Physical Function (PF) is assigned to the Cortex-M7.

For reading operation, the Cortex-A directly reads the virtual configuration space. For write operation, the Cortex-A sends the write RPMSG message to the Cortex-M7 and waits for the response of the Cortex-M7 or timeout. When the Cortex-M7 receives the message, it performs the following functions:

1. Writes the real configuration space.
2. Reads the value of the register from the real configuration space.
3. Writes the value to the address corresponding to the virtual configuration space, so that the Cortex-A can read the correct value from the virtual configuration space.

For example, if the Cortex-A wants to modify a register value for the VF and it is unable to access the physical configuration memory. It performs the following functions:

1. It sends an RPMSG to the Cortex-M7.
2. The Cortex-M7 receives the message.
3. The Cortex-M7 writes the physical configuration, then writes it back to the virtual configuration space for the Linux to access it.
4. The Linux assumes that it has modified this register autonomously.

For the write operation to the configuration space of the ENETC PF, the Cortex-M7 must not write both the real and virtual configuration spaces and can directly response to the Cortex-A, except for the SRIOV-related registers (to enable VF, the Cortex-M7 sets the SRIOV-related registers).

In addition, for the Function Level Reset (FLR) operation of VF, the Cortex-M7 must wait for the completion of FLR execution before writing the virtual configuration space and then response to the Cortex-A.

2.4 VSI-to-PSI messaging

To perform the VSI-to-PSI messaging, complete the VSI devices configuration and installation. If the Linux wants to perform the operation (such as querying the current link status) that involves reading and writing PSI registers, it must go through the VSI-to-PSI messaging.

The set of registers and interrupt lines communicates with the PSI. The VSI 0 and the VSI 1 send request messages for specific services, usually resources, or configuration requests to which the PSI must reply with the confirmation messages.

Figure 5 shows that to obtain the relevant information from PSI, the VSI uses the VSI-to-PSI messaging, or requests the PSI to perform the following configurations:

- VSI MAC address modifications
- VLAN filtering
- MAC address filtering
- Getting link status

Consider an example of querying link status on the VSI. It creates a VSI-to-PSI message and sends it to the PSI. The PSI accesses the related registers about link status and replies to the request from the VSI.

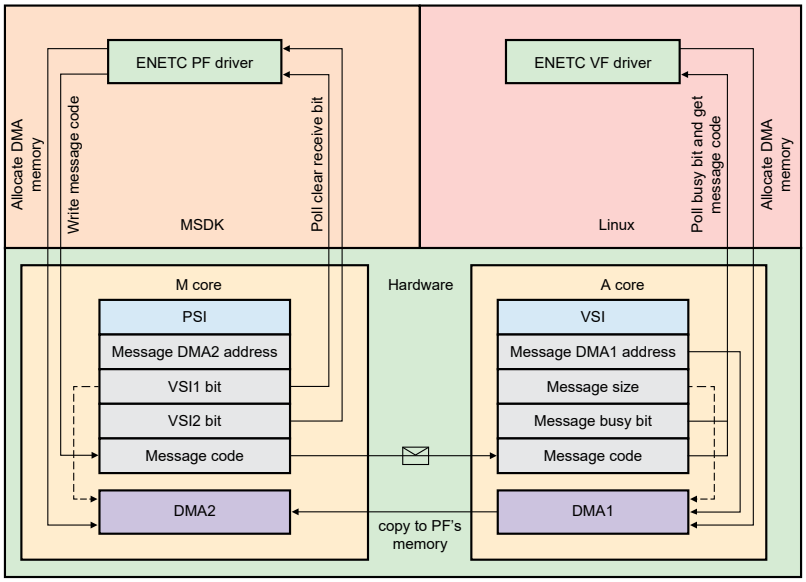

Figure 5. VSI-to-PSI messaging

3 Build netc_share demo

The `netc_share` demonstrates the NETC virtualization, for more details, see Section 2. It creates three SI on IMX95LPD5EVK-19. The Cortex-M7 uses PSI, and the Linux on the Cortex-A uses VSI 0 and VSI 1. From the outside, IMX95LPD5EVK-19 has three MAC addresses, corresponding to the three independent ports for transmitting and receiving Ethernet packets. On Linux, the network protocol stack or the DPDK handles the process of Tx/Rx. On the Cortex-M7, the `lwIP` protocol stack handles the process of Tx/Rx.

