## Document information

<table>
<thead>
<tr>
<th>Information</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keywords</strong></td>
<td>REALTIMEEDGEUG, Real-time Edge Software, Real-time Networking, Real-time System, Protocols, i.MX boards, QoriQ (Layerscape) boards, i.MX 6ULL EVK, i.MX 8DXL EVK, i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK, i.MX 93 9x9 QSB, LX2160ARDB Rev2, NXP hardware platforms</td>
</tr>
<tr>
<td><strong>Abstract</strong></td>
<td>This document describes the features and implementation of Real-time Edge Software on NXP hardware platforms. The key technology components include Real-time System, Real-time Networking, Heterogeneous Multicore Framework, Heterogeneous Multi-SoC Framework, and Protocols.</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Real-time Edge software

Real-time Edge software is an evolved version of Open Industrial Linux (OpenIL) for real-time and deterministic systems in different fields. The key technology components include Real-time System, Heterogeneous Multicore Framework, Heterogeneous Multi-SoC Framework, Real-time Networking and Protocols.

• Real-time System includes PREEMPT_RT Linux, Native RTOS on Cortex-A, Jailhouse, U-Boot based Baremetal framework, RTOS and Baremetal on Cortex-M, and different combinations of these systems.
• Heterogeneous Multicore Framework provides a general software framework to support Heterogeneous AMP. It enables AMP to be inter-connected and provides a unified resource management and life-cycle management.
• Heterogeneous Multi-SoC Framework enables the usage of a combination of MPU and MCU. It extends MCU's hardware as the MPU's hardware component.
• Real-time Networking includes TSN technology, TSN standards, management, configuration, and applications. Networking and redundancy features are also supported.
• Protocols component includes support for industry standard protocols such as EtherCAT, CoE, FlexCan, OPC-UA, and others.

This document describes the features and implementation of Real-time Edge Software on NXP hardware platforms.

1.2 Real-time Edge Software Yocto Project

For using Yocto build environment, refer to the Real-time Edge Yocto Project User Guide. This document describes the steps to build Real-time Edge images using a Yocto Project build environment for both i.MX and QorIQ (Layerscape) boards.

1.3 Supported NXP platforms

The Table 1 lists the NXP hardware SoCs and boards that support the Real-time Edge software.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Architecture</th>
<th>Boot</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX 6ULL EVK</td>
<td>Arm v7</td>
<td>SD</td>
</tr>
<tr>
<td>i.MX 8DXL LPDDR4 EVK</td>
<td>Arm v8</td>
<td>SD</td>
</tr>
<tr>
<td>i.MX 8M Mini LPDDR4 EVK</td>
<td>Arm v8</td>
<td>SD</td>
</tr>
<tr>
<td>i.MX 8M Plus LPDDR4 EVK</td>
<td>Arm v8</td>
<td>SD</td>
</tr>
<tr>
<td>i.MX 93 EVK</td>
<td>Arm v8</td>
<td>SD</td>
</tr>
<tr>
<td>i.MX 93 9x9 QSB</td>
<td>Arm v8</td>
<td>SD</td>
</tr>
<tr>
<td>LS1028ARDB</td>
<td>Arm v8</td>
<td>SD, eMMC</td>
</tr>
<tr>
<td>LS1043ARDB</td>
<td>Arm v8</td>
<td>SD</td>
</tr>
<tr>
<td>LS1046ARDB</td>
<td>Arm v8</td>
<td>SD, eMMC</td>
</tr>
<tr>
<td>LS1046AFRWY</td>
<td>Arm v8</td>
<td>SD</td>
</tr>
<tr>
<td>LX2160ARDB Rev 2</td>
<td>Arm v8</td>
<td>SD</td>
</tr>
</tbody>
</table>
1.3.1 Switch settings

The Table 2 lists and describes the switch configuration for the platforms supported by Real-time Edge software.

Table 2. Switch setting for various NXP platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Boot source</th>
<th>Switch setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX 6ULL EVK</td>
<td>Internal Boot / MicroSD</td>
<td>SW602 = 0b'10 (internal boot) and SW601[1:4] = 0b'0010 (MicroSD)</td>
</tr>
<tr>
<td>i.MX 8DXL LPDDR4 EVK</td>
<td>SD</td>
<td>SW1[1:4] = 0b'1100</td>
</tr>
<tr>
<td>i.MX 8M Mini LPDDR4 EVK</td>
<td>MicroSD / uSDHC2</td>
<td>• SW1101[1:10] = 0b'0110101000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SW1102[1:10] = 0b'0001101000</td>
</tr>
<tr>
<td>i.MX 8M Plus LPDDR4 EVK</td>
<td>MicroSD / SDHC2</td>
<td>SW4[1:4] = 0b'0011</td>
</tr>
<tr>
<td>i.MX 93 EVK</td>
<td>MicroSD / uSDHC2</td>
<td>SW1301[1:4] = 0b'01100</td>
</tr>
<tr>
<td>i.MX 93 9x9 QSB</td>
<td>MicroSD / uSDHC2</td>
<td>SW601[1:4] = 0b'00111</td>
</tr>
<tr>
<td>LS1028ARDB</td>
<td>SD, eMMC</td>
<td>SD: SW2[1:8] = 0b'10001000000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eMMC: SW2[1:8] = 0b'10001000000</td>
</tr>
<tr>
<td>LS1043ARDB</td>
<td>SD</td>
<td>SW4[1:8] + SW5[1:8] = 0b'00100000_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UART1 output select</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SW3[3] = 0b'0: RJ45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SW3[3] = 0b'1: CMSIS-DAP (MiniUSB)</td>
</tr>
<tr>
<td>LS1046ARDB</td>
<td>SD, eMMC</td>
<td>SW5[1:8] + SW4[1:8] = 0b'00100000_0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UART1 output select</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SW4[4] = 0b'0: RJ45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SW4[4] = 0b'1: CMSIS-DAP (MicroUSB)</td>
</tr>
<tr>
<td>LS1046AFRWY</td>
<td>SD</td>
<td>SW1[1:10] = 0b'00100000000</td>
</tr>
<tr>
<td>LX2160ARDB Rev2</td>
<td>SD</td>
<td>SW1[1:8] = 0b'100010000</td>
</tr>
</tbody>
</table>

1.3.2 Flashing pre-built images

Pre-built images for platforms supported by Real-time Edge software can be downloaded from NXP website from the below URL:

https://www.nxp.com/design/software/development-software/real-time-edge-software:REALTIME-EDGE-SOFTWARE.

Download the image required and extract it by using the commands below: (The code below shows the commands used for LS1028ARDB-PA as an example)

```
$ unzip Real-time_Edge_v2.7_LS1028ARDB.zip
$ cd Real-time_Edge_v2.7_LS1028ARDB/real-time-edge
$ ls
atf fsl-ls1028a-rdb-jailhouse-without-enetc.dtb
dp Image-ls1028ardb.bin
fsl-ls1028a-rdb-dpdk.dtb npx-image-real-time-edge-ls1028ardb.manifest
fsl-ls1028a-rdb-dsa-swp5-eno3.dtb npx-image-real-time-edge-
ls1028ardb.rootfs.tar.bz2
fsl-ls1028a-rdb.dtb npx-image-real-time-edge-ls1028ardb.wic.zst
fsl-ls1028a-rdb-jailhouse.dtb rcw
$ zstd -d npx-image-real-time-edge-ls1028ardb.wic.zst
```
Insert SD card, device node “sdX” (for example: sdc) is created in directory “/dev/” with USB reader, flash file
“nxp-image-real-time-edge-ls1028ardb.wic” to SD card:

```bash
$ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wic of=/dev/sdc bs=1M conv=fsync
```

After flashing this image to SD card, insert this SD card into LS1028ARDB board, connect UART1 port and
open it. Then, powering on the LS1028ARDB board displays the message as shown in Figure 1.

![LS1028ARDB boot log](image)

**Figure 1. LS1028ARDB boot log**

1.4 Related documentation

All documentation related to Real Time Edge is available on the link: REALTIME EDGE Documentation. The
following documents are available:

- **Real-time Edge Yocto Project User Guide** (refer to it for using Yocto build environment)
- **GenAVB/TSN Stack Evaluation User Guide** (provides information on how to set up Audio Video Bridging
evaluation experiments of the GenAVB/TSN Stack on NXP platforms)
- **Harpoon User’s Guide** (provides information to build Harpoon Yocto images)
- **i.MX6ULL EVK GenAVB/TSN Rework Application Note** (AN13678)
- For details about the graphics feature available in i.MX 8M Plus and i.MX 8M Mini boards, refer to the **i.MX
  Graphics User's Guide**

To boot up and set up the boards mentioned in this document, refer to the instructions available in the following
user guides:

- **i.MX 6ULL EVK Quick Start Guide**
- **i.MX 8M Mini LPDDR4 EVK Quick Start Guide**
- **i.MX 8M Plus LPDDR4 EVK Quick Start Guide**
1.5 Acronyms and abbreviations

The Table 3 lists the acronyms used in this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVB</td>
<td>Audio video bridging</td>
</tr>
<tr>
<td>AMP</td>
<td>Asymmetric multiprocessing</td>
</tr>
<tr>
<td>BC</td>
<td>Boundary clock</td>
</tr>
<tr>
<td>BLE</td>
<td>Bluetooth low energy</td>
</tr>
<tr>
<td>BMC</td>
<td>Best master clock</td>
</tr>
<tr>
<td>CA</td>
<td>Client application</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller area network</td>
</tr>
<tr>
<td>CBS</td>
<td>Credit-based shaper</td>
</tr>
<tr>
<td>CDW</td>
<td>Concurrent Dual Wi-Fi</td>
</tr>
<tr>
<td>CMLDS</td>
<td>Common Mean Link Delay Service</td>
</tr>
<tr>
<td>DoS</td>
<td>Daniel-of-Service</td>
</tr>
<tr>
<td>DEI</td>
<td>Drop eligibility indication</td>
</tr>
<tr>
<td>DP</td>
<td>Display port</td>
</tr>
<tr>
<td>EtherCAT</td>
<td>Ethernet for control automation technology</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic control units</td>
</tr>
<tr>
<td>FDB</td>
<td>Forwarding database</td>
</tr>
<tr>
<td>FQTSS</td>
<td>Forwarding and queuing enhancements for time-sensitive streams</td>
</tr>
<tr>
<td>FMan</td>
<td>Frame manager</td>
</tr>
<tr>
<td>GPU</td>
<td>General processor unit</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet control message protocol</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of electrical and electronics engineers</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet engineering task force</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter-processor communication</td>
</tr>
<tr>
<td>KM</td>
<td>Key management</td>
</tr>
<tr>
<td>LBT</td>
<td>Latency and bandwidth tester</td>
</tr>
<tr>
<td>MAC</td>
<td>Medium access control</td>
</tr>
<tr>
<td>MU</td>
<td>Message Unit</td>
</tr>
<tr>
<td>NFC</td>
<td>Near field communication</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>NCI</td>
<td>NFC controller interface</td>
</tr>
<tr>
<td>NMT</td>
<td>Network management</td>
</tr>
<tr>
<td>OC</td>
<td>Ordinary clock</td>
</tr>
<tr>
<td>OpenIL</td>
<td>Open industry Linux</td>
</tr>
<tr>
<td>OPC</td>
<td>Open platform communications</td>
</tr>
<tr>
<td>OP-TEE</td>
<td>Open portable trusted execution environment</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>OTA</td>
<td>Over-the-air</td>
</tr>
<tr>
<td>OTPMK</td>
<td>One-time programmable master key</td>
</tr>
<tr>
<td>PCP</td>
<td>Priority code point</td>
</tr>
<tr>
<td>PDO</td>
<td>Process data object</td>
</tr>
<tr>
<td>PHC</td>
<td>PTP hardware clock</td>
</tr>
<tr>
<td>PIT</td>
<td>Packet inter-arrival times</td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable logic controller</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision time protocol</td>
</tr>
<tr>
<td>QSPI</td>
<td>Queued serial peripheral interface</td>
</tr>
<tr>
<td>RCW</td>
<td>Reset configuration word</td>
</tr>
<tr>
<td>REE</td>
<td>Rich execution environment</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote procedure call</td>
</tr>
<tr>
<td>RPMSG</td>
<td>Remote processor messaging</td>
</tr>
<tr>
<td>RTEdge</td>
<td>Real-time edge</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-time clock</td>
</tr>
<tr>
<td>RTT</td>
<td>Round-trip times</td>
</tr>
<tr>
<td>RX</td>
<td>Receiver</td>
</tr>
<tr>
<td>SABRE</td>
<td>Smart application blueprint for rapid engineering</td>
</tr>
<tr>
<td>SDO</td>
<td>Service data object</td>
</tr>
<tr>
<td>SOEM</td>
<td>Simple Open EtherCAT master</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial periphery interface</td>
</tr>
<tr>
<td>SRP</td>
<td>Stream reservation protocol</td>
</tr>
<tr>
<td>SRTM</td>
<td>Simplified Real-time Messaging</td>
</tr>
<tr>
<td>SRK</td>
<td>Single root key</td>
</tr>
<tr>
<td>TA</td>
<td>Trusted application</td>
</tr>
<tr>
<td>TAS</td>
<td>Time-aware scheduler</td>
</tr>
<tr>
<td>TC</td>
<td>Traffic classification</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission control protocol</td>
</tr>
<tr>
<td>TEE</td>
<td>Trusted execution environment</td>
</tr>
</tbody>
</table>
Table 3. Acronyms and abbreviations...continued

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFTP</td>
<td>Trivial file transfer protocol</td>
</tr>
<tr>
<td>TSN</td>
<td>Time sensitive networking</td>
</tr>
<tr>
<td>TX</td>
<td>Transmitter</td>
</tr>
<tr>
<td>TZASC</td>
<td>Trust zone address space controller</td>
</tr>
<tr>
<td>UDP</td>
<td>User datagram protocol</td>
</tr>
<tr>
<td>VLAN</td>
<td>Virtual local area network</td>
</tr>
</tbody>
</table>
2 Release notes

2.1 What's new

The following sections describe the new features for each release.

2.1.1 What's new in Real-time Edge software v2.7

• **Real-time System**
  – Preempt-RT Linux 6.1.36-rt12
  – Baremetal: math library extended to all platforms
  – Harpoon 2.5

• **Heterogeneous multcore Framework**
  – lwIP on Cortex-A Core (ENET on i.MX8MP)
  – RPMSG between two FreeRTOS
  – RPMSG performance evaluation tools
  – RAM console on FreeRTOS
  – Flexible bootstraps with application

• **Heterogeneous Multi-SoC Framework**
  – NETC DSA switch driver on Linux
  – Device driver of DSA control interface on Linux DSA
  – Service driver of DSA control interface on i.MX RT1180
  – NETC DSA switch configuration on i.MX RT1180

• **Protocols**
  – AVB bridge with SJA1105

• **Benchmarking**
  – Heterogeneous Multicore VirtIO performance optimization

• **NPI**
  – i.MX93 A1 9*9
    – Preempt RT, Baremetal, Jailhouse, Heterogeneous Multicore (RPMSG, UART sharing), TSN web-UI configuration

• **Based on if-6.1.36-2.1.0**
  – U-Boot v2023.04
  – LTS 6.1.36
  – MCUX SDK 2.13.1
  – Yocto mickledore 4.2

• **Documentation**
  – Real-time Edge QRG (Quick Reference Guide)

2.1.2 What's new in Real-time Edge software v2.6

• **Real-time system**
  – Preempt-RT Linux 6.1.22-rt8
  – Baremetal
    – LS1028A
      – Preempt-RT Linux + Baremetal
      – All Cortex-A running under Baremetal
– Math library support
– DM mode for Baremetal example and driver: I2C, QSPI
– Harpoon 2.4.0

**Heterogeneous multicore framework**
– VirtIO Ethernet sharing RFP

**Protocols**
– EtherCAT master CodeSYS networking optimization
– i.MX 8M Plus, i.MX 8M Mini, i.MX93, i.MX 6ULL
– TSN
– Enhancements for Avnu Alliance Conformance Test for IEEE802.1Qbu/IEEE802.3br
– AVB Milan 1.1 Test Suite conformance

**Benchmark**
– Heterogeneous multicore performance: networking
– CodeSYS EtherCAT master stack benchmarking

**NPI**
– i.MX93 A0 9*9
– Preempt RT, TSN, TSN stack, and config tool
– i.MX8DXL: AVB Media Clock Recovery

**Based on IF-6.1.22-2.0.0**
– U-Boot v2023.04
– LTS 6.1.22
– MCUX SDK 2.13.1
– Yocto mickledore 4.2

### 2.1.3 What's new in Real-time Edge software v2.5

**Real-time system**
– Heterogeneous multi-core
– RPMSG Vring buffer increasing from 256 KB to 8 MB
– VirtIO network sharing with performance optimization
– Baremetal improvements on LS1046A
– All Cortex-A cores running under Baremetal
– Flextimer
– Baremetal example and driver change to DM mode: GPIO
– Integration of Harpoon 2.3
– Support for AVB Talker in FreeRTOS audio app
– Support for RPMsg control (FreeRTOS, all boards)
– Support for Virtual Ethernet
– Basic support for i.MX 93 ("hello world")

**Protocols**
– EtherCAT master
– Basic CodeSYS PLC control support and native driver optimization
– i.MX 8M Plus, i.MX 8M Mini, i.MX93, i.MX 6ULL

**NPI**
– i.MX93 A0 11*11
– Baremetal, RPMSG based UART sharing
– AVB Media Clock Recovery
• i.MX8DXL: AVB audio talker/listener

• Platform
  – eMMC booting on LS1028ARDB and LS1046ARDB
  – Removal of testing and documentation support, keeping code inclusion for:
    – LS1021AIOT, LS1021ATSN, LS1021ATWR, LS1012ARDB
  – Based on If-5.15.71-rt-2.2.0
    – LTS 5.15.71
    – Yocto Kirkstone 4.0
    – U-Boot v2022.04

2.1.4 What's new in Real-time Edge software v2.4

• Real-time system
  – Preempt-RT Linux-5.15.52-rt
  – Heterogeneous multi-core
    – Inter-core communication between Cortex-A core and Cortex-A/Cortex-M core on i.MX 8M Plus and i.MX 8M Mini
      – UART 9-bit Multidrop mode (RS-485) support
      – RPMSG between Cortex-A cores
      – Linux SGI mailbox driver on Linux
      – RPMSG Lite with SGI mailbox on RTOS
    – Loading binaries on i.MX 8M Mini and i.MX 8M Plus to the Cortex-M from Linux
  – Baremetal extensions on LS1046A
    – Single hardware interrupt routed to multiple cores
    – Newlib math library
    – Integration of Harpoon 2.2
    – Audio SMP pipeline (Zephyr)
    – RPMsg-based IPC through Linux control application
    – audio AVB pipeline (FreeRTOS)
    – Support for AVB Listener in FreeRTOS audio app

• Real-time Networking
  – TSN
    – Dynamic TSN configuration of Qci for bandwidth limitation
    – Qbu: added preemption TLV on LLDP package and preemption verification support
  – AVB
    – AVB integration improvements

• Protocols
  – AVB Milan extensions
  – EtherCAT master stacks
    – EtherCAT master multiple axes control system
  – HMI: LS1028A and i.MX 8M Plus
    – HTML5/chromium
  – Modbus
    – Libmodbus package integration
    – Modbus-simulator client and server
– WiFi enabled on i.MX 8DXL

**Reference design**
– EtherCAT master multiple axes control system
  – HCFA 60-axes servo using CSP mode

**NPI**
– i.MX93 A0 11*11: Preempt-RT, EtherCAT master, AVB/TSN, TSN stack and config tools, TSN performance, OPC-UA Pub/Sub
– i.MX8DXL: Preempt-RT, EtherCAT master, TSN stack and config tools, OPC-UA Pub/Sub

**Based on If-5.15.52-2.1.0**
– LTS 5.15.52
– Yocto Kirkstone 4.0
– U-boot v2022.04

2.1.5 What’s new in Real-time Edge software v2.3

**Real-time Networking**
– TSN
  – Dynamic TSN configuration (EAR)
    – Qci configuration
    – CAF configuration based on 802.1 Qch
  – YANG modules updating to latest version
– AVB
  – Endpoint support on i.MX 6ULL, i.MX 8M Plus, and i.MX 8M Mini

**Real-time System**
– PREEMPT-RT Linux-5.15.5-rt22
– Heterogeneous AMP software
  – Yocto based unified delivery for Cortex-A and Cortex-M
  – Resource sharing
    – RPMSG-based UART sharing
      – Virtual UART to physical UART 1:1 mapping
      – Virtual UART to physical UART n:1 mapping
      – Virtual UART to physical UART flexible mapping
– Harpoon (RTOS on Cortex-A)
  – Zephyr integration on i.MX 8M Plus and i.MX 8M Mini
  – Audio Application
    – Sine wave playback
    – Record and playback (loopback)
    – Audio pipeline
  – Industrial applications:
    – TSN over Ethernet test application
    – CAN test application

**Protocols**
– EtherCAT master stack
  – IGH EtherCAT master native driver on LS1043A and LS1046A
  – Multiple EtherCAT masters
  – Flexible port selection for EtherCAT and Ethernet
  – SOEM EtherCAT master stack enablement (PRC):
    – RTOS on Cortex-M on i.MX 8M Plus
2.1.6 What’s new in Real-time Edge software v2.2

• Real-time Networking
  – TSN
    – 802.1AS: PHY delay correction calibration
    – AF_XDP performance improvements
    – IEEE 1588 PTP UDP on LS1028ARDB TSN switch
• Real-time system
  – PREEMPT-RT Linux-5.10.72-rt53
  – Harpoon (RTOS on Cortex-A)
    – Integration of Harpoon on i.MX 8M Plus and i.MX 8M Mini
• Protocols
  – EtherCAT master stack
    – IGH EtherCAT master native driver on LS1043A and LS1046A
    – Multiple EtherCAT masters
    – Flexible port selection for EtherCAT and Ethernet
    – SOEM EtherCAT master stack enablement (EAR):
      – RTOS on Cortex-M on i.MX 8M Plus
      – FreeRTOS
        or without an operating system
• Based on lf-5.10.72-2.2.0
  – Linux 5.10.72-rt
  – U-Boot v2021.04
  – Yocto Hardknott 3.3
• Real-time Networking
  – TSN
    – 802.1AS-2020
      – CMLDS (generic interface to PTP stack)
    – TSN application
      – TSN application with AF_XDP data path
    – TSN configuration
      – Path selection for Qbv
      – Schedule mapping for Qbv
  – Real-time system
    – PREEMPT-RT Linux-5.10.52-rt47
      – Jailhouse
        – GPIO in non-root cell Linux support on LS1028ARDB
        – ENETC in non-root cell Linux support on LS1028ARDB
  – Protocols
    – Native EtherCAT-capable network driver module on ENETC (LS1028ARDB)
    – Native EtherCAT-capable network driver module on FEC (i.MX 8M Plus EVK)
    – EtherCAT: CoE 6-8 axis control
    – OPC UA PubSub
    – OPC UA PubSub over TSN
  – Based on i.MX L5.10.52_2.1.0
    – Linux 5.10.52-rt
    – U-Boot v2021.04
    – Yocto Hardknott 3.3

2.1.8 What's new in Real-time Edge software v2.0
• Based on Yocto project 3.2 (Gatesgarth)
• Real-time System
  – PREEMPT-RT Linux
  – Heterogeneous architecture
    – Baremetal: PREEMPT-RT Linux on A core + Baremetal architecture on A core
      – i.MX 8M Plus EVK, i.MX 8M Mini EVK, LS1028ARDB, LS1046ARDB, LS1043ARDB, LS1021A-IoT
    – Jailhouse: PREEMPT-RT Linux on A core + Jailhouse + PREEMPT-RT Linux on A core
      – i.MX 8M Plus EVK, LS1028ARDB, LS1046ARDB
• Real-time Networking
  – TSN
    – TSN Standards
      – IEEE 802.1Qav
      – IEEE 802.1Qbv
      – IEEE 802.1Qbu
      – IEEE 802.1Qci
      – IEEE 802.1CB
      – IEEE 802.1AS-2020 (gPTP)
      – IEEE 802.1Qat-2010 (SRP)
    – TSN Configurations
      – Linux tc command and tsntool
      – NETCONF/YANG
– Dynamic TSN configuration - web-based TSN configuration, dynamic topology discovery
– TSN Applications
  – Example for real-time traffic processing
– Networking
  – 802.1 Q-in-Q
  – VCAP tc flower chain mode
  – Priority set, VLAN tag push/pop/modify, Policer Burst and Rate Configuration, drop/trap/redirect

• Industrial
  – EtherCAT master
  – IGH EtherCAT master stack
  – Native EtherCAT-capable network driver module (i.MX 8M Mini EVK)
  – FlexCAN
  – SocketCAN on Linux kernel
  – CANOpen
  – CANOpen master and slave example code
  – CoE: CANOpen over EtherCAT
    – CiA402(DS402) profile framework based on IGH CoE interface
    – EtherCAT CoE 6-8 axis control (i.MX 8M Mini EVK)
  – OPC UA/OPC UA PubSub
    – open62541
  – Modbus
    – Modbus master and slave
    – Modbus-RTU
    – Modbus-TCP
    – Modbus-ASCII

• New Added Platform
  – i.MX 6ULL EVK

2.1.9 What's new in OpenIL v1.11

What's New:
• TSN
  – 802.1AS-2020
    – Initial support for multi-domain on i.MX 8M Plus and LS1028A
• Hardware
  – i.MX 8M Plus silicon A1
• Linux Kernel
  – LTS 5.4.70 for i.MX 8 Series
• U-Boot
  – v2020.04 for i.MX 8 Series
• Baremetal
  – v2020.04 for Layerscape and i.MX 8 Series
  – i.MX 8M Plus EVK

2.1.10 What's new in OpenIL v1.10

What's New:
• **TSN**
  – VCAP chain mode
  – GenAVB/TSN stack

• **Real-time**
  – PREEMPT-RT 5.4 on i.MX 8M Mini
    – Ethernet
    – PCIe
    – GPIO
    – DSI

• **Baremetal**
  – i.MX 8M Mini EVK (A core to A core)
    – ICC
    – Ethernet
    – GPIO

• **OpenIL framework**
  – Board
    – i.MX 8M Mini platform
      – GPU: OpenGL ES
       – Display: OpenGL ES, Weston, DSI-MIPI, CSI-MIPI

2.1.11 What's new in OpenIL v1.9

**What's New:**

• **TSN**
  – tc flower support for Qbu and Qci
  – 802.1 QinQ
  – Multi-ports TSN switch solution
  – i.MX 8M Plus - TSN

• **Real-time**
  – PREEMPT-RT 5.4 on i.MX 8M Plus

• **Baremetal**
  – LX2160ARDB rev2 support and ICC

• **OpenIL framework**
  – linuxptp uprev to 3.0
  – Board
    – i.MX 8M Plus EVK
      – TSN: Qbv, Qbu, Qav
       – GPU: OpenGL ES, OpenCL
       – Display: OpenGL ES, Weston
      – LS1028ARDB
        – Display: OpenGL ES, Weston
        – GPU: OpenGL ES, OpenCL
        – LX2160ARDB Rev2

2.1.12 What's new in OpenIL v1.8

**What's New:**
• TSN
  – tc VCAP support for VLAN-retagging
  – tc VCAP support for police
  – tc support for Qav and Qbv
  – SJA1105 DSA Support and clock synchronization
  – YANG modules for network config (IP, MAC, and VLAN)
• Real time
  – PREEMPT-RT 5.4
• Baremetal
  – LX2160A rev1 ICC
• OpenIL framework
  – buildroot uprev to 2020.02
  – Kernel/U-Boot
    – Linux upgraded to LSDK20.04 - Linux-5.4.3
    – U-Boot upgraded to LSDK20.04 - U-Boot 2019.10
  – Board
    – i.MX 8M Mini
    – Foxconn LS1028ATSN board with SJA1105

2.1.13 What's new in OpenIL v1.7

What's New:
• TSN
  – BC-based 802.1AS bridge mode
  – Netopper2 support based on sysrepo. Support Qbv, Qbu, Qci configuration
  – VLAN-based tc flower policer
  – Web-based TSN configuration tool - available for Qbv, Qbu, and Qci configuration
• Real time
  – Xenomai
    – Xenomai I-pipe uprev to 4.19
  – Baremetal
    – SAI support on LS1028
      – i.MX6Q Baremetal ICC
• Industrial protocols
  – CANopen over EtherCAT
• OpenIL framework
  – Kernel/U-Boot
    – Linux upgraded to LSDK1909 - 4.19
    – U-Boot upgraded to U-Boot-2019.04
  – Boards
    – LX2160ARDB SD boot
    – LX2160ARDB XSPI boot
    – LS1028ARDB XSPI boot
    – LS1046ARDB eMMC boot

2.1.14 What's new in OpenIL v1.6

What's New:
• TSN
  – Web-based TSN configuration tool - available for Qbv and Qbu configuration
  – TSN driver enhancement
• Real time
  – Baremetal
    – i.MX6Q-sabresd Baremetal support
• NETCONF/YANG
  – NETCONF/YANG model for Qbu and Qci protocol
• Industrial protocols
  – LS1028A - BEE click board

2.1.15 What's new in OpenIL v1.5

What's New:
• TSN
  – Web-based TSN configuration tool - available for Qbv and Qbu configuration
  – 802.1AS endpoint mode for LS1028A TSN switch
• Real time
  – Xenomai
    – LS1028 ENETC Xenomai RTNET support
  – Baremetal
    – LS1028 Baremetal ENETC support
• NETCONF/YANG
  – NETCONF/YANG model for Qbu protocol
• Industrial protocols
  – LS1028A - BLE click board

2.1.16 What's new in OpenIL v1.4

What's New:
• TSN
  – ENETC TSN driver: Qbv, Qbu, Qci, Qav
  – ENETC 1588 two steps timestamping support
  – SWTICH TSN driver: Qbv, Qci, Qbu, Qav, 802.1CB support
• Real time
  – Xenomai
    – LS1028ARDB
  – Baremetal
    – LS1021AloT, LS1043ARDB, LS1046ARDB
    – LS1028 Baremetal basic Baremetal support
• Industrial protocols
  – LS1028A - NFC click board
    – QT5.11
• OpenIL framework
  – boards: LS1028ARDB
### 2.2 Feature support matrix

*Table 4* shows the features that are supported in this release.

**Table 4. Key features**

<table>
<thead>
<tr>
<th>Feature</th>
<th>LMX 6ULL 14x14 EVK</th>
<th>LMX 8DXL LPDDR4 EVK</th>
<th>LMX 8M Mini LPDDR4 EVK</th>
<th>LMX 8M Plus LPDDR4 EVK</th>
<th>LMX 93 9x9 LPDDR4 QSB</th>
<th>LS1028 ARDB</th>
<th>LS1043 ARDB</th>
<th>LS1046 AFRWY</th>
<th>LS1046 ARDB</th>
<th>LS1046 AFRWY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot mode</td>
<td>SD</td>
<td>eMMC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preempt-RT Linux</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>PCIe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethernet</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>GPIO</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPI</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>UART</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>USB</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2C</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QSPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flextimer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux (communication with Baremetal)</td>
<td>ICC</td>
<td>ICC</td>
<td>ICC</td>
<td>ICC</td>
<td>ICC</td>
<td>ICC</td>
<td>ICC</td>
<td>ICC</td>
<td>ICC</td>
<td>ICC</td>
</tr>
<tr>
<td>Single HW Interrupt to multiple cores</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Newlib Math library</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>All Cortex-A cores running under Baremetal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jailhouse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Harpoon (RTOS on Cortex-A)</td>
<td>FreeRTOS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Zephyr</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Flexible Real-time System</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flexible Life Cycle Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U-Boot booting Native RTOS/Baremetal A Core Image</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>U-Boot booting NativeRTOS/Baremetal MCore Image</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Linux booting Native RTOS/Baremetal MCore Image</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RPMSG between A Core Linux and M-Core RTOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPMSG between A Core Linux and A-Core RTOS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
### Table 4. Key features...continued

<table>
<thead>
<tr>
<th>Feature</th>
<th>LMX 6ULL 14x14 EVK</th>
<th>LMX 8DXL LPDDR4 EVK</th>
<th>LMX 8M Mini Plus LPDDR4 EVK</th>
<th>LMX 93 LPDDR4 ARDB</th>
<th>LS1028 ARDB</th>
<th>LS1043 ARDB</th>
<th>LS1046 AFRWY</th>
<th>LX2160 ARDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPMSG between ACore and A-Core RTOS</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPMSG between A core and M core with enhanced RMB buffer</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPMSG Performance Evaluation</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART Sharing based on RPMSG</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous Multicore VirtIO Performance Evaluation</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous Multicore VirtIO Network Sharing</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSA single port mode</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSA bridge mode</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MTU configuration</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLAN configuration</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FDB configuration</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Port statistics</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux tc command</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Qbv</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Qbu</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Qci</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qav</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>802.1AS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>802.1CB</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCAP chain mode</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>802.1 Q-in-Q</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSN tool</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qbv</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Qbu</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Qci</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLAN config</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web-based configuration</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Qbv</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qbu</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qci</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic topology discovery</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Qci</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CQF</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Qbv</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
### Table 4. Key features...continued

| Feature | LMX 6ULL 14x14 EVK | LMX 8DXL LPDDR4 EVK | LMX 8M Mini LPDDR4 EVK | LMX 8M Plus LPDDR4 EVK | LMX 93 9x9 LPDDR4 EVK | LMX 93 9x9 QSB | LS1028 ARDB | LS1043 ARDB | LS1046 AFRWY | LS1046 ARDB | LS2160 ARDB |
|---------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------|-------------|-------------|-------------|-------------|-------------|
| AVB standards | | | | | | | | | | | | |
| AV/TP Talker/Listener | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| AV/DECC | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| MAAP | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Milan | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| AVB Bridge on SJA1105Q-EVB | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| IEEE 1588/802.1AS | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Industrial Protocol | | | | | | | | | | | | |
| EtherCAT master | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| IGH EtherCAT master stack | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| IGH native Ethernet device driver | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| SOEM | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| CodeSYS EtherCAT master stack | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| FlexCAN | | | | | | | | | | | | Y |
| CANopen | | | | | | | | | | | | |
| OPC UA | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| open62541 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| OPC UA Pub/Sub over TSN | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| BEE (Mikroe Click board) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| BLE (Mikroe Click board) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| NFC (Mikroe Click board) | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Modbus | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Modbus-RTU | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Modbus-TCP | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |

[1] Media clock recovery is implemented through a software-based sampling
2.3 Open, fixed, and closed issues

This section contains three tables that describe Open, Fixed, and Closed issues.