Before running this demo on the IMX95LPD5EVK-19 board, build the `flash.bin` containing the demo image.

To build the `flash.bin`, prepare the following files:

- `netc_share.bin`
- `m33_image-mx95netc.bin`
- Other imx-boot-tools files (including `bl31.bin`, `u-boot.bin`, and so on)

3.1 Build netc_share.bin

The `netc_share.bin` is the binary file compiled from the `netc_share` code in the Cortex-M7 SDK. It performs the Cortex-M7 operation, see Section 2.2, which configures the following:

- ENETC2 PSI and VSI 0 and VSI 1 MAC addresses
- Buffer descriptor rings
- Interrupts, and so on

To handle the request messages from the Linux, it initiates the SRTM NETC services. Any code that manipulates the PSI, such as receiving or sending packets using the Cortex-M7 goes in this file.

Obtain the `netc_share.bin` in the following two ways:

- Directly from the release
- Build it from the source code

The `netc_share.bin` is included in the latest Board Support Packages (BSP) for IMX95LPD5EVK-19 board as `imx95-19x19-evk_m7_TCM_netc_share.bin`. Therefore, you can obtain it easily without building.

Build the `netc_share.bin` from source code under Windows with IAR as follows:

1. Get the MCUXpresso SDK for IMX95LPD5EVK-19 as follows:
   a. Open MCUXpresso SDK Builder.
   b. Select **Boards → i.MX and i.MX RT → IMX95LPD5EVK-19 (MIMX9596xxxxN)**.

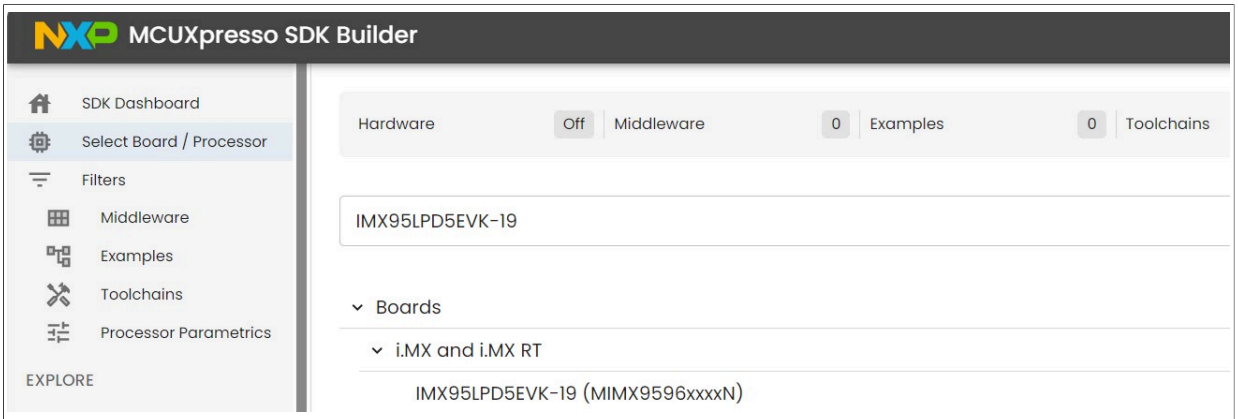

Figure 6. MCUXpresso SDK builder

   c. Click the **BUILD SDK** button at the bottom of the webpage.