- Open issues do not currently have a resolution. Workaround suggestions are provided where possible. Refer Table 5.
- Fixed issues have a software fix that has been integrated into the 'Fixed In' Release. Refer Table 6.
- Closed issues are issues where the root cause and fix are outside the scope of Real-time Edge Software. Disposition is to provide the explanation. Refer Table 7.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Opened In</th>
<th>Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDLINUX-3750</td>
<td>logout after listing services by running 'service --status-all' on LS1028ARDB</td>
<td>Real-time Edge software v2.6</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Opened In</th>
<th>Fixed In</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDLINUX-3632</td>
<td>Graphics feature is not functioning on LS1028ARDB</td>
<td>Real-time Edge software v2.6</td>
<td>Real-time Edge software v2.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Opened In</th>
<th>Disposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
3 Real-time system

3.1 Overview

Real-time System is to address different Real-time requirements on multicore platforms, including different schedule latency requirements, Inter-Core communication, hardware resource sharing, unified life-cycle management, and unified building and deployment mechanism.

For different schedule latency requirements, Real-time System provides Preempt-RT Linux, native RTOS on Cortex-A core and Cortex-M core, RTOS on Cortex-A core with Jailhouse, Baremetal framework, and a flexible combination of different cores running Preempt-RT Linux and RTOS/Baremetal to meet different Real-time requirements for different use cases.

Heterogeneous Multicore Framework in Real-Rime Edge provides different inter-core communication mechanisms and hardware resources sharing mechanisms between different CPU Core and different OS to cover high-performance communication and Real-time requirements. Unified CPU Core life cycle management provides a unified mechanism to bootstrap the Cortex-A core and Cortex-M core on the heterogeneous MPU system. Refer to chapter Section 4 for more details.

The unified software building and deploying mechanism provides easy building and deployment for the software components running on the Cortex-A core and Cortex-M core. These components and features of this mechanism are depicted in Figure 2.

Figure 2. Real-time system

Real-time Edge software provides a general software architecture to run Real-time Systems on MPU platforms with the following features:

• Different frameworks and flexible combinations for different schedule latency requirements
  The Figure 3 shows all the OS/Baremetal supported in Real-time Edge on MPU platforms with different-scale schedule latency:
  – Preempt-RT Linux on Cortex-A Core

All information provided in this document is subject to legal disclaimers.
© 2023 NXP B.V. All rights reserved.
User guide Rev. 2.7 — 18 December 2023
Real-time Linux kernel provides deterministic low latency compared to Linux.

- **Baremetal on Cortex-A Core**
  Single or multiple Baremetal instance run Cortex-A Core(s) with zero schedule latency.

- **Native RTOS SMP/AMP on Cortex-A Core**
  Native RTOS (FreeRTOS or Zephyr) is kicked to one or more Cortex-A Core from U-Boot, no Jailhouse is used and targets lower latency and higher performance as compared to RTOS with Jailhouse.

- **RTOS SMP/AMP on Cortex-A Core with Jailhouse**
  RTOS (FreeRTOS or Zephyr) runs in Jailhouse inmate with hardware resource isolations on Cortex-A Core.

- **RTOS and Baremetal on Cortex-M Core**
  Generally used for Real-time Control system, but has less CPU computing ability than Cortex-A Core.

All these Real-time OS or BareMetal can be combined to be a flexible AMP system on multicore system. For example, i.MX8M Plus platform has four Cortex-A53 Core and one Cortex-M7 Core. The Real-time Edge software supports flexible AMP system to run these OS/BareMetal combinations:

- Four Cortex-A53 cores run SMP Preempt-RT Linux, Cortex-M7 core run RTOS.
- Four Cortex-A53 cores run four Baremetal/RTOS instances, Cortex-M7 core run RTOS.
- One or more Cortex-A53 cores run Preempt-RT Linux, the other Cortex-A53 cores run one or more Baremetal/RTOS instances, Cortex-M7 core run RTOS.
- One or more Cortex-A53 cores run Preempt-RT Linux as Jailhouse Root Cell, the other Cortex-A53 cores run as one or more inmate cell(s) with RTOS.

**Unified Software Building, Deploy and Release**

- All different OS/application running on different cores are built via Yocto.
- A bitbake command is used to create all images on different cores.
- Single Flash Image includes all OS/applications running on all CPU cores.

**Heterogeneous Multicore Framework**
A common framework with the following key features and functions:

– **Inter-Core Data Communication and Resource Sharing**
  Common Heterogeneous Multicore Framework provides data communication and resource sharing between M-Core and A-Core(s) or between different A-Cores simultaneously: RPMSG provides standard message communication for low bandwidth use cases, and Heterogeneous Multicore Virtio provides high performance data path and resource sharing to meet high bandwidth requirement.

– **Unified Life-Cycle management for flexible AMP**

Real-time Edge software supports Preempt-RT Linux, FreeRTOS, Zephyr, and Baremetal running on different processors with Heterogeneous Multicore Framework. The Table 8 shows the support matrix on NXP platforms:

<table>
<thead>
<tr>
<th>Real-time System</th>
<th>i.MX 8M Mini LPDDR4 EVK</th>
<th>i.MX 8M Plus LPDDR4 EVK</th>
<th>i.MX 93 EVK</th>
<th>LS1028 ARDB</th>
<th>LS1043 ARDB</th>
<th>LS1046 ARDB</th>
<th>LX2160ARDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>4 X A53 1 X M4</td>
<td>4 X A53 1 X M7</td>
<td>2 X A53 1 X M33</td>
<td>2 X A72 4 X A53 4 X A72</td>
<td>16 X A72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preempt-RT Linux</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Baremetal</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Native RTOS</td>
<td>FreeRTOS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Zephyr</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Jailhouse</td>
<td>Baremetal</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>FreeRTOS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Zephyr</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Harpoon</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

### 3.2 Building, deploying, and releasing unified software

The Yocto project is an open source collaboration project that helps developers create custom Linux-based systems regardless of the hardware architecture. The project provides a flexible set of tools and a space where embedded developers worldwide can share technologies, software stacks, configurations, and best practices. These can further be used to create tailored Linux images for embedded and IoT devices, or anywhere a customized Linux OS is needed. Moreover, Linux factory selects Yocto as the building tool. Real-time Edge also selects Yocto as the unified SW release tool. Figure 4 shows the unified Yocto structure for Heterogeneous AMP.
• Yocto Layer meta-real-time-edge focuses on Linux and baremetal application building on Cortex-A core.
• Yocto Layer meta-nxp-harpoon focuses on RTOS building on Cortex-A core.
• Yocto Layer meta-rtos-industrial focuses on RTOS building running on Cortex-M core.

Only one build command is required to generate the complete image, including all binaries running on core A and core M.

For example:

```bash
# setup yocto environment for imx8mp-lpddr4-evk board
$ DISTRO=nxp-real-time-edge MACHINE=imx8mp-lpddr4-evk source real-time-edge-setup-env.sh
# build all images for imx8mp-lpddr4-evk board
$ bitbake nxp-image-real-time-edge
```

### 3.2.1 Yocto layer for Cortex-A core

The Cortex-A core allows users to run Linux, Jailhouse, Baremetal, and RTOS. The corresponding Yocto layer description is as follows:

1. **Linux and Rootfs**
   
   The Yocto layer meta-real-time-edge focuses on Linux building on Cortex-A cores. This layer is based on Linux factory and describes the process for building all applications for Linux and rootfs on Cortex-A core.

2. **Jailhouse**
   
   The scripts under meta-real-time-edge/recipes-extended/real-time-edge-jailhouse describe how to build Jailhouse running on Cortex-A core.

3. **Baremetal application**
   
   The scripts under meta-real-time-edge/recipes-extended/real-time-edge-baremetal describe how to build baremetal application on Cortex-A core. Refer to Section 3.4 for details.

4. **Harpoon (RTOS on A core)**
   
   Harpoon provides an environment for developing real-time demanding applications on an RTOS running on one (or several) Cortex-A core(s) in parallel of a Linux distribution, leveraging the 64-bit Arm (R) architecture for higher performance. The system starts on Linux and the Jailhouse partitions the hardware to run both Linux and the guest RTOS in parallel. The hardware partitioning is configurable and depends on the use case. The Yocto layer meta-nxp-harpoon describes how to build these applications on Cortex-A core. For more information, refer to Harpoon User's Guide. See Section 1.4.

### 3.2.2 Yocto layer for Cortex-M core

When the application runs on the Cortex-M core, it uses different toolchain and source code. For a unified compilation interface, Yocto meta layer meta-rtos-industrial is introduced into Real-time Edge project. The meta-rtos-industrial layer provides the build environment to create MCUX SDK application for Cortex-M cores.

#### 3.2.2.1 Introduction to meta-rtos-industrial

The Figure 5 shows the meta-rtos-industrial file structure.
3.2.2.1.1 Source code definition

All source code related definition is under `recipes-kernel/mcxu-kernel` folder.

The file `mcux-sdk-src.inc` defines all the repos of [NXPMicro/mcux-sdk: MCUXpresso SDK (github.com)](https://github.com) and the new repos.

If a new repo needs to be downloaded, append a new line to "SRC_URI" with the URL and location of the required repo.

For example, use the below code to download 'SOEM' stack to `git/core/components/SOEM` folder:

```
git://$(NXPMICRO_BASE)/soem.git;protocol=https;nobranch=1;destsuffix=git/core/components/SOEM;name=SOEM 
```

`mcux-sdk-src-XXX.inc` defines the MCUX SDK repo commit ID for the release XXX. For example, `mcux-sdk-src-2.11.0.inc` contains all the commit IDs for the release 2.11.0 repository.

The parameter `PREFERRED_VERSION_MCUX-SDK` defines the default version in `mcux-sdk-src.inc`. If you want to compile a different version, overwrite this parameter in the `local.conf`.

For example:

```
# Add the below line into local.conf
PREFERRED_VERSION_MCUX-SDK = "2.10.0"
```

3.2.2.1.2 Example definition

The file `mcux-examples.inc` describes the common method to compile install and deploy examples. Each example `bb` file should include this file and then specify the folder of the example.
Use the command below to add a new example:

```bash
include mcux-example.inc
MCUX_EXAMPLE_DIR = "examples/${RTOS-INDUSTRIAL-BOARD}/demo_apps/hello_world"
```

### 3.2.2.1.3 Toolchain definition

The file `recipes-devtools/external-arm-toolchain/gcc-arm-none-eabi_VERSION.bb` describes how to download, install, and deploy `gcc-arm-none-eabi` toolchain of the specific `VERSION`.

This layer also supports external toolchain. Parameter "ARMGCC_DIR" can be overwritten to point the external toolchain.

For example:

```bash
ARMGCC_DIR = "/MYPATH/arm-none-eabi"
```

### 3.2.2.2 Integration of meta-rtos-industrial

To integrate `meta-rtos-industrial` into the Real-time Edge project, you need to specify the board and examples.

The board name is different between i.MX SDK and MCUX SDK. For example, in order to compile Cortex-M application for i.MX 8M Mini EVK with LPDDR4, use the board name `evkmimx8mm` instead of `imx8mm-lpddr4-evk`. The file `rtos-industrial-examples.inc` is created under `meta-real-time-edge/distro/include` to map the board names. The board name used by MCUX SDK should be set to parameter `RTOS-INDUSTRIAL-BOARD`.

In the path `meta-real-time-edge/recipes-nxp/packagegroups`, `packagegroup-real-time-edge-rtos.bb` is used for examples that are compiled. These examples should be installed into `/examples`.

### 3.2.2.3 Building meta-rtos-industrial

As the meta-rtos-industrial is already integrated into Real-time Edge, we do not need any special commands or settings to enable building the rtos application. When building `nxp-image-real-time-edge` image, all examples defined in `packagegroup-real-time-edge-rtos.bb` are built and installed into `/examples` folder in rootfs.

Use the below commands to create `nxp-image-real-time-edge` image for `imx8mm-lpddr4-evk` board.

```bash
$ mkdir yocto-real-time-edge
$ cd yocto-real-time-edge
  -b real-time-edge-mickledore -m real-time-edge-2.7.0.xml
$ repo sync
$ DISTRO=nxp-real-time-edge MACHINE=imx8mm-lpddr4-evk source real-time-edge-setup-env.sh -b build-imx8mpevk-real-time-edge
$ bitbake nxp-image-real-time-edge
```

The example binary are located under `tmp/deploy/images/imx8mm-lpddr4-evk/examples` and `/examples` of rootfs.

```bash
examples/
|-- heterogeneous-multicore
|  |-- hello-world-ca
|  |  |  `-- ddr_release
```
If you just want to compile a special example, you can use the following command:

For example:

```bash
$ DISTRO=nxp-real-time-edge MACHINE=imx8mm-lpddr4-evk bitbake demo-hello-world
```

### 3.3 Preempt-RT Linux

The Preempt-RT Linux option turns the kernel into a real-time kernel. It does so by replacing various locking primitives (for example, spinlocks and rwlocks) with preemptible priority-inheritance aware variants. The Preempt-RT Linux option also enforces interrupt threading and introduces mechanisms to break up long non-preemptible sections. This makes the kernel fully preemptible and brings most execution contexts under scheduler control. However, very low level and critical code paths (entry code, scheduler, low level interrupt handling) remain non-preemptible.

### 3.3.1 System Real-time Latency tests

The basic measurement tool for Real-time Linux is cyclictest.

#### 3.3.1.1 Running Cyclictest

Cyclictest provides statistics about the latencies of the system. It accurately and repeatedly measures the difference between the intended wake-up time of a thread and the time at which it actually wakes up. It can measure latencies in real-time systems caused by the hardware, the firmware, and the operating system.

Thomas Gleixner (tglx) wrote the original test, but several people had later contributed modifications. Cyclictest is part of the test suite, rt-tests. Clark Williams and John Kacur currently maintain Cyclictest.

**cyclictest:**

- Use the below command to perform Latency Test:

  ```bash
  $ cyclictest -p90 -h50 -D30m
  ```
Note: For detailed parameters of Cyclictest, refer to Cyclictest Web Page.

3.3.2 Real-time application development

This section describes the steps for developing the Real-time application.

Real-time Application: API, Basic Structure, Background:

- Basic Linux application rules are the same; Use the POSIX API.
- There is still a division of Kernel Space and User Space.
- Linux applications run in User Space.
- For details, refer to: https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/application_base

Real-time Application: Users can build it using the steps below:

- Using the cross-compiler example:
  ```bash
  $ arm-linux-gnueabihf-gcc <filename>.c -o <filename>.out -lrt -Wall
  ```

- Using the native compiler on a target example:
  ```bash
  $ gcc <filename>.c -o <filename>.out -lrt -Wall
  ```

Scheduling policies have two classes:

1. Completely Fair Scheduling (CFS)

- **SCHED_NORMAL** (traditionally called SCHED_OTHER): The scheduling policy that is used for regular tasks. Every task gets a so-called ‘nice value’. It is a value between -20 for the highest nice value and 19 for the lowest nice value. The average value of execution time of the task depends on the associated nice value.

- **SCHED_BATCH**: Does not preempt nearly as often as regular tasks. Hence, it allows tasks to run longer and make better use of caches, but at the cost of interactivity. This is well suited for batch jobs and optimized for throughput.

- **SCHED_IDLE**: This policy is even weaker than nice 19. However, it is not a true idle timer scheduler in order to avoid getting into priority inversion problems, which would deadlock the machine.

2. Real-time policies

- **SCHED_FIFO**: Tasks have a priority between 1 (low) and 99 (high). A task running under this policy is scheduled until it finishes or a higher prioritized task preempts it.

- **SCHED_RR**: This policy is derived from SCHED_FIFO. The difference with respect to SCHED_FIFO policy is that a task runs during a defined time slice (if it is not preempted by a higher prioritized task). It can be interrupted by a task with the same priority once the time slice is used up. The time slice definition is exported in procfs (/proc/sys/kernel/sched_rr_timeslice_ms).

- **SCHED_DEADLINE**: This policy implements the Global Earliest Deadline First (GEDF) algorithm. Tasks scheduled under this policy can preempt any task scheduled with SCHED_FIFO or SCHED_RR.

3.4 Baremetal on Cortex-A core

The following sections provide an overview of the Real-time Edge Baremetal framework on A core including:

- Features supported
- Getting started with Baremetal framework using the supported platforms:
  - NXP Layerscape platforms
  - i.MX 8M / i.MX 93 platforms.
It also describes how to run a sample Baremetal framework on the host environment and develop customer-specific applications based on Baremetal framework.

### 3.4.1 Baremetal framework

The Baremetal framework supports the scenarios that need low latency, real-time response, and high-performance. There is no OS running on the cores and customer-specific application runs on the core directly. The Figure 6 depicts the baremetal framework architecture.

![Baremetal framework architecture](image)

The main features of the Baremetal framework are as follows:

- Core0 runs as master and runs the Baremetal or the operating system such as Linux, Vxworks.
- Slave cores run the Baremetal application.
- Easy assignment of different IP blocks to different cores.
- Interrupts between different cores and high-performance mechanism for data transfer.
- Different UART for core0 and slave cores for easy debug.
- Communication via shared memory.

The master core0 runs the Baremetal under master mode. It then loads the Baremetal application to the slave cores and starts the Baremetal application. The Figure 7 depicts the boot flow diagram:
Figure 7. Baremetal framework boot flow diagram

The Table 9 lists the industrial IoT features supported by various NXP processors and boards.

Table 9. BareMetal features supported by NXP processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Board</th>
<th>Main features supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX 8M Mini</td>
<td>i.MX 8M Mini LPDDR4 EVK</td>
<td>UART, IPI, data transfer, Ethernet, GPIO</td>
</tr>
<tr>
<td>i.MX 8M Plus</td>
<td>i.MX 8M Plus LPDDR4 EVK</td>
<td>UART, IPI, data transfer, Ethernet, GPIO</td>
</tr>
<tr>
<td>i.MX 93</td>
<td>i.MX 93 EVK</td>
<td>UART, IPI, data transfer, Ethernet</td>
</tr>
<tr>
<td></td>
<td>i.MX 93 9x9 LPDDR4 QSB</td>
<td>UART, IPI, data transfer</td>
</tr>
<tr>
<td>LS1028A</td>
<td>LS1028ARDB</td>
<td>I2C, UART, ENETC, IPI, data transfer, SAI</td>
</tr>
<tr>
<td>LS1043A</td>
<td>LS1043ARDB</td>
<td>IRQ, IPI, data transfer, Ethernet, IFC, I2C, UART, FMan, USB, PCIe</td>
</tr>
<tr>
<td>LS1046A</td>
<td>LS1046ARDB</td>
<td>IRQ, IPI, data transfer, Ethernet, IFC, I2C, UART, FMan, QSPI, USB, PCIe, GPIO</td>
</tr>
<tr>
<td>LX2160A/Rev2</td>
<td>LX2160ARDB</td>
<td>UART, IPI, data transfer</td>
</tr>
</tbody>
</table>

Typical use cases are as follows:

1. Core0 as a master core runs Linux to manage slave cores and communicate with server. Slave cores run Baremetal application for real-time processing. Refer Figure 8.
2. All cores run BareMetal application for real-time processing. Refer Figure 9.

3.4.2 Getting started

This section describes how to set up the environment and run the Baremetal examples on slave cores (assuming that the core0 is the master core and the other cores are the slave cores).

3.4.2.1 Hardware and software requirements

In order to run baremetal framework scenarios, the following are required:
3.4.2.2 Hardware setup

This section describes the hardware setup required for the NXP boards for running the Baremetal framework examples.

3.4.2.2.1 i.MX 8M Mini LPDDR4 EVK and i.MX 8M Plus LPDDR4 EVK board

Follow the steps below.

1. **i.MX 8M Plus LPDDR4 EVK**: There is one USB MicroB Debug port on board. Four UART ports can be found when the MicroB cable connects to PC.

```
/dev/ttyUSB0
/dev/ttyUSB1
/dev/ttyUSB2
/dev/ttyUSB3
```

Use `/dev/ttyUSB2` for core0 (master core) and `/dev/ttyUSB3` for core1, core2, and core3 (slave cores).

2. **i.MX 8M Mini LPDDR4 EVK**: There is one USB MicroB Debug port on board. Two UART ports can be found when the MicroB cable connects to PC.

```
/dev/ttyUSB0
/dev/ttyUSB1
```

Use `/dev/ttyUSB1` for core0 (master core) and `/dev/ttyUSB0` for core1, core2, and core3 (slave cores).

3. **GPIO setup**

For GPIO test on i.MX 8M Plus LPDDR4 EVK, connect pin 7 and pin 8 of J21 by a jumper, as shown in Figure 10.
4. For GPIO test on i.MX 8M Mini LPDDR4 EVK, connect pin7 and pin8 of J1003 by a jumper as shown in Figure 11.
3.4.2.2 LS1028ARDB, LX2160ARDB, LS1043ARDB, or LS1046ARDB

If the Real-time Edge Baremetal framework is developed using one of the boards- LS1028ARDB, LX2160ARDB, LS1043ARDB, or LS1046ARDB, two serial cables are needed for connection. One serial cable is used for core0, to connect to UART1 port. The other cable is used for slave cores and connects to the UART2 port.

To support SAI feature on LS1028ARDB, set switch SW5_8 to "ON".

3.4.2.3 i.MX 93 EVK

On i.MX 93 EVK board, the USB Type-C connector (J1401) provides four UART ports when connected to PC using USB cable. The third port (LPUART1) is used for core0 (master core) and the fourth port (LPUART2) is used for core1 (slave core).
3.4.2.2.4  i.MX93 9x9 LPDDR4 QSB

On i.MX 93 9x9 LPDDR4 QSB board, the USB Type-C connector (J1708) provides four UART ports when connected to PC using USB cable. The third port (LPUART2) is used for core0 (master core) and the fourth port (LPUART3) is used for core1 (slave core).

3.4.2.3  Software building

There are two methods to build the Baremetal images:

- The first method is to compile the images in a standalone way, and is described in the following section.
- The second method is to build the Baremetal images using Real-time Edge framework. This method is described in the document, Real-time Edge Yocto Project User Guide in section "Building the image through Yocto".

3.4.2.3.1  Building Baremetal binary for slave cores

Perform the steps mentioned below:

1. Download the project source from the following path:
   https://github.com/nxp-real-time-edge-sw/real-time-edge-uboot.git
2. Check it out to the tag:
   • Real-Time-Edge-v2.7-baremetal-202312
3. Configure cross-toolchain on your host environment.
4. Then, run the following commands:

```bash
/* build Baremetal image for i.MX 8M Mini LPDDR4 EVK Rev.C board */
$ make imx8mm_evk_baremetal_slave_defconfig
$ make
/* build Baremetal image for i.MX 8M Plus LPDDR4 EVK board */
$ make imx8mp_evk_baremetal_slave_defconfig
$ make
/* build Baremetal image for i.MX 93 EVK board */
$ make imx93_11x11_evk_baremetal_slave_defconfig
$ make
/* build Baremetal image for i.MX 93 9x9 LPDDR4 QSB board */
$ make imx93_9x9_qsb_baremetal_slave_defconfig
$ make
/* build Baremetal image for LS1028ARDB board */
$ make ls1028ardb_baremetal_slave_defconfig
$ make
/* build Baremetal image with SAI for LS1028ARDB board */
$ make ls1028ardb_baremetal_slave_sai_defconfig
$ make
/* build Baremetal image for LS1043ARDB board */
$ make ls1043ardb_baremetal_slave_defconfig
$ make
/* build Baremetal image for LS1046ARDB board */
$ make ls1046ardb_baremetal_slave_defconfig
$ make
/* build Baremetal image for LX2160ARDB board */
$ make lx2160ardb_baremetal_slave_defconfig
$ make
```

5. Finally, the file `u-boot-dtb.bin` used for Baremetal is generated.

Follow Real-time Edge Software Yocto Project to get the code and build images for these platforms.
3.4.2.3.2 Building the image through Yocto

There are two methods to build the Baremetal images. The first method, which is used to compile the images in a standalone way, is described in Section 3.4.2.3. The other method is to build the Baremetal images using Real-time Edge software framework, and is described in this section.

The Real-time Edge software is designed for embedded industrial usage. It is an integrated Linux distribution for industry. With the current version, the Baremetal can be built and implemented conveniently.

3.4.2.3.2.1 Getting Real-time Edge software

The latest release is available at the following URL:
https://github.com/nxp-real-time-edge-sw/yocto-real-time-edge.git

Follow Yocto documentation "Real-time Edge Yocto Project User Guide" to get the code and build the image. Refer to Section 1.4.

3.4.2.3.2.2 Building the Baremetal images

This section describes the steps for building the Baremetal images for various boards. The steps described are applicable to the boards such as LS1043ARDB, LS1046ARDB, LX2160ARDB, i.MX 8M Plus LPDDR4 EVK, and i.MX 8M Mini LPDDR4 EVK board.

Building the Baremetal images for various boards

Run the following commands to build the final Baremetal image for Layerscape and i.MX platforms.

```bash
$ cd yocto-real-time-edge

For i.MX 93 EVK Baremetal image:

$ DISTRO=nxp-real-time-edge-baremetal MACHINE=imx93evk source real-time-edge-setup-env.sh -b build-imx93evk-bm

For i.MX 93 9x9 LPDDR4 QSB Baremetal image:

$ DISTRO=nxp-real-time-edge-baremetal MACHINE=imx93-9x9-lpddr4-qsb source real-time-edge-setup-env.sh -b build-imx93qsb-bm

For LS1028ARDB Baremetal image:

$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1028ardb source real-time-edge-setup-env.sh -b build-ls1028ardb-bm

For LS1043ARDB Baremetal image:

$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1043ardb source real-time-edge-setup-env.sh -b build-ls1043ardb-bm

For LS1046ARDB Baremetal image:

$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1046ardb source real-time-edge-setup-env.sh -b build-ls1046ardb-bm
```
For LX2160ARDB Baremetal image:

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=lx2160ardb-rev2 source real-time-edge-setup-env.sh -b build-lx2160ardb-bm
```

For i.MX 8M Plus LPDDR4 EVK Baremetal image:

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=imx8mp-lpddr4-evk source real-time-edge-setup-env.sh -b build-imx8mp-evk-bm
```

For i.MX 8M Mini LPDDR4 EVK Baremetal image:

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=imx8mm-lpddr4-evk source real-time-edge-setup-env.sh -b build-imx8mm-evk-bm
```

Then, use:

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

### 3.4.2.4 Booting up the Linux with Baremetal

Use the following steps to boot up the system with the images built from Real-time Edge software.

For platforms that can be booted up from the SD card, there are just two steps required to program the image into SD card.

1. Insert an SD card (at least 4 GB size) into any Linux host machine.
2. Find the image file in building directory (for example: ls1028ardb):
   ```
tmp/deploy/images/ls1028ardb/nxp-image-real-time-edge-ls1028ardb.wic.zst
   ```
3. Then, run the following commands:
   ```
   $ zstd -d nxp-image-real-time-edge-ls1028ardb.wic.zst
   $ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wic of=/dev/sdx bs=1M conv=fsync
   # or in some other host machine:
   $ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wic of=/dev/mmcblkx bs=1M conv=fsync
   # find the right SD Card device name in your host machine and replace the "sdx" or "mmcblkx".
   ```
4. Then, insert the SD card into the target board (for example ls1028ardb) and power on.

After completion of the above mentioned steps, the Linux system boots up on the master core (core 0), and the Baremetal system boots up on slave core (core 1) automatically.

### 3.4.3 Running examples

The following sections describe how to run the Baremetal examples on the host environment for LS1028ARDB board. Similar steps can be followed for LS1043ARDB, LS1046ARDB, i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 93 9x9 LPDDR4 QSB board.

### 3.4.3.1 Preparing the console

In current Baremetal framework design, two UART ports are used as console. One UART is used for master core and the other UART is used for slave cores. Refer to Section 3.2.2.2 for preparing the console.
3.4.3.2 Running the Baremetal binary

As described earlier, there are two methods to compile the Baremetal framework. One is a standalone method and the other method uses the Real-time Edge software. These methods are described in Section 3.4.2.3 and Section 3.4.2.3.2 respectively.

- If the Real-time Edge software is used to compile the Baremetal image, the Baremetal image is included in the `nxp-image-real-time-edge-xxxx.wic.zst`. In this case, the master core starts the Baremetal image on slave cores automatically.

- If standalone compilation method is used, perform the steps below to run the Baremetal binary from U-Boot prompt of master core. See the below example run on Layerscape platform:

1. After starting U-Boot on the master, download the bare metal image: `u-boot-dtb.bin` on 0x84000000
   using the command below:
   ```
   => tftp 0x84000000 xxxx/u-boot-dtb.bin
   ```
   Where
   - `xxxx` is your tftp server directory.
   - 0x84000000 is the address of `CONFIG_TEXT_BASE` on bare metal for Layerscape platforms.
   **Note:**
   a. The address is 0x50200000 for i.MX 8M Plus LPDDR4 EVK and i.MX 8M Mini LPDDR4 EVK boards.
   b. The address is 0x90200000 for i.MX 93 EVK board.

2. Then, start the Baremetal cores using the command below:
   ```
   => dcache flush; cpu 1 release 0x84000000
   ```
   **Note:** In the command `cpu <num> release 0x84000000`, the `num` can be 1, 2, 3, ... to the maximum CPU number.
   For i.MX 8M Plus LPDDR4 EVK and i.MX 8M Mini LPDDR4 EVK boards, use the below command:
   ```
   => dcache flush;cpu 1 release 50200000;sleep 6;cpu 2 release 50200000;sleep 2;cpu 3 release 50200000;
   ```
   3. Last, the UART2 port displays the logs, and the bare metal application runs on slave cores successfully.

3.4.4 Development based on Baremetal framework

This chapter describes how to develop customer-specific application based on Baremetal framework.

3.4.4.1 Developing the Baremetal application

The “app” directory in the U-boot repository includes the test cases for testing the I2C, GPIO, and IRQ init features. Users can write their custom applications and store them in this directory.

3.4.4.2 Main file app.c

The file `<U-boot path>/app/app.c`, is the main file to add all applications. Users can modify the `app.c` file to add their applications.

- When the standalone method is used to build the Baremetal image as described in Section 3.4.2.3, change the directory to `U-boot path` to read or edit the `app.c` file.
- When the Real-time Edge software is used to compile the Baremetal binary, change to the building directory to view or edit the `app.c` file.
The following is a sample code of the file `app.c` that shows how to add the example test cases of I2C, IRQ, and GPIO.