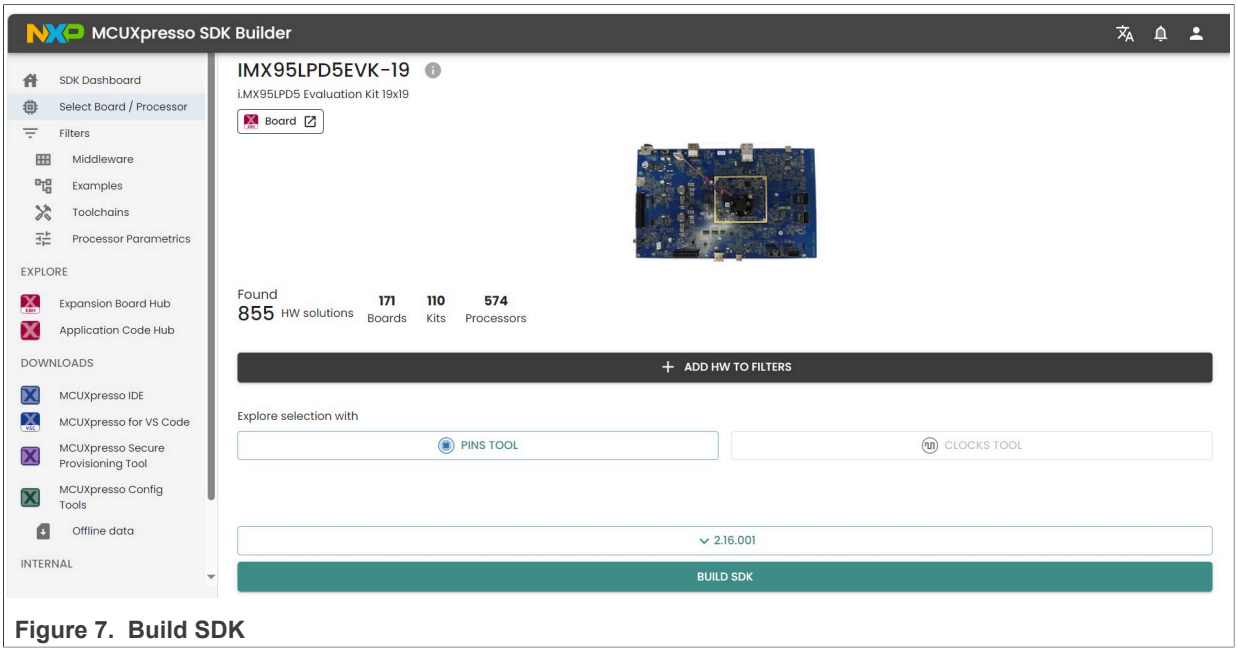

Figure 7. Build SDK

   d. Select **Host OS** and **Toolchain / IDE**.

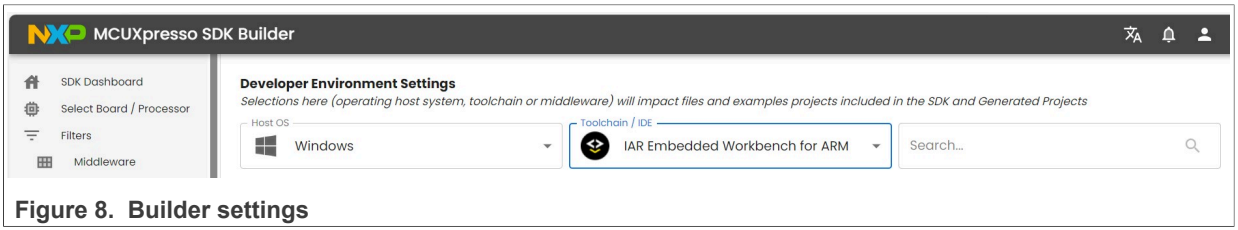

Figure 8. Builder settings

   e. Click the **BUILD SDK** button, and the MCUXpresso SDK is built as shown in Figure 9.

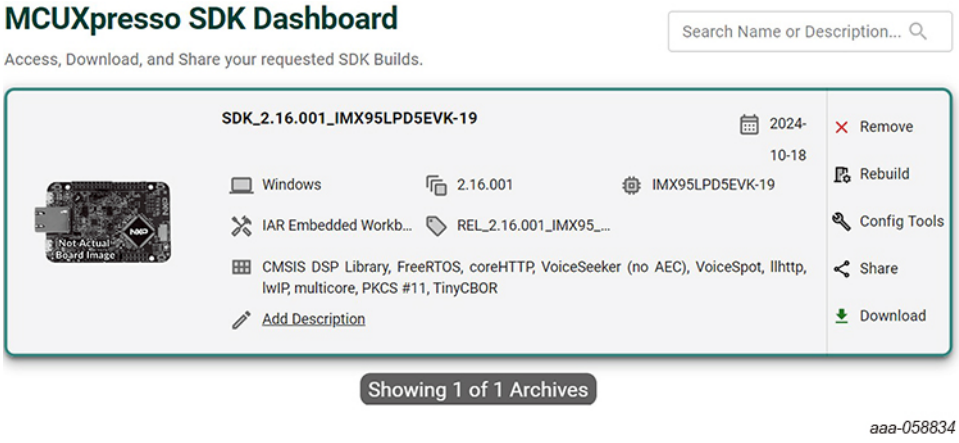

Figure 9. IMX95LPD5EVK-19 SDK

f. To get the MCUXpresso SDK, click **Download** and unzip it. Figure 10 shows the root directory of the SDK.