```c
void core1_main(void)
{
    test_i2c();
    test_irq_init();
    test_gpio();
    return;
}
```

### 3.4.4.3 Common header files

There are some common APIs provided by Baremetal. The table below describes the header files that include the APIs.

<table>
<thead>
<tr>
<th>Header file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asm/io.h</td>
<td>Read/Write IO APIs. For example, <code>_raw_readb</code>, <code>_raw_writeb</code>, <code>out_be32</code>, and <code>in_be32</code>.</td>
</tr>
<tr>
<td>linux/string.h</td>
<td>APIs for manipulating strings. For example, <code>strlen</code>, <code>strcpy</code>, and <code>strcmp</code>.</td>
</tr>
<tr>
<td>linux/delay.h</td>
<td>APIs used for small pauses. For example, <code>udelay</code> and <code>mdelay</code>.</td>
</tr>
<tr>
<td>linux/types.h</td>
<td>APIs specifying common types. For example, <code>_u32</code> and <code>_u64</code>.</td>
</tr>
<tr>
<td>common.h</td>
<td>Common APIs For example, <code>printf</code> and <code>puts</code>.</td>
</tr>
</tbody>
</table>

### 3.4.4.4 GPIO example

The file `uboot/app/test_gpio.c` is an example to test the GPIO feature, and shows how to write a GPIO application.

Here is an example for the i.MX 8M Mini board:

1. First, you need the GPIO header file, `asm-generic/gpio.h` and `dm.h`, which include all interfaces for the GPIO.
2. Then, find the corresponding GPIO description according to the name of the GPIO (such as `GPIO5_7`), configure `GPIO5_7` to OUT direction, configure `GPIO5_8` to IN direction and request it.
3. Now, by writing the value 1 or 0 to `GPIO5_7`, you can receive the same value from `GPIO5_8`.

The Table 11 shows the APIs used in the file `test_gpio.c` application example.

<table>
<thead>
<tr>
<th>Function declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int dm_gpio_lookup_name(const char *name, struct gpio_desc *desc)</code></td>
<td>Look up a named GPIO and return its description</td>
</tr>
<tr>
<td></td>
<td>name - Name to look up, such as <code>GPIO5_7</code></td>
</tr>
<tr>
<td></td>
<td>desc - GPIO description</td>
</tr>
<tr>
<td></td>
<td>Returns: 0 if OK, -ve on error</td>
</tr>
</tbody>
</table>
### 3.4.4.5 I2C example

The file `uboot/app/test_i2c.c` can be used as an example to test the I2C feature and shows how to write an I2C application.

On LS1043ARDB board, read a fixed data from offset 0 of INA220 device (0x40). If the data is 0x39, a message, `[ok] I2C test ok` is displayed on the console.

The table below shows the APIs used in the sample file, `test_i2c.c`.

#### Table 12. I2C APIs and their description

<table>
<thead>
<tr>
<th>Function declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int i2c_set_bus_num (unsigned int bus)</code></td>
<td>Sets the I2C bus. Returns 0 if OK, -1 on error.</td>
</tr>
<tr>
<td><code>int i2c_read (uint8_t chip, unsigned int addr, int alen, uint8_t *buffer, int len)</code></td>
<td>Read data from I2C device chip. Returns 0 if OK, not 0 on error.</td>
</tr>
<tr>
<td><code>int i2c_write (uint8_t chip, unsigned int addr, int alen, uint8_t *buffer, int len)</code></td>
<td>Writes data to I2C device chip. Returns 0 if OK, not 0 on error.</td>
</tr>
</tbody>
</table>
### 3.4.4.6 IRQ example

The file, `uboot/app/test_irq_init.c`, is an example to test the IRQ and IPI (Inter-Processor Interrupts) feature, and shows how to write an IRQ application. The process is described in brief below.

The file `asm/interrupt-gic.h` is the header file of IRQ, and includes all its interfaces. Then, register an IRQ function for SGI 0. After setting an SGI signal, the CPU gets this IRQ and runs the IRQ function. Then, register a hardware interrupt function to show how to use the external hardware interrupt.

SGI IRQ is used for inter-processor interrupts, and it can only be used between bare metal cores. In case you want to communicate between Baremetal core and Linux core, refer to Section 3.4.4.16. SGI IRQ id is 0-15. The SGI IRQ id '8' is reserved for ICC.

**Note:** For i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 93 9x9 LPDDR4 QSB boards, SGI IRQ id is 9.

The Table 13 shows the APIs used in the sample file, `test_irq_init.c`.

<table>
<thead>
<tr>
<th>Table 13. IRQ APIs and their description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Return type</strong></td>
</tr>
</tbody>
</table>
| | int irq_desc_register(struct irq *irq_data, void (*irq_handle)(int, int, void *), void *data) | Registers an IRQ function.  
| | void gic_send_sgi(u32 id, int core_mask) | Sets a SGI IRQ signal.  
| | int irq_set_affinity(struct irq *irq, int core_mask) | Sends the target core for hw IRQ.  
| | int irq_set_polarity(struct udevice *dev, uint id, bool active_low) | Sets the type for hardware IRQ to identify whether the corresponding interrupt is edge-triggered or level-sensitive. |

### 3.4.4.7 QSPI example

The file `uboot/app/test_qspi.c` provides an example that can be used to test the QSPI feature. The below steps show how to write a QSPI application:

1. First, locate the QSPI header files `spi_flash.h` and `spi.h`, which include all interfaces for QSPI.
2. Then, initialize the QSPI flash. Subsequently, erase the corresponding flash area and confirm that the erase operation is successful.
3. Now, read or write to the flash with an offset of 0x3f00000 and size of 0x40000.

The Table 14 shows the APIs used in the file `test_qspi.c` example.

<table>
<thead>
<tr>
<th>Table 14. QSPI APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API name (type)</strong></td>
</tr>
</tbody>
</table>
| | `spi_find_bus_and_cs(bus,cs, &bus_dev, &new)` | The API finds if a SPI device already exists.  
| | | *“bus”: bus index, zero based.  
| | | *“cs”: the value to chip select mode. |
### 3.4.4.8 IFC example

Both LS1043ARDB and LS1046ARDB have IFC controller. However, LS1043ARDB supports both NOR flash and NAND flash, whereas LS1046ARDB supports only NAND flash.

NOR and NAND flash messages are displayed while booting Baremetal cores, as shown below:

<table>
<thead>
<tr>
<th>ID</th>
<th>Flash Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NAND</td>
<td>512 MiB</td>
</tr>
<tr>
<td>1</td>
<td>Flash</td>
<td>128 MiB</td>
</tr>
</tbody>
</table>

or (LS1046ARDB)

<table>
<thead>
<tr>
<th>ID</th>
<th>Flash Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NAND</td>
<td>512 MiB</td>
</tr>
</tbody>
</table>

There is no example code to test it, but we can use a few commands to verify these features.
For LS1043ARDB NOR Flash (the map memory address is 0x60000000), the below command can be used to verify it:

```
=> md 0x60000000
1:60000000: 55aa55aa 0001ee01 10001008 0000000a    .U.U............
1:60000010: 00000000 00000000 02005514 12400080    ........U....@..
1:60000020: 005002e0 002000c1 00000000 00000000    ..P... ..........
1:60000030: 00000000 00880300 00000000 01110000    ............W.x....
1:60000040: 96000000 01000000 78015709 10e00000    ............W.....
1:60000050: 0001809 08000000 18045709 9e000000    ............W...```

For NAND flash on LS1043ARDB and LS1046ARDB, "nand" command can be used to verify it (nand erase, nand read, nand write, and so on.):
```
=> nand info
1:Device 0: nand0, sector size 128 KiB
1:  Page size       2048 b
1:  OOB size          64 b
1:  Erase size    131072 b
1:  subpagesize     2048 b
1:  options     0x00004200
1:  bbt options 0x00028000

=> nand
1:nand - NAND sub-system
1:Usage:
nand info - show available NAND devices
nand device [dev] - show or set current device
nand read - addr off|partition size
nand write - addr off|partition size
  read/write 'size' bytes starting at offset 'off'
  to/from memory address 'addr', skipping bad blocks.
nand read.raw - addr off|partition [count]
  read.raw[.noverify] - addr off|partition [count]
  Usage read.raw/write.raw to avoid ECC and access the flash as-is.
nand erase[.spread] [clean] off size - erase 'size' bytes from offset 'off'
  With '.spread', erase enough for given file size, otherwise,
  'size' includes skipped bad blocks.
nand erase.part [clean] partition - erase entire mtd partition'
nand erase.chip [clean] - erase entire chip'
nand bad - show bad blocks
nand dump[.oob] off - dump page
nand scrub [-y] off size | scrub.part partition | scrub.chip
  really clean NAND erasing bad blocks (UNSAFE)
nand markbad off [...] - mark bad block(s) at offset (UNSAFE)
nand biterr off - make a bit error at offset
```

3.4.4.9 Ethernet example

The file uboot/app/test_net.c provides an example to test the Ethernet feature and shows how to write a net application for using this feature.
Here is an example for the LS1043ARDB (or LS1046ARDB) board.

**Note:** For LS1046ARDB board, network could be assigned by setting `CONFIG_FMAN_FMAN1_COREID=<core num>` (core num could be 0 - 3). If you want to verify it on core 1, please set `CONFIG_FMAN_FMAN1_COREID=1` and compile baremetal image in standalone way to enable the network for the target core.

1. Connect one Ethernet port of LS1043ARDB board to one host machine using Ethernet cable.
   - For LS1046ARDB, the default `ethact` is FM1@DTSEC5. Network cable should be connected to SGMII1 port.
   - For LS1043ARDB, the default `ethact` is FM1@DTSEC3. Network cable should be connected to RGMII1 port.
2. Configure the IP address of the host machine as 192.168.1.2.
3. Power up the LS1043ARDB board. If the network is connected, the message `host 192.168.1.2 is alive` is displayed on the console.
4. The IP addresses of the board and host machine are defined in the file `test_net.c`. In this file, modify the IP address of LS1043ARDB board using variable `ipaddr` and change the IP address of host machine using variable `ping_ip`.

The table below lists the Net APIs and their description.

<table>
<thead>
<tr>
<th>API name (type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void net_init (void)</td>
<td>Initializes the network</td>
</tr>
</tbody>
</table>
| int net_loop (enum proto_t protocol) | Main network processing loop.  
  • enum proto_t protocol - protocol type |
| int eth_receive (void *packet, int length) | Reads data from NIC device chip.  
  • void *packet  
  • length - Network packet length  
  Returns length |
| int eth_send (void *packet, int length) | Writes data to NIC device chip.  
  • packet - pointer to the packet is sent  
  • length - Network packet length  
  Returns length |

### 3.4.4.10 USB example

The file `uboot/app/test_usb.c` provides an example that can be used to test the USB features. The steps below show how to write a USB application:

1. Connect a USB disk to the USB port.
2. Include the header file, `usb.h`, which includes all APIs for USB.
3. Initialize the USB device using the `usb_init` API.
4. Scan the USB storage device on the USB bus using the `usb_stor_scan` API.
5. Get the device number using the `blk_get_devnum_by_type` API.
6. Read data from the USB disk using the `blk_dread` API.
7. Write data to the USB disk using the `blk_dwrite` API.

The table below shows the APIs used in the file `test_usb.c` example:

<table>
<thead>
<tr>
<th>API name (type)</th>
<th>Description</th>
</tr>
</thead>
</table>
Table 16. USB APIs and their description...continued

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int usb_init(void)</td>
<td>Initializes the USB controller.</td>
</tr>
<tr>
<td>int usb_stop(void)</td>
<td>Stops the USB controller.</td>
</tr>
<tr>
<td>int usb_stor_scan(int mode)</td>
<td>Scans the USB and reports device information to the user if mode = 1</td>
</tr>
<tr>
<td></td>
<td>• Mode – if mode = 1, the information is returned to user</td>
</tr>
<tr>
<td></td>
<td>• the current device, or</td>
</tr>
<tr>
<td></td>
<td>• -1 (if device not found).</td>
</tr>
<tr>
<td>struct blk_desc *blk_get_devnum_by_type(enum if_type if_type, int devnum)</td>
<td>Get a block device by type and number.</td>
</tr>
<tr>
<td></td>
<td>• if_type – Block device type</td>
</tr>
<tr>
<td></td>
<td>• devnum – device number</td>
</tr>
<tr>
<td></td>
<td>Returns</td>
</tr>
<tr>
<td></td>
<td>• Points to block device descriptor, or</td>
</tr>
<tr>
<td></td>
<td>• NULL (if not found).</td>
</tr>
<tr>
<td>unsigned long blk_dread(struct blk_desc *block_dev, lbaint_t start, lbaint_t blkcnt, void *buffer);</td>
<td>Reads data from USB device.</td>
</tr>
<tr>
<td></td>
<td>• block_dev – block device descriptor</td>
</tr>
<tr>
<td></td>
<td>• start – start block</td>
</tr>
<tr>
<td></td>
<td>• blkcnt – block number</td>
</tr>
<tr>
<td></td>
<td>• buffer – buffer to store the data</td>
</tr>
<tr>
<td></td>
<td>Returns</td>
</tr>
<tr>
<td></td>
<td>• Points to block device descriptor, or</td>
</tr>
<tr>
<td></td>
<td>• NULL (if not found).</td>
</tr>
<tr>
<td>unsigned long blk_dwrite(struct blk_desc *block_dev, lbaint_t start, lbaint_t blkcnt, const void *buffer);</td>
<td>Writes data to USB device.</td>
</tr>
<tr>
<td></td>
<td>• block_dev – block device descriptor</td>
</tr>
<tr>
<td></td>
<td>• start – start block</td>
</tr>
<tr>
<td></td>
<td>• blkcnt – block number</td>
</tr>
<tr>
<td></td>
<td>• buffer – buffer to store the data</td>
</tr>
<tr>
<td></td>
<td>Returns</td>
</tr>
<tr>
<td></td>
<td>• Points to block device descriptor, or</td>
</tr>
<tr>
<td></td>
<td>• NULL (if not found).</td>
</tr>
</tbody>
</table>

3.4.4.11 PCIe example

The file `app/test_pcie.c` provides a sample code to test PCIe and network card (such as e1000) features. The steps below show how to write a PCIe and net application:

1. Insert a PCIe network card (such as e1000) into PCIe2, or PCIe3 slot (if it exists).
2. Configure the IP address of the host machine to 192.168.1.2.
3. Include the files `include/pci.h` and `include/netdev.h`.
4. Initialize the PCIe controller using the `pci_init` API.
5. Get uclass device by its name using the `uclass_get_device_by_seq` API.
6. Initialize the PCIe network device using the `pci_eth_init` API.
7. Begin pinging the host machine using the `net_loop` API.

The table below shows the APIs used in the file `test_pcie.c` example.

<table>
<thead>
<tr>
<th>API name (type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void pci_init(void)</td>
<td>Initializes the PCIe controller. Does not return a value.</td>
</tr>
<tr>
<td>int uclass_get_device_by_seq(enum uclass_id id, int seq, struct udevice **devp)</td>
<td>Gets the uclass device based on an ID and sequence:</td>
</tr>
<tr>
<td></td>
<td>• id – uclass ID</td>
</tr>
<tr>
<td></td>
<td>• seq – sequence</td>
</tr>
<tr>
<td></td>
<td>• devp – Pointer to device</td>
</tr>
</tbody>
</table>
### 3.4.4.12 ENETC example

The file `app/test_net.c` provides an example to test ENETC Ethernet feature and shows how to write a net application for using this feature. This example is a special case of using Net APIs.

The file `test_net` for ENETC is only an example for the LS1028ARDB board with `CONFIG_ENETC_COREID_SET` enabled.

1. Connect ENETC port of LS1028ARDB board to one host machine using Ethernet cable.
2. Configure the IP address of the host machine as `192.168.1.2`.
3. Power up the LS1028ARDB board. If the network is connected, the message `host 192.168.1.2 is alive` is displayed on the console.
4. The IP addresses of the board and host machines are defined in the file `test_net.c`. In this file, modify the IP address of LS1028ARDB board using variable `ipaddr` and change the IP address of host machine using variable `ping_ip`.

The table below lists the Net APIs for ENETC and their description, refer to Section 3.4.4 for other Net APIs.

<table>
<thead>
<tr>
<th>API name (type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void pci_init(void)</code></td>
<td>Initializes the PCIe controller. Does not return a value.</td>
</tr>
<tr>
<td><code>void eth_initialize(void)</code></td>
<td>Initializes the Ethernet.</td>
</tr>
</tbody>
</table>

### 3.4.4.13 SAI example

The audio feature needs SAI module and codec drivers. The following sections provide an introduction to SAI module and the audio codec (SGTL5000). These sections also describe the steps for integrating audio with Baremetal and running an audio application on Baremetal.

#### 3.4.4.13.1 Synchronous Audio Interface (SAI)

The LS1028A integrates six SAI modules, but only SAI4 is used by LS1028ARDB board. The synchronous audio interface (SAI) supports full duplex serial interfaces with frame synchronization. The bit clock and frame sync of SAI are both generated externally (SGTL5000).

- Transmitter with independent bit clock and frame sync supporting 1 data line
- Receiver with independent bit clock and frame sync supporting 1 data line
- Maximum Frame Size of 32 words
• Word size of between 8-bits and 32-bits
• Word size configured separately for first word and remaining words in frame
• Asynchronous 32 × 32-bit FIFO for each transmit and receive channel
• Supports graceful restart after FIFO error

![Figure 12. SAI block diagram](image)

3.4.4.13.2 Audio codec (SGTL5000)

The SGTL5000 is a low-power stereo codec with headphone amplifier from NXP. It is designed to provide a complete audio solution for products requiring LINEIN, MIC_IN, LINEOUT, headphone-out, and digital I/Os. It allows an 8.0 MHz to 27 MHz system clock as input. The codec supports 8.0 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 44.1 kHz, 48 kHz, and 96 kHz sampling frequencies. The LS1028ARDB board provides a 25 MHz crystal oscillator to the SGTL5000.

The SGTL5000 provides two interfaces (I2C and SPI) to setup registers. The LS1028ARDB board uses I2C interface.
3.4.4.13.3 Digital interface formats

The SGTL5000 provides five common digital interface formats. The SAI and SGTL5000 digital interface formats must be the same.

- **I2S Format (n = bit length)**

```
CHIP_I2SO_CTRL field values:
(SCLKFREQ = 0; SCLK_INV = 0; DLEN = 1; I2S_MODE = 0; LRALIGN = 0; LRPOL = 0)
```

![I2S Format (n = bit length)](image)

- **Left Justified Format (n = bit length)**

Figure 13. System block diagram, signal flow, and gain

Figure 14. I2S Format (n = bit length)
CHIP_I2S0_CTRL field values:
(SCLKFREQ = 0; SCLK_INV = 0; DLEN = 1; I2S_MODE = 0; LRALIGN = 1; LRPOL = 0)

Figure 15. Left Justified Format (n = bit length)

• Right Justified Format (n = bit length)

CHIP_I2S0_CTRL field values:
(SCLKFREQ = 0; SCLKINV = 0; DLEN = 1; I2S_MODE = 1; LRALIGN = 1; LRPOL = 0)

Figure 16. Right Justified Format (n = bit length)

• PCM Format A

CHIP_I2S0_CTRL = 0x01F4
(SCLKFREQ = 1; MS = 1; SCLK_INV = 1; DLEN = 3; I2S_MODE = 2; LRALIGN = 0)

Figure 17. PCM Format A

• PCM Format B
3.4.4.13.4 Running the SAI application

In order to run SAI application, Baremetal images should be rebuilt with SAI support.

1. Enable SAI support in Real-time Edge software

```
$ cd yocto-real-time-edge/sources/meta-real-time-edge
# Open file "conf/distro/include/real-time-edge-base.inc", add "sai" to
"DISTRO_FEATURES:append:ls1028ardb" like this:
DISTRO_FEATURES:append:ls1028ardb = " jailhouse real-time-edge-libbee real-time-
edge-libblep libnfc-nci \ 
wayland-protocols weston imx-gpu-viv libdrm kmscube \ 
real-time-edge-sysrepo tsn-scripts wayland alsa sai"
```

2. Build the image

```
$ cd yocto-real-time-edge
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1028ardb source real-time-edge-
setup-env.sh -b build-ls1028ardb-bm
$ bitbake nxp-image-real-time-edge
```

3. Play a demo audio file in slave core after booting the board:

```
=> wavplayer
******************************************************************************
audioformat: PCM nchannels: 1 samplerate: 16000 bitrate: 256000 blockalign: 2
bps: 16 datasize: 67968 datastart: 44
******************************************************************************
sctl5000 revision 0x11 fsl_sai_ofdata_to_platdata Probed
sound 'sound' with codec 'codec@a' and i2s 'sai@f130000'
i2s_transfer_tx_data The music waits for the end! The music is finished!
******************************************************************************
```

3.4.4.14 FlexTimer example

The FlexTimer module (FTM) works on Baremetal core as the wakeup source for LS1046ARDB. It can support nanosecond (ns) level alarm setting.

There is no example code to test it, but we can use a few commands to verify these features.

Use the below commands to verify FTM feature:
• Use "ftm" command to get help information:

```bash
=> ftm
1:ftm - ftm alarm test
```

**Usage:**
- `ftm test ftm alarm`
- `show` - show FTM test result
- `start [count] us` - start FTM test
- `stop` - stop FTM test

• Use "ftm start [count]" command to start ftm test:

```bash
=> ftm start
1:Use default alarm time - 5 us
1:FTM test start.
```

```bash
=> ftm start 100
1:FTM test start.
```

• Using "ftm stop" command to stop ftm test and show the test result:

```bash
=> ftm stop
1:FTM test stop.
```

<table>
<thead>
<tr>
<th>irq count</th>
<th>total (us)</th>
<th>average (us)</th>
<th>max (us)</th>
<th>min (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3087560</td>
<td>309579251</td>
<td>100</td>
<td>102</td>
<td>100</td>
</tr>
</tbody>
</table>

• Use "ftm show" command to show the test result:

```bash
=> ftm show
1:irq count | total (us) | average (us) | max (us) | min (us) |
1:317803    | 31854521   | 100          | 102      | 100      |
```

The table below lists the attributes for "ftm show" and "ftm stop" result:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq count</td>
<td>Generated interrupt single count since &quot;ftm start&quot; command</td>
</tr>
<tr>
<td>total (us)</td>
<td>The time since &quot;ftm start&quot; command</td>
</tr>
<tr>
<td>average (us)</td>
<td>The average time between two interrupt signals</td>
</tr>
<tr>
<td>max (us)</td>
<td>The maximum time between two interrupt signals</td>
</tr>
<tr>
<td>min (us)</td>
<td>The minimum time between two interrupts signals</td>
</tr>
</tbody>
</table>

The table below lists the FTM APIs and their description.

<table>
<thead>
<tr>
<th>API Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| int ftm_rtc_set_alarm_by_us (struct udevice *dev, unsigned long us, void (*func)(void *)) | Setting alarm by us count
  - struct udevice *dev – device struct of ftm
  - unsigned long us – the time for ftm alarm
  - void (*func)(void *) – the handle function when timeup |

| void ftm_rtc_set_alarm (struct udevice *dev, u16 ticks, void (*func)(void *)) | Setting alarm by ftm timer count
  - struct udevice *dev – device struct of ftm
  - u16 ticks – the timer counter for ftm alarm
  - void (*func)(void *) – the handle function when timeup |

| void ftm_rtc_alarm_stop(struct udevice *dev) | Stop and reset ftm alarm
  - struct udevice *dev – device struct of ftm |
### 3.4.4.15 Newlib’s math library

In order to control IO devices such as changing the speed or angle, mathematical calculations are required. Newlib’s math library is added to support such calculations. Newlib is a C library intended for use on embedded systems.

All math related files are under `math` folder. The file directory structure is as follows:

```
math
├── COPYING
└── include
    ├── math.h
    └── lib
        ├── libm.a
    README
```

To use math library, the below code should be in the header of the file, and then we can directly call all kinds of math APIs.

```c
#include <math.h>
#undef __always_inline
#undef __section
#include <stdlib.h>
#include <common.h>
#include <command.h>
#undef log
```

For the detailed usage, refer to the example file which is `math.c` under `cmd` folder. The example shows how to call the API of math library including `acos/asin/atan/cos/sin/tan` and `log/pow/sqrt`. We can use the `math` command to verify these APIs under U-Boot command.

For example:

```
=> math
```

**math - Test Math Functions**

**Usage:**

math - Only test some simple math functions:

```c
math acos x(double)
math asin x(double)
math atan x(double)
math atan2 y x(double)
math cos x(double)
math cosh x(double)
math sin x(double)
math sinh x(double)
math tan x(double)
math tanh x(double)
math exp x(double)
math ldexp x(double) exp(int)
math log x(double)
```
math log10 x(double)
math pow x(double) y(double)
math sqrt x(double)
math ceil x(double)
math fabs x(double)
math floor x(double)
math fmod x(double) y(double)

=> math asin 0.8
  = 0.927
1:=> math sin 1.0
  = 0.841
1:=> math cos 1.0
  = 0.540
=> math log 10
  = 2.302
1:=> math log10 10
  = 1.000

3.4.4.16 ICC module

Inter-core communication (ICC) module works on Linux core (master) and Baremetal core (slave). It provides the data transfer between cores via SGI inter-core interrupt and shared memory blocks. It can support multicore silicon platform and transfer the data concurrently and efficiently.

ICC module structure is based on two basics:

- SGI: Software-generated Interrupts in Arm GIC, used to generate inter-core interrupts. The ICC module uses the number 8 SGI interrupt for all Linux and Baremetal cores.
- Shared memory: A memory space shared by all platform cores. The base address and size of the share memory should be defined in header files before compilation.

ICC modules can work concurrently, lock-free among multicore platform, and support broadcast case with Buffer Descriptor Ring mechanism.

The figure below shows the basic operating principle for data transfer from Core 0 to Core 1. After the data writing and head point moving to next, Core 0 triggers a SGI (8) to Core 1. After this step, the Core 1 gets the BD ring updated status and reads the new data, then moves the tail point to next.
For a multicore platform (that is, four cores), the total BD rings are arranged as shown in the following figure. (See the BD rings on Core 0 and Core 1.)

All the ICC ring structures, BD structures, and blocks for data are in the shared memory. A four-core platform ICC module would map the shared memory as shown in the figure below.
Generally, Core 0 runs Linux as master core while other cores run Baremetal as slaves. They obtain the same size of shared memory to structure the rings and BDs, and split the blocks space with 4k unit for each block. The reserved space at the top of the shared memory is out of the ICC module and for the custom usage.

For LS1028ARDB platform with two cores, the shared memory map is defined as:

- The total shared memory size is 256 MB.
- The reserved space for custom usage is 16 MB at the top of the shared memory space.
- Core 0 runs Linux as master core, the share memory size for ICC is 120 MB, in which the ring and BD structure space is 2 M, and the block space for data is 118 MB with 4K for each block.
- Core 1 runs Baremetal as slave core, the share memory size for ICC is 120 MB, in which the ring and BD structure space is 2M, and the block space for data is 118 MB with 4K for each block.

The ICC module includes two parts of the code:

- ICC code for Linux user space, works for data transfer between master core and slave cores. The code is integrated into the Real-time Edge software and named `real-time-edge-icc`. After the compilation, the icc binary is put into the Linux file system.
- ICC code for Baremetal runs on every slave core, works for data transfer between Baremetal cores and master core.


```
|-- icc-main.c   -- the example case commands
|-- inter-core-comm.c
|-- inter-core-comm.h -- include the header file to use ICC module
```
__Makefile__

The ICC code for Baremetal in `baremetal` directory:

```
baremetal/
├── arch/arm/lib/inter-core-comm.c
├── arch/arm/include/asm/inter-core-comm.h  -- includes the header file to use ICC module
└── cmd/icc.c  -- the example case commands
```

The ICC modules of the APIs are exported out for usage in both Linux user space and Baremetal code.

### Table 21. ICC APIs

<table>
<thead>
<tr>
<th>APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unsigned long icc_ring_state(int coreid)</code></td>
<td>Checks the ring and block state.</td>
</tr>
<tr>
<td></td>
<td>Returns:</td>
</tr>
<tr>
<td></td>
<td>• 0 - if empty.</td>
</tr>
<tr>
<td></td>
<td>• !0 - the working block address currently.</td>
</tr>
<tr>
<td><code>Unsigned long icc_block_request(void)</code></td>
<td>Requests a block, which is ICC_BLOCK_UNIT_SIZE size.</td>
</tr>
<tr>
<td></td>
<td>Returns:</td>
</tr>
<tr>
<td></td>
<td>• 0 - failed.</td>
</tr>
<tr>
<td></td>
<td>• !0 - block address can be used.</td>
</tr>
<tr>
<td><code>void icc_block_free(unsigned long block)</code></td>
<td>Frees a block requested.</td>
</tr>
<tr>
<td></td>
<td>Be careful if the destination cores are working on this block.</td>
</tr>
<tr>
<td><code>int icc_irq_register(int src_coreid, void (*irq_handle)(int, unsigned long, unsigned int))</code></td>
<td>Registers ICC callback handler for received data.</td>
</tr>
<tr>
<td></td>
<td>Returns:</td>
</tr>
<tr>
<td></td>
<td>• 0 - on success</td>
</tr>
<tr>
<td></td>
<td>• -1 - if failed.</td>
</tr>
<tr>
<td><code>int icc_set_block(int core_mask, unsigned int byte_count, unsigned long block)</code></td>
<td>Sends the data in the block to a core or multicore.</td>
</tr>
<tr>
<td></td>
<td>This triggers the SGI interrupt.</td>
</tr>
<tr>
<td></td>
<td>Returns:</td>
</tr>
<tr>
<td></td>
<td>• 0 - on success</td>
</tr>
<tr>
<td></td>
<td>• -1 - if failed.</td>
</tr>
<tr>
<td><code>void icc_show(void)</code></td>
<td>Shows the ICC basic information.</td>
</tr>
<tr>
<td><code>int icc_init(void)</code></td>
<td>Initializes the ICC module.</td>
</tr>
</tbody>
</table>

#### 3.4.4.16.1 ICC examples

This section provides example commands for use cases in both Linux user space and Baremetal code. They can be used to check and verify the ICC module conveniently.

1. In Linux user space, use the command `icc` to display the supported cases.

```
[root@LS1046ARDB ~] # icc
icc show - Shows all icc rings status at this core
icc perf <core_mask> <counts> - ICC performance to cores <core_mask> with <counts> bytes
icc send <core_mask> <data> <counts> - Sends <counts> <data> to cores <core_mask>
icc irq <core_mask> <irq> - Sends SGI <irq> ID[0 - 15] to <core_mask>
icc read <addr> <counts> - Reads <counts> 32bit register from <addr>
icc write <addr> <data> - Writes <data> to a register <addr>
```
Likewise, in Baremetal system, use the command `icc` to view the supported cases.

```
=> icc
1:icc - Inter-core communication via SGI interrupt
1:Usage:
icc show                                - Show all icc rings status at this
core
icc perf <core_mask> <counts>
   <core_mask> with <counts> bytes      - ICC performance to cores
icc send <core_mask> <data> <counts>
   <core_mask>                        - Send <counts> <data> to cores
icc irq <core_mask> <irq>
   <core_mask>                        - Send SGI <irq> ID[0 - 15] to
```

2. The ICC module command examples on LS1046ARDB with Linux (Core 0) + Baremetal (Core 1, 2, 3) system:

Run `icc send 0x2 0x55 128` to send 128 bytes data 0x55 to core 1.

```
[root@LS1046ARDB ~] # icc send 0x2 0x55 128
gic_base: 0xffffa033f000, share_base: 0xffff9133f000, share_phy: 0xd0000000,
block_phy: 0x0d0200000
ICC send testing ...
Target cores: 0x2, bytes: 128
ICC send: 128 bytes to 0x2 cores success
all cores: reserved_share_memory_base: 0xd0000000; size: 16777216
mycoreid: 0; ICC_SGI: 8; share_memory_size: 62914560
block_unit_size: 4096; block number: 14848; block_idx: 0
#ring 0 base: 0xffff9133f000; dest_core: 0; SGI: 8
desc_num: 128; desc_base: 0xd000000c0; head: 0; tail: 0
busy_counts: 0; interrupt_counts: 0
#ring 1 base: 0xffff9133f030; dest_core: 1; SGI: 8
desc_num: 128; desc_base: 0xd000008c0; head: 1; tail: 1
busy_counts: 0; interrupt_counts: 1
#ring 2 base: 0xffff9133f060; dest_core: 2; SGI: 8
desc_num: 128; desc_base: 0xd000010c0; head: 0; tail: 0
busy_counts: 0; interrupt_counts: 0
#ring 3 base: 0xffff9133f090; dest_core: 3; SGI: 8
desc_num: 128; desc_base: 0xd000018c0; head: 0; tail: 0
busy_counts: 0; interrupt_counts: 0
```

At the same time, Core 1 displays the received information.

```
=> 1:Get the ICC from core 0; block: 0xd0200000, bytes: 128, value: 0x55
```

3. ICC command run on Baremetal side

```
=> icc send 0x1 0xaa 128
1:ICC send testing ...
1:Target cores: 0x1, bytes: 128
1:ICC send: 128 bytes to 0x1 cores success
1:all cores: reserved_share_memory_base: 0xd0000000; size: 16777216
1:mycoreid: 1; ICC_SGI: 8; share_memory_size: 62914560
1:block_unit_size: 4096; block number: 14848; block_idx: 0
1:#ring 0 base: 00000000d3c00000; dest_core: 0; SGI: 8
1:desc_num: 128; desc_base: 00000000d3c000c0; head: 1; tail: 1
1:busy_counts: 0; interrupt_counts: 1
1:#ring 1 base: 00000000d3c00030; dest_core: 1; SGI: 8
1:desc_num: 128; desc_base: 00000000d3c0010c0; head: 0; tail: 0
1:busy_counts: 0; interrupt_counts: 0
1:#ring 2 base: 00000000d3c00060; dest_core: 2; SGI: 8
1:desc_num: 128; desc_base: 00000000d3c0008c0; head: 0; tail: 0
1:busy_counts: 0; interrupt_counts: 0
```
Then, Core 0 side (Linux) receives this data:

```
[root@LS1046ARDB ~] # [ 4247.733753] 000: Get the ICC from core 1; block: 0xd3e00000, bytes: 128, value: 0xaa
```

### 3.4.4.17 Single hardware interrupt routed to multiple cores

This section describes how to use GPIO to simulate external interrupt to notify all slave cores. With this feature, all the slave cores can be triggered to perform operations almost at the same time via a single hardware interrupt.