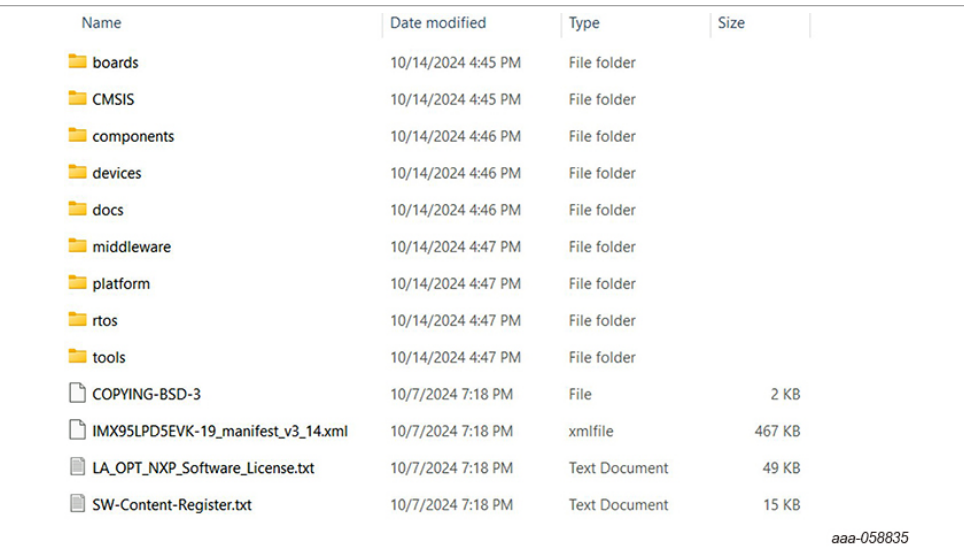

Figure 10. Root directory of the SDK

2. Build the `netc_share.bin` as follows:
   - Find the `netc_share` source code at `boards/imx95lpd5evk19/demo_apps/netc_share_sm` and the IAR workspace file at `boards/imx95lpd5evk19/demo_apps/netc_share_sm/cm7/iar/netc_share.eww`.
   - Open the file directly with **IAR Embedded Workbench IDE**.
   - To build the demo: Right-click the **netc_share - debug** and select `Make` as shown in Figure 11.

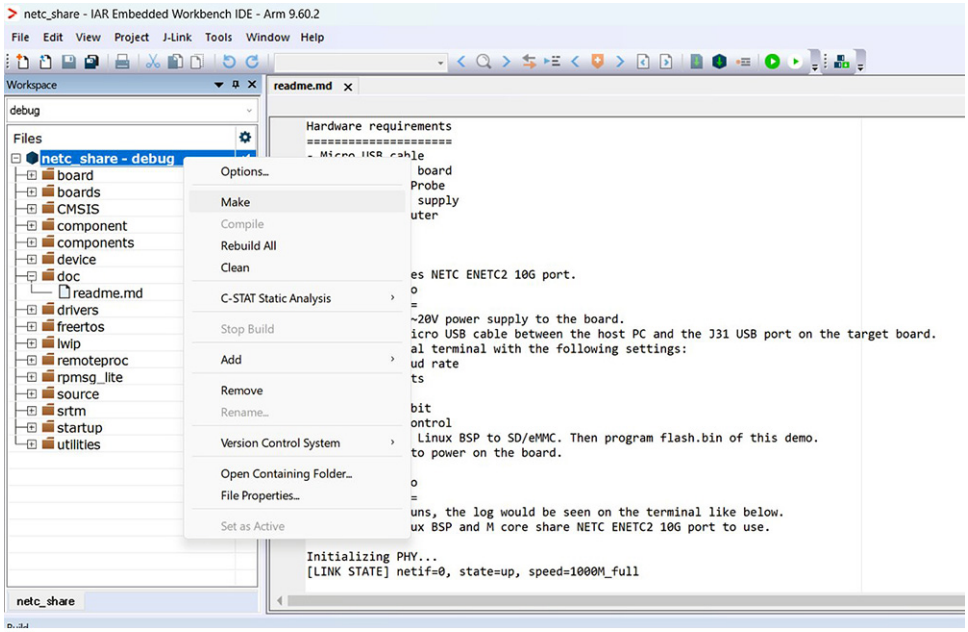

Figure 11. netc_share build by IAR

- The `netc_share.bin` is generated at `boards/imx95lpd5evk19/demo_apps/netc_share_sm/cm7/iar/Debug` by default.