Two GPIO pins are selected. One pin is used to output 0 and 1 to simulate an external hardware. The other one is used as an interrupt pin to trigger an interrupt to a core under pull-down mode. When the core receives the interrupt, it triggers other slave cores via ICC SGI interrupt.

This feature is supported on LS1046ARDB and i.MX 8M Mini. The GPIO interrupt number and two GPIO pins required for the interrupt test can be obtained from the corresponding `dts` file.

In `fsl-ls1046a-rdb.dts`

```dts
gpio_int {
    compatible = "fsl,gpio-int";
    gpios = &gpio1 1 0>,
            &gpio1 2 0>;
    interrupts = <0 99 0>;
};
```

In `imx8mm-evk.dts`

```dts
gpio_int {
    compatible = "fsl,gpio-int";
    gpios = &gpio5 7 0>,
            &gpio5 8 0>;
    interrupts = <0 104 0>;
};
```

On LS1046ARDB, GPIO2_01 and GPIO2_02 are selected. GPIO2_01 is used to simulate an external hardware whereas GPIO2_02 triggers an interrupt. Connect the GPIO2_01 and GPIO2_02 pins on the board. TP14 and TP13 are connected to GPIO2_01 and GPIO2_02 separately. The figure below shows how to connect TP14 and TP13.
On i.MX 8M Mini, connect the pins GPIO5_07 and GPIO5_08 on the board. GPIO5_07 is used to simulate an external hardware whereas GPIO5_08 triggers an interrupt. The figure below shows how to connect GPIO5_07 and GPIO5_08.
On LS1046ARDB, GPIO2_01 and GPIO2_02 are multiplexed with SPI_CS_B[0] and SDHC_DAT[4] signals. User must configure RCW[382 ~ 383] to 0b'10 to enable GPIO2[0] signal.

Since GPIO2 is assigned to Baremetal core, Linux should not use it again. We can disable GPIO2 under Linux via dts file. The below code should be added in Linux kernel file fsl-ls1046a-rdb-sdk-bm.dts to disable GPIO2.

```c
&gpio1 {
  status = "disabled";
};
```

### Table 22. GPIO_INT driver APIs and their description

<table>
<thead>
<tr>
<th>Function declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int gpio_request_by_name(struct udevice *dev, const char *list_name, int index, struct gpio_desc *desc, int flags)</td>
<td>Locate and request a GPIO by name dev- Device requesting the GPIO index - Index number of the GPIO in that list use request (0 = first) desc - Returns GPIO description information flags - Indicates the GPIO input/output settings Returns: 0 if OK, ~ve on error</td>
</tr>
</tbody>
</table>
Table 22. GPIO_INT driver APIs and their description…continued

<table>
<thead>
<tr>
<th>Function declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int dm_gpio_set_value(const struct gpio_desc *desc, int value)</td>
<td>Configures the direction of GPIO to OUT and writes the value to it. Description: desc - GPIO description value - the value written to this GPIO Returns: 0 if OK, -ve on error</td>
</tr>
<tr>
<td>int dm_gpio_set_interrupt(const struct gpio_desc *desc)</td>
<td>Enable GPIO interrupt Description: desc - GPIO description Returns: 0 if OK, -ve on error</td>
</tr>
<tr>
<td>int dm_gpio_clr_interrupt(const struct gpio_desc *desc)</td>
<td>Disable GPIO interrupt Description: desc - GPIO description Returns: 0 if OK, -ve on error</td>
</tr>
</tbody>
</table>

Under Baremetal, gpio_interrupt command provides "enable", "start" and "stop" commands to control these two pins.

- gpio_interrupt enable - Initializes the gpio_int driver and enables the interrupt.
- gpio_interrupt start - Sets GPIO2_01 to high.
- gpio_interrupt stop - Sets GPIO2_01 to low.

gpio_interrupt can run under BareMetal console.

The gpio_interrupt enable command should be run first to initialize two pins. Then, use the command pair gpio_interrupt start and gpio_interrupt stop to pull high and pull down GPIO2_01/GPIO5_07. After the command pair, GPIO2_02/GPIO5_08 triggers an interrupt when getting pull-down signal. The core1 sends the SGI interrupt to other slave cores. The time of GPIO interrupt and SGI interrupt is dumped by each slave core. The latency is the time difference between GPIO interrupt and SGI interrupt. The below example shows the latency is about 1 µs. It means all slave cores could be triggered within 1 µs.

```
=> gpio_interrupt enable
=> gpio_interrupt start
=> gpio_interrupt stop
1:Time(us): 0x33cc582
3:Time(us): 0x33cc585, Get the SGI from CoreID: 1
2:Time(us): 0x33cc584, Get the SGI from CoreID: 1
=>
```

3.4.4.18 Hardware resource allocation

This section describes how to modify the hardware resource allocation depending on the application and used reference design board.

3.4.4.18.1 LS1028ARDB board

This section describes the ENETC configuration setting for LS1028A reference design boards.

3.4.4.18.1.1 ENETC

LS1028ARDB has only one ENETC controller in use, which is assigned to core1 as the default setting. The controller can be reconfigured by using the command, make menuconfig.

See the following:

```
ARM architecture --->
```
3.4.4.18.1.2 I2C

This section describes how to configure the I2C bus on LS1028A reference design boards.

LS1028ARDB has eight I2C controllers, but only controller 0 is used for I2C devices. For example, RTC, Thermal Monitor, and Linux (core 0) use this controller for some features (for example, RTC). Therefore, the code below just shows how to enable I2C on Baremetal side.

**Note:**

*Operate the I2C devices in Baremetal side CAREFULLY.*

```c
#define CONFIG_SYS_I2C_MXC_I2C1 /* enable I2C bus 0 */
#define CONFIG_SYS_I2C_MXC_I2C2 /* enable I2C bus 1 */
#define CONFIG_SYS_I2C_MXC_I2C3 /* enable I2C bus 2 */
#define CONFIG_SYS_I2C_MXC_I2C4 /* enable I2C bus 3 */
#define CONFIG_I2C_BUS_CORE_ID_SET
#define CONFIG_SYS_I2C_MXC_I2C0_COREID 1
```

The `CONFIG_SYS_I2C_MXC_I2C0_COREID` defines the slave core that runs the I2C bus.

Since I2C is enabled in DM mode on Baremetal side, there is no automatic code to test it. Follow the below steps to read RTC (0x51 address, is on bus 2) on Baremetal side:

```bash
=> i2c bus
Bus 0: i2c@2000000 (active 0)
  77: i2c-mux@77, offset len 1, flags 0
  57: generic 57, offset len 1, flags 0
Bus 1: i2c@2000000->i2c-mux@77->i2c@01
Bus 2: i2c@2000000->i2c-mux@77->i2c@03
  51: rtc@51, offset len 1, flags 0
Bus 3: i2c@2010000
Bus 4: i2c@2020000
Bus 5: i2c@2030000
Bus 6: i2c@2040000
Bus 7: i2c@2050000
Bus 8: i2c@2060000
Bus 9: i2c@2070000
=> i2c md 0x51 0
Error reading the chip: -121
=> i2c dev 2
Setting bus to 2
=> i2c md 0x51 0
0000: 04 00 36 03 12 15 02 12 20 80 80 80 80 80 00 c2 ..6......  .......
```

3.4.4.18.1.3 SAI

LS1028ARDB has only one SAI module in use, which is assigned to core1 in the default setting. The SAI module can be reconfigured by using the command, `make menuconfig`.

See the following:

```
Command line interface -->
```
3.4.4.18.2 LS1043ARDB or LS1046ARDB board

The following sections describe the hardware resource allocation for the LS1043ARDB or LS1046ARDB boards for implementing the supported features.

3.4.4.18.2.1 Linux DTS

Remove cpu1, cpu2, cpu3 nodes on DTS, and remove all the devices that bare metal has used.

3.4.4.18.2.2 Memory configuration

This section describes the memory configuration for LS1043ARDB or LS1046ARDB boards.

The LS1043ARDB or LS1046ARDB boards have a DDR of size 2 GB. To use the bare metal framework, configure DDR into three partitions:

- 512M for core0 (Linux)
- 256M for core1 (bare metal)
- 256M for core2 (bare metal)
- 256M for core3 (bare metal), and 256M for shared memory.

The configuration can be defined in the file `include/configs/ls1043a_baremetal.h`.

```c
#define CFG_BAREMETAL_SYS_SDRAM_MASTER_SIZE (512 * 1024 * 1024UL)
#define CFG_BAREMETAL_SYS_SDRAM_SLAVE_SIZE (256 * 1024 * 1024UL)
#define CFG_BAREMETAL_SYS_SDRAM_RESERVE_SIZE (16 * 1024 * 1024UL)
#define CFG_BAREMETAL_SYS_SDRAM_SHARE_SIZE \ 
    ((256 * 1024 * 1024UL) - CFG_BAREMETAL_SYS_SDRAM_RESERVE_SIZE)
```

**Note:** The memory configuration must be consistent with the U-Boot configuration of core0.

The memory configuration for bare metal is shown in the figure below.
The functions included in `malloc.h` in the table below can be used to allocate or free memory in program. Modify `CONFIG_SYS_MALLOC_LEN` in defconfig of the board to change the maximum size of malloc.

### Table 23. Memory APIs description

<table>
<thead>
<tr>
<th>API name (type)</th>
<th>Description</th>
</tr>
</thead>
</table>
| `void* malloc(size_t n)` | Allocates memory  
• "n" – length of allocated chunk  
• Returns a pointer to the newly allocated chunk |
| `void free(void *ptr)` | Releases the chunk of memory pointed to by ptr (where "ptr" is a pointer to the chunk of memory) |

The GPIO for LS1043ARDB (or LS1046ARDB) has four GPIO controllers. You need to add a GPIO node in the file `ls1043/6a-rdb.dts` to assign a GPIO resource to different cores. The configuration can be done in the file `arch/arm/dts/fsl-ls1043/6a-rdb.dts`.

#### 3.4.4.18.2.3 GPIO

LS1043 and LS1046A have four GPIO controllers. You can add a GPIO node in the file `ls1043-rdb.dts` or `ls1046a-rdb.dts` to assign a GPIO resource to different cores. The configuration is in `arch/arm/dts/fsl-ls1043a-rdb.dts` or `arch/arm/dts/fsl-ls1046a-rdb.dts`. Use the command below to add a GPIO node:

```c
&gpio2 {  
    status = "okay";
};
```

#### 3.4.4.18.2.4 I2C

This section describes how to configure the I2C bus on LS1028A, LS1043A, or LS1046A reference design boards.

The LS1043ARDB (or LS1028ARDB / LS1046ARDB) has four I2C controllers. You can configure the I2C bus using the file `ls1043ardb_bm_defconfig` using the commands below:

```make
CONFIG_SYS_I2C_MXC_I2C1=y
```
CONFIG_SYS_I2C_MXC_I2C2=y
CONFIG_SYS_I2C_MXC_I2C3=y
CONFIG_SYS_I2C_MXC_I2C4=y
CONFIG_I2C_COREID SET=y
CONFIG_SYS_I2C_MXC_I2C0_COREID=1
CONFIG_SYS_I2C_MXC_I2C1_COREID=2
CONFIG_SYS_I2C_MXC_I2C2_COREID=3
CONFIG_SYS_I2C_MXC_I2C3_COREID=1

The `CONFIG_SYS_I2C_MXC_I2C0_COREID` defines the slave core that runs the I2C bus.

### 3.4.4.18.2.5 Hardware interrupts

LS1043A has twelve IRQs as external IO signals connected to interrupt the controller. These twelve IRQs can be used on baremetal cores. The ids for these signals, IRQ0-IRQ11 are: 163, 164, 165, 167, 168, 169, 177, 178, 179, 181, 182, and 183. GIC interrupt APIs are defined in `asm/interrupt-gic.h`. The following example shows how to register a hardware interrupt:

```c
//register HW interrupt
int irq_desc_register(struct irq *irq_data, void (*irq_handle)(int, int, void *), void *data);
int irq_set_polarity(struct udevice *dev, uint irq, bool active_low);
int irq_set_affinity(struct irq *irq, int core_mask);
```

### 3.4.4.18.2.6 QSPI

LS1046ARDB has a QSPI flash device. To configure the QSPI on `ls1046ardb_config.h`, use the command below:

```c
#define CONFIG_FSL_QSPI_COREID 1
```

Here, the `CONFIG_FSL_QSPI_COREID` defines the slave core that runs this QSPI.

### 3.4.4.18.2.7 IFC

LS1043A and LS1046A have IFC controller. LS1043RDB supports both NOR flash and NAND flash, whereas LS1046RDB supports only NAND flash.

1. IFC is disabled in Linux kernel via disabling "ifc" node:

```c
&ifc {
    status = "disabled";
};
```

2. Enter the `Baremetal-Framework` directory path and then execute the commands below: (IFC is enabled by default)

```c
make menuconfig
ARM architecture --> [*] Enable baremetal
[*] Enable IFC for baremetal
(1) IFC is assigned to that core
```

### 3.4.4.18.2.8 Ethernet

This section describes the Ethernet configuration settings for LS1043A or LS1046A reference design boards. LS1043A or LS1046A has only one FMan, so you should remove the DPAA driver in Linux.
1. Disable the DPAA driver in Linux kernel:

```
Device Drivers --->
Staging drivers --->
< > Freescale Datapath Queue and Buffer management
```

2. Enter the `Baremetal-Framework` directory and then execute the commands below:

```
make menuconfig
ARM architecture ---> [*] Enable baremetal [*] Enable fman
for baremetal (1) FMAN1 is assigned to that core
```

Configure FMan to the specified core by modifying the `FMan1 is assigned to that core` value, which is the default configuration, to `core1`.

### 3.4.4.18.2.9 USB

This section describes the USB configuration setting for LS1043A and LS1046A reference design boards.

Both LS1043A and LS1046A have three DW3 USB controllers. By default, these are assigned as core1, core2, and core3. Users can reconfigure the controllers by using the `make menuconfig` command as shown below:

```
ARM architecture --->
[*] Enable baremetal
[*] Enable USB for baremetal
(1) USB0 is assigned to core1
(2) USB1 is assigned to core2
(3) USB2 is assigned to core3
(3) USB Controller numbers
```

### 3.4.4.18.2.10 PCIe Express (PCIe)

This section describes the PCIe configuration setting for LS1043A and LS1046A reference design boards.

Both LS1043A and LS1046A have three PCIe controllers. By default, these are assigned as core0, core1, and core2. To reconfigure them, use the command `make menuconfig`, as shown below:

```
ARM architecture --->
[*] Enable baremetal
(0) PCIe1 is assigned to core0
(1) PCIe2 is assigned to core1
(2) PCIe3 is assigned to core2
(3) PCIe Controller numbers
```

### 3.4.4.18.3 LX2160ARDB board

The following sections describe the hardware resource allocation for the LX2160ARDB boards for implementing the supported features.