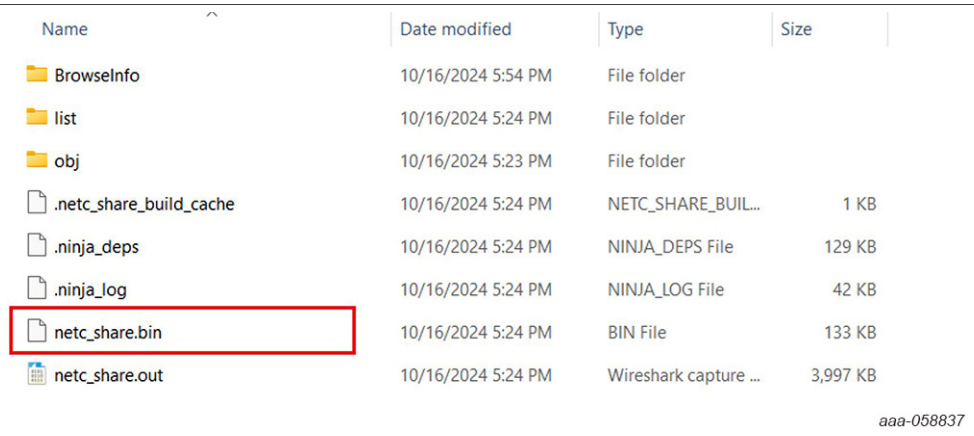

Figure 12. `netc_share` output directory

3.2 Build m33_image-mx95netc.bin

The SM runs on the Cortex-M processor on the NXP i.MX processors. The Cortex-M33 is the boot core. It runs the boot ROM, which loads the SM (and other boot code), and then branches to the SM. The SM configures some aspects of the hardware such as isolation mechanisms and starts other cores in the system.

The `netc_share` demo needs its specific `m33_image.bin` named `m33_image-mx95netc.bin`. It assigns the ENETC2 from the default Cortex-A to the Cortex-M7, and some related I2C permissions to control Physical Layer (PHY) on the Cortex-M7.

To get the sources and build them under a Linux environment, perform the following steps:

1. Clone the `imx-sm` repository as follows:
   - Get the `imx-sm` sources from GitHub via Git.

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

2. Configure the Toolchain as follows:
   a. Use an Arm cross-compiler for compiling.
   b. Download and install the required `arm-none-eabi` toolchain from the Arm GNU Toolchain Downloads.
   c. Set the `TOOLS` shell variable to the Toolchain directory.

```
export TOOLS=<directory contain the GCC compiler directory>
```

   For example, if the GCC Toolchain is installed in `~/tools/arm-gnu-toolchain-12.3.rel1-x86_64-arm-none-eabi`, set `TOOLS=~/tools`.
3. Build the `m33_image.bin` as follows:
   a. Generate the `mx95netc` as `imx-sm`, which does not contain the `make config` item.
   b. Execute the `make` command.

```
make config=mx95netc all
```

   c. The `m33_image.bin` is generated at `build/mx95netc/m33_image.bin`.

3.3 Build imx-boot-tools

The Yocto provides the files needed to build the `flash.bin`. To set up Yocto environment for IMX95LPD5EVK-19, see *i.MX Yocto Project User's Guide* (document UG10164).

After the Yocto environment configuration is completed, build the `imx-boot-tools` as follows:

```
bitbake imx-boot
```

Wait for the build to complete, and the following files are available at `tmp/deploy/images/imx95evk/imx-boot-tools`:

- `bl31-imx95.bin`
- `lpddr5_dmem_v202311.bin`
- `lpddr5_imem_v202311.bin`
- `lpddr5_dmem_qb_v202311.bin`
- `lpddr5_imem_qb_v202311.bin`
- `mkimage_imx8`
- `mx95a0-ahab-container.img`
- `oei-m33-ddr.bin`
- `oei-m33-tcm.bin`
- `soc.mak`
- `u-boot-imx95evk.bin-sd`
- `u-boot-spl.bin-imx95evk-sd`

Note: *These files are not dedicated to `netc_share` demo but are required when building `flash.bin`.*

3.4 Build flash.bin

The `flash.bin` for IMX95LPD5EVK-19 contains built-in boot images, and the `netc_share` image. To build a `flash.bin` under the Linux environment, use `imx-mkimage` as follows:

1. Clone `imx-mkimage` repository as follows:
   - Get `imx-mkimage` sources from GitHub.

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

2. Copy the following files to `imx-mkimage/iMX95`:
   - `netc_share.bin`, renamed as `m7_image.bin`, see Section 3.1
   - `m33_image.bin`, see Section 3.2
   - `bl31-imx95.bin`, renamed as `bl31.bin`, see Section 3.3
   - `lpddr5_dmem_v202311.bin`
   - `lpddr5_imem_v202311.bin`
   - `lpddr5_dmem_qb_v202311.bin`
   - `lpddr5_imem_qb_v202311.bin`
   - `mkimage_imx8`
   - `mx95a0-ahab-container.img`
   - `oei-m33-ddr.bin`
   - `oei-m33-tcm.bin`
   - `soc.mak`
   - `u-boot-imx95evk.bin-sd`, renamed as `u-boot.bin`
   - `u-boot-spl.bin-imx95evk-sd`, renamed as `u-boot-spl.bin`
3. Run the `make` command as follows:
   - Before running a `make` command, update the `mkimage_imx8` at `imx-mkimage/` with 2.
   - Execute the following commands:

```
cp iMX95/mkimage_imx8 ./
make SOC=iMX95 OEI=YES flash_all
```

   - The `flash.bin` is created at `imx-mkimage/iMX95`.

   Note: *If you run `make clean`, you must run `cp iMX95/mkimage_imx8 ./` again.*

4 Setup netc_share demo

This section guides you to set up the `net_share` demo on IMX95LPD5EVK-19. Before set up, ensure that the `flash.bin` is built correctly.

4.1 Hardware connection

Figure 13 shows the connection of the IMX95LPD5EVK-19. Inserted the network cable into ENETC2. To burn `flash.bin`, the Universal Update Utility (UUU) uses the UUU download USB cable. If you do not use UUU, it can be omitted.

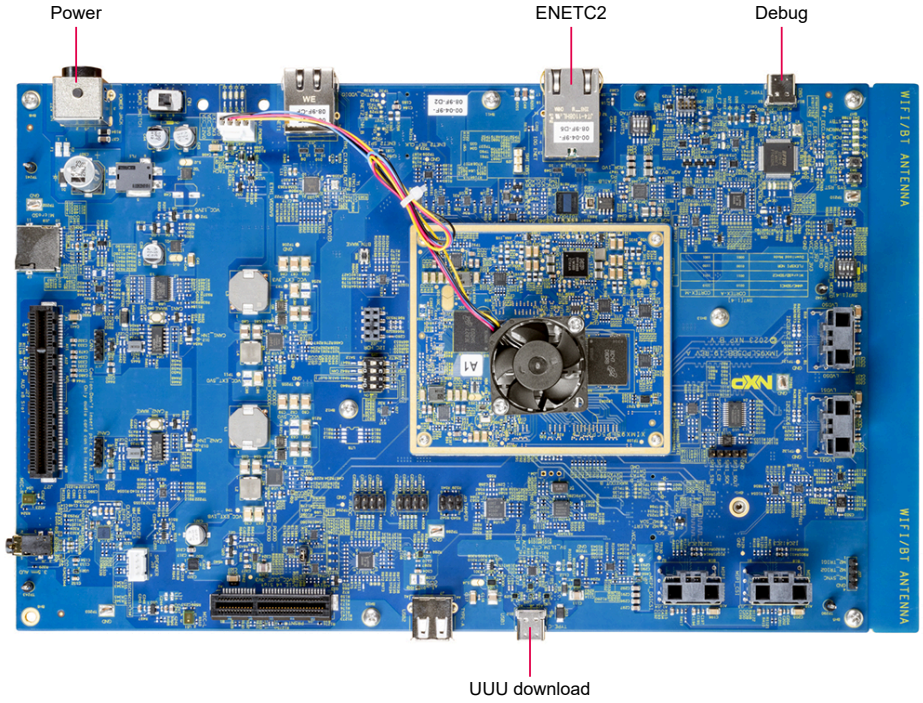

Figure 13. IMX95LPD5EVK-19 connection

Use a serial tool, for example MobaXterm, to open the first and the third serial ports on the board with the following settings:

- Baud rates = 115200
- Parity = None
- Data = 8 bits
- Stop = 1 bit

4.2 Burn the flash.bin and the Linux image to the SD card

Burn `flash.bin` to the MicroSD/eMMC at 32 kB (0x8000) offset with dd or UUU and then plug the MicroSD card to the board.

For example:

- Burn the `flash.bin` to the MicroSD card with dd as follows:

```
dd if=flash.bin of=/dev/sdX bs=1k seek=32 && sync
```

- Burn the `flash.bin` to the MicroSD/eMMC with UUU as follows:
  1. To download the board firmware, connect the USB Type-C port to the PC through the USB cable.
  2. Switch to the serial downloader mode; the boot core is the Cortex-M33.

```
uuu -b sd imx-boot-imx95-19x19-lpddr5-evk-sd.bin-flash_all flash.bin
```

Note:

- *Get the `imx-boot-imx95-19x19-lpddr5-evk-sd.bin-flash_all` from the Linux BSP.*
- *Generate the `flash.bin`, see Section 3.4.*

4.3 U-Boot configuration

Change the boot mode to SW7[1:4] = 1011 for the SD boot.

1. Power on the board and enter the U-Boot on the third serial port.
2. Set the DTB file as `imx95-19x19-evk-netc-rpmsg.dtb`.
3. If it becomes invalid after reboot, optionally, use `saveenv` to save the fdtfile configuration. Otherwise, reenter U-Boot to configure the DTB file after reboot.

```
u-boot> setenv fdtfile imx95-19x19-evk-netc-rpmsg.dtb
u-boot> saveenv
u-boot> boot
```

5 Test netc_share demo

This section guides you to test the `netc_share` demo including the VSI creation, network connectivity checking, and the DPDK test.

5.1 Test network connectivity

When the Linux boot is complete, there is no eth device to be listed.

To create the VSI 0 and the VSI 1 on Linux, run the following command:

```
echo 2 > /sys/bus/pci/devices/0002\:00\:10.0/sriov_numvfs
```

To list eth device, run `ip address`. Both the network interfaces of VF can get the IP addresses from the Dynamic Host Configuration Protocol (DHCP) server. Perform the connectivity tests like `ping` for the IP addresses.

```
$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
 default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
    valid_lft forever preferred_lft forever
2: can0: <NOARP,ECHO> mtu 16 qdisc noop state DOWN group default qlen 10
    link/can
3: eth0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP
 group default qlen 1000
    link/ether 00:00:fa:fa:dd:a0 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::200:faff:fefa:dda0/64 scope link proto kernel_ll
    valid_lft forever preferred_lft forever
4: eth1: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP
 group default qlen 1000
    link/ether 00:00:fa:fa:dd:a1 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::200:faff:fefa:dda1/64 scope link proto kernel_ll
    valid_lft forever preferred_lft forever
```

The first serial port outputs debug messages from the Cortex-M7 simultaneously.

```
Start SRTM communication
Initializing PHY...
```

```
[LINK STATE] netif=0, state=down
[LINK STATE] netif=0, state=up, speed=1000M_full

*********************************************
 DHCP example
*********************************************
 DHCP state       : BOUND

 IPv4 Address     : 192.168.0.164
 IPv4 Subnet mask : 255.255.255.0
 IPv4 Gateway     : 192.168.0.1
```

5.2 Test DPDK

This section describes a method for testing the DPDK on Linux. For more details on the DPDK use case, see *i.MX Linux Reference Manual* (document RM00293).

1. Power up the board and enter the U-Boot. Set the huge page size `hugepagesz`:

```
u-boot> setenv fdtfile imx95-19x19-evk-netc-rpmsg.dtb
u-boot> editenv mmcargs
edit: setenv bootargs ${cpuidle} ${jh_clk} ${mcore_args} console=${console}
 root=${mmcroot} default-hugepagesz=2m hugepagesz=2m hugepages=448
 iommu.passthrough=1
u-boot> boot
```

2. When the Linux boot is complete, load the `kpage_ncache.ko`:

```
insmod /usr/lib/modules/$(uname -r)/updates/kpage_ncache/kpage_ncache.ko
```

3. Create two VF for the Linux, and the `eth1` is assigned to DPDK:

```
echo 2 > /sys/bus/pci/devices/0002\:00\:10.0/sriov_numvfs
```

4. To identify the PCI address of the `eth1`, use `dpdk-devbind.py -s` as follows:

```
$ dpdk-devbind.py -s
Network devices using kernel driver
===================================
0002:00:10.0 'Device 080b' if= drv=fsl_enetc4 unused=vfio-pci,uio_pci_generic
0002:00:12.0 'Device ef00' if=eth0 drv=fsl_enetc_vf unused=vfio-
pci,uio_pci_generic *Active*
0002:00:14.0 'Device ef00' if=eth1 drv=fsl_enetc_vf unused=vfio-
pci,uio_pci_generic *Active*
```

5. Bind the `eth1` to DPDK. Meanwhile, the Linux network protocol stack uses the `eth0`:

```
$ ip link set eth1 down
$ dpdk-devbind.py -b uio_pci_generic 0002:00:14.0
```

6. To check if the DPDK port is created, execute `dpdk-devbind.py -s` as follows:

```
$ dpdk-devbind.py -s
Network devices using DPDK-compatible driver
============================================
0002:00:14.0 'Device ef00' drv=uio_pci_generic unused=vfio-pci
Network devices using kernel driver
===================================
```

```
0002:00:10.0 'Device 080b' if= drv=fsl_enetc4 unused=vfio-pci,uio_pci_generic
0002:00:12.0 'Device ef00' if=eth0 drv=fsl_enetc_vf unused=vfio-
pci,uio_pci_generic *Active*
```

7. Run the `dpdk-l2fwd`. To get the forwarding information display, use another device to ping the IP address of `eth0` as follows:

```
$ dpdk-l2fwd -c 0x1 -n 1 -- -p 0x1 -P
Port statistics ====================================
Statistics for port 0 ------------------------------
Packets sent:                        27
Packets received:                    27
Packets dropped:                      0
Aggregate statistics ===============================
Total packets sent:                  27
Total packets received:              27
Total packets dropped:                0
```

6 References

Table 2 lists the references used to supplement this document.

Table 2. References

Document	Link/how to access
i.MX Yocto Project User's Guide	UG10164
i.MX Linux Reference Manual	RM00293
MCUXpresso SDK Builder	MCUXpresso SDK Builder
Arm GNU Toolchain Downloads	Arm GNU Toolchain Downloads

7 Acronyms

Table 3 lists the acronyms used in this document.

Table 3. Acronyms

Acronym	Description
BSP	Board Support Packages
DHCP	Dynamic Host Configuration Protocol
DPDK	Data Plane Development Kit
ENETC	Ethernet Controller
FLR	Function Level Reset
IDE	Integrated Development Environment
LDID	Local Domain Identifier
NETC	Net Controller
PCI	Peripheral Component Interconnect
PF	Physical Function
PHY	Physical Layer

Table 3. Acronyms...*continued*

Acronym	Description
PSI	Physical Station Interface
ROM	Read-Only Memory
RPMSG	Remote Processor Messaging
RTOS	Real Time Operating System
SDK	Software Development Kit
SI	Station Interface
SM	System Manager
SRTM	Simplified Real-Time Messaging
UUU	Universal Update Utility
VF	Virtual Functions
VSI	Virtual Station Interface

8 Note about the source code in the document

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

Copyright 2026 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.

9 Revision history

Table 4 summarizes the revisions to this document.

Table 4. Revision history

Document ID	Release date	Description
AN14542 v.3.0	20 April 2026	Initial public release
AN14542 v.2.0	21 February 2025	Updated Legal information

Table 4. Revision history...*continued*

Document ID	Release date	Description
AN14542 v.1.0	22 January 2025	Initial NDA 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.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

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.

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