#### 3.4.4.18.3.1 Memory configuration

This section describes the memory configuration for LX2160ARDB boards.

The LX2160ARDB boards have a 16 GB size DDR. To use the Baremetal framework, configure DDR into three partitions:

- 15G for core0 (Linux)
• 64M per core from core1 to core15 (baremetal), and 64M for shared memory.

The configuration can be defined in the file include/configs/lx2160ardb_config.h.

```c
#define CFG_BAREMETAL_SYS_SDRAM_MASTER_SIZE    (512 * 1024 * 1024UL)
#define CFG_BAREMETAL_SYS_SDRAM_SLAVE_SIZE     (64 * 1024 * 1024UL)
#define CFG_BAREMETAL_SYS_SDRAM_RESERVE_SIZE   (16 * 1024 * 1024UL)
#define CFG_BAREMETAL_SYS_SDRAM_SHARE_SIZE    
  ((64 * 1024 * 1024UL) - CFG_BAREMETAL_SYS_SDRAM_RESERVE_SIZE)
```

The functions included in malloc.h in the table below can be used to allocate or free memory in program. Modify CONFIG_SYS_MALLOC_LEN in include/configs/lx2160ardb.h to change the maximum size of malloc.

<table>
<thead>
<tr>
<th>API name (type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void_t* malloc (size_t n)</td>
<td>Allocates memory</td>
</tr>
<tr>
<td></td>
<td>• &quot;n&quot; – length of allocated chunk</td>
</tr>
<tr>
<td></td>
<td>• Returns a pointer to the newly allocated chunk</td>
</tr>
<tr>
<td>void free (void *ptr)</td>
<td>Releases the chunk of memory pointed to by ptr (where &quot;ptr&quot; is a pointer to</td>
</tr>
<tr>
<td></td>
<td>the chunk of memory)</td>
</tr>
</tbody>
</table>

### 3.4.4.18.4 i.MX 8M Mini LPDDR4 EVK and i.MX 8M Plus LPDDR4 EVK board

#### 3.4.4.18.4.1 Linux DTS

When using Baremetal, users should remove all the devices from kernel that Baremetal has used, for example:

```c
&fec1 {
    status = "disabled";
};
&gpio5 {
    status = "disabled";
};
&uart3 {
    status = "disabled";
};
```

#### 3.4.4.18.4.2 Memory configuration

This section describes the memory configuration for i.MX 8M Mini LPDDR4 EVK or i.MX 8M Plus LPDDR4 EVK boards.

1. The boards have a 6 GB DDR memory. To use the Baremetal framework, configure DDR into five partitions:
   • 6016M for core0 (Linux)
   • 32M for core1 (bare metal)
   • 32M for core2 (bare metal)
   • 32M for core3 (bare metal)
   • 32M for shared memory.
The configuration can be defined in the file `include/configs/imx8mm_baremetal.h` or `include/configs/imx8mp_baremetal.h`.

```c
#define CFG_BAREMETAL_SYS_SDRAM_SLAVE_SIZE   (32 * 1024 * 1024UL)
#define CFG_BAREMETAL_SYS_SDRAM_RESERVE_SIZE (4 * 1024 * 1024UL)
```

2. Memory Reserve

For IPI data transfer, Baremetal needs to share memory between master core and slave core. Hence, users should reserve some memory from the Linux kernel, as shown in the following DT file:

```dts
reserved-memory { #address-cells = <2>; #size-cells = <2>; ranges; bm_reserved: baremetal@0x60000000 { no-map; reg = <0 0x60000000 0 0x10000000>; }; };
```

### 3.4.4.18.4.3 GPIO

1. Connect pin7 and pin8 of J1003. The test_gpio case in Baremetal uses pin7 and pin8 of J1003, so connect these two pins.

2. Boot the Baremetal on slave core. If the GPIO is working fine, the message below is displayed:

```
[ok] GPIO test ok
```

3. Disable the devices from kernel.

For the test_gpio case, use GPIO5_7 (pin8 of J1003) and GPIO5_8 (pin7 of J1003). These two pins are muxed as UART3_TXD and UART3_CTS, so should disable GPIO5 and UART3 from kernel.

```c
&gpio5 { status = "disabled"; };
&uart3 { status = "disabled"; };
```

### 3.4.4.18.4.4 Ethernet

This section describes the Ethernet configuration settings for i.MX 8M Mini LPDDR4 EVK or i.MX 8M Plus LPDDR4 EVK boards.

1. Disable the Ethernet card from dts files:

```c
&fec1 { status = "disabled"; }
```

**Note:**

1. **i.MX 8M Mini LPDDR4 EVK** has only one NIC, default status of eth0(fec1) is disabled. *If user does not use eth0 in Baremetal, can enable fec1 in kernel dts file.*

2. **i.MX 8M Plus LPDDR4 EVK** has two NICs, default setting is eth0 for Baremetal, eth1 for Linux.

2. Confirm Baremetal configuration using the command below:

```
make menuconfig ARM architecture ---> [*] Enable baremetal [*] Enable NIC for baremetal (1) which core that NIC is assigned to
```

Configure NIC to the specified core by modifying the NIC to assign that core value, which is the default configuration, to core1.
3.5 Native RTOS on Cortex-A core

Native RTOS refer to the RTOS running without Hypervisor and is kicked to specified Cortex-A Core by U-Boot commands.

3.5.1 Overview

Real-time Edge system supports Native RTOS including FreeRTOS and Zephyr running on Cortex-A cores. Currently we have two methods to run RTOS on Cortex-A Core:

- Jailhouse RTOS: leverage Jailhouse Hypervisor to run RTOS in Jailhouse inmate cells
- Native RTOS: running on Cortex-A Core without any Hypervisor, similar with BareMetal mode.

Jailhouse Hypervisor provides a mechanism to isolate hardware resources, such as memory and peripherals. However, hypervisor implementation causes Cortex-A core's privileged execution level switching between EL1 and EL2 at runtime, so it introduces extra real-time latency.

Native RTOS is running on Cortex-A Core directly just like Linux kernel running, and there is no any Hypervisor is leveraged, so it has a better Real-time performance compared with RTOS in Jailhouse inmate. It is targeted for high real-time performance use cases with less real-time latency.

In order to run Native RTOS on Cortex-A Core, it needs to define hardware resource used by each OS to make sure there is no resource conflict between different operating systems. If Linux is used with RTOS simultaneously, the device nodes for peripherals used by RTOS should be disabled or removed from Linux device tree.

3.5.2 Building native RTOS on Cortex-A core

There are two methods to build Native RTOS running on Cortex-A Core, one method is to leverage Yocto, another method is to build the image by using ARM gcc directly.

Some native RTOS examples are available in the Heterogeneous Multicore repo, refer to Section 4.2 for how to build Native RTOS.

3.5.3 Booting native RTOS image on Cortex-A core

Native RTOS image for Cortex-A core can be kicked to specific Cortex-A Core by using the U-Boot command. The below example shows how to run hello_world examples on the i.MX 8M Mini EVK, i.MX 8M Plus EVK, and i.MX93 EVK boards.

1. Setup UART console for Native RTOS
   Connect DEBUG UART slot on the board to your PC through the USB Cable. This step creates on the PC two USB serial ports (port0 and port1) for i.MX 8M Mini EVK board, and four USB serial ports (port0 ~ port3) for i.MX 8M Plus EVK and the i.MX93 EVK boards. Open two UART consoles for UART port0 and port1 on i.MX 8M Mini EVK board or port2 and port3 on i.MX 8M Plus EVK board and iMX93 EVK board UART using the following setup:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit
   The first UART console is used for Linux boot up, another one is used for RTOS running on Cortex-A Core.

2. Booting Native RTOS Image
   After powering up the board and entering U-Boot command line, execute the following U-Boot commands to run the hello_world example.
Boot `hello_world` example on the fourth Cortex-A Core on i.MX 8M Mini EVK board using the commands below:

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca53 RTOS0_UART4.bin
=> dcache flush; icache flush
=> cpu 3 release 0x93C00000
```

After the preceding steps are followed, the second UART console displays the following RTOS log:

```
Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
RTOS1: RAM console@0x95bff000
RTOS2: RAM console@0x96bff000
RTOS3: RAM console@0x97bff000
tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac
```

Use the below commands for booting the `hello_world` example on the fourth Cortex-A Core on i.MX 8M Plus EVK board:

```
=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca53 RTOS0_UART4.bin
=> dcache flush; icache flush;
=> cpu 3 release 0xC0000000
```

Booting `hello_world` example on the second Cortex-A Core on i.MX93 EVK board:

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca55 RTOS0_UART2.bin
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

After the preceding steps are followed, the second UART console displays the following RTOS log:

```
Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX93-EVK
RTOS1: RAM console@0xd1fff000
tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac
```

The Native RTOS image can also be booted from the first Cortex-A Core, which is called master Core. Use the same command but use the "go" command to replace "cpu" command. For example, use the command below to boot `hello_world` example on the first Cortex-A Core on i.MX 8M Mini EVK board:

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca53 RTOS0_UART4.bin
=> dcache off; dcache flush; icache flush; icache off
=> go 0x93C00000
```

After the preceding steps are followed, the second UART console displays the following RTOS log:

```
Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
RTOS1: RAM console@0x95bff000
RTOS2: RAM console@0x96bff000
RTOS3: RAM console@0x97bff000
tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac
tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac
```
3.6 RTOS on Cortex-A core with Jailhouse

3.6.1 Jailhouse

3.6.1.1 Overview

Jailhouse is a partitioning Hypervisor based on Linux. It is able to run baremetal applications or (adapted) operating systems besides Linux. For this purpose, it configures CPU and device virtualization features of the hardware platform in a way that none of these domains, called "cells" here, can interfere with each other in an unacceptable way.

Jailhouse is optimized for simplicity rather than feature richness. Jailhouse does not support overcommitment of resources such as CPUs, RAM, or devices. This feature makes it different from full-featured Linux-based hypervisors such as KVM or Xen. It performs no scheduling and only virtualizes those resources in software, which are essential for a platform and cannot be partitioned in hardware.

Once Jailhouse is activated, it runs Baremetal. This implies that it takes full control over the hardware and needs no external support. However, in contrast to other baremetal hypervisors, it requires a normal Linux system to be loaded and configured. Its management interface is based on Linux infrastructure. So, you boot Linux first, then, enable Jailhouse and finally split off parts of the system's resources and assign them to additional cells.

3.6.1.2 Running PREEMPT_RT Linux in Inmate

3.6.1.2.1 i.MX 8M Plus LPDDR4 EVK

Perform the following steps on i.MX 8M Plus LPDDR4 EVK board:

1. Execute `run jh_mmcboot` at U-Boot prompt on the terminal of UART2.
2. Wait for Linux OS to boot up and login.
3. Execute non-root Linux demo (Assuming rootfs have been deployed in `/dev/mmcblk2p2`):

   ```
   # cd /usr/share/jailhouse/scripts
   # ./linux-demo-imx8mp.sh
   ```

4. Check the output on the terminal of UART4:

   ```
   [ 0.717545] printk: console [ttymxc3] enabled
   [ 0.721628] printk: bootconsole [ec_imx6q0] disabled
   [ 0.732428] loop: module loaded
   [ 0.732902] of_reserved_mem_lookup() returned NULL
   [ 0.732952] megasas: 07.714.04.00-rc1
   [ 0.733632] imx ahci driver is registered.
   [ 0.735615] tun: Universal TUN/TAP device driver, 1.6
   [ 0.735835] thunder_xcv, ver 1.0
   [ 0.735863] thunder_bgx, ver 1.0
   [ 0.735889] nicpf, ver 1.0
   [ 0.736340] hclge is initializing
   [ 0.736351] hns3: Hisilicon Ethernet Network Driver for Hip08 Family - version
   [ 0.736354] hns3: Copyright (c) 2017 Huawei Corporation.
   [ 0.736382] e1000: Intel(R) PRO/1000 Network Driver
   [ 0.736384] e1000: Copyright (c) 1999-2006 Intel Corporation.
   [ 0.736416] e1000e: Intel(R) PRO/1000 Network Driver
   [ 0.736418] e1000e: Copyright(c) 1999 - 2015 Intel Corporation.
   [ 0.736447] igb: Intel(R) Gigabit Ethernet Network Driver
   [ 0.736450] igb: Copyright (c) 2007-2014 Intel Corporation.
   ```
NXP Real-time Edge Distro 2.2 imx8mp-lpddr4-evk ttymxc3
root@imx8mp-lpddr4-evk:~#
root@imx8mp-lpddr4-evk:~# uname -a
Linux imx8mp-evk 5.10.72-rt53-lts-5.10.y+g5304e555731 #1 SMP PREEMPT_RT Tue Mar 1 06:03:05 UTC 2022 aarch64 aarch64 aarch64 GNU/Linux
root@imx8mp-lpddr4-evk:~#

Note: If the case fails because of rootfs error, update rootfs using the following command:

# rm -fr /run/media/mmcblk2p2/*
# cp -frd /usr /bin /etc /home /fat /lib /linuxrc /lost+found /media /mnt /opt /root /sbin /run/media/mmcblk2p2/

5. Exit Jailhouse.

3.6.1.2.2 LS1028ARDB
Perform the steps listed in the following section to run PREEMPT_RT Linux on LS1028ARDB board.

3.6.1.2.2.1 Linux in non-root cell
Perform the following steps to run PREEMPT_RT Linux in Inmate on LS1028ARDB platform:

1. Execute `run jh_mmcboot` from U-Boot prompt.
2. Wait for Linux OS to boot up and log in it.
3. Execute non-root Linux demo:

```
# cd /usr/share/jailhouse/scripts
# ./linux-demo-ls1028ardb.sh
```

4. Exit Jailhouse.

```
# ../tools/jailhouse disable
```

3.6.1.2.2.2 ENETC in non-root cell
Follow the below steps for ENETC that is assigned to non-root cell:

1. Under U-Boot prompt, run the below commands to set the device tree blob, which has ENETC nodes removed and then boot up Linux:

```
=> setenv jh_mmcboot `setenv dtb fsl-ls1028a-rdb-jailhouse-without-enetc.dtb;run bootcmd`
=> run jh_mmcboot
```

2. Wait for Linux OS to boot up and then log in.
3. Execute non-root Linux demo:

```
# cd /usr/share/jailhouse/scripts
# ./linux-demo-ls1028ardb-enetc.sh
```

Then, network can be available in none-root cell Linux.

4. Exit Jailhouse.

```
# ../tools/jailhouse disable
```
Note:
In this case, the GICv3 ITS node is also removed from the root cell Linux device tree. The node is assigned to non-root cell to service ENETC MSI-X interrupts, so the root cell Linux does not support the MSI/MSI-X service anymore.

3.6.1.2.2.3 GPIO in non-root cell

GPIO3 controller is assigned to non-root cell, below steps is for GPIO that is assigned to non-root cell:

1. Hardware setup
   Connect J11 Pin5 (1588_ALARM_OUT1/GPIO3_DAT11) to Pin 8 (1588_CLK_OUT/GPIO3_DAT10)

2. RCW setting
   In `/dash-rcw/ls1028ardb/R_SQPP_0x85bb/rcw_1500_sdboot.rcw`, change as below:
   ```
   EC1_SAI4_5_PMUX=1
   EC1_SAI3_6_PMUX=1
   ```
   
   **EC1_SAI4_5_PMUX is set to 0b001, EC1_SAI3_6_PMUX is set to 0b001 to select GPIO.**

3. Software configuration required:
   a. Configure CPLD register BRDCFG3 (offset 053h) bit 2 to 0 (IEEE signals connect to the IEEE header) in U-Boot prompt:
      ```
      => i2c mw 66 53 00
      ```
   b. Boot up Linux using Jailhouse DTB and bring up non-root Linux:
      ```
      => run jh_mmcboot
      ```
   c. Wait for Linux OS to boot up and login.
   d. Execute non-root Linux demo.
      ```
      # cd /usr/share/jailhouse/scripts
      # ./linux-demo-ls1028ardb.sh
      ```

4. Test GPIO function in non-root Linux.
   a. Export GPIO pin
      ```
      # ls /sys/class/gpio
      # echo 490 > /sys/class/gpio/export
      # echo 491 > /sys/class/gpio/export
      ```
   b. Configure GPIO output and input.
      ```
      # echo out > /sys/class/gpio/gpio490/direction
      # cat /sys/class/gpio/gpio490/direction
      # cat /sys/class/gpio/gpio491/direction
      ```
   c. Verify write 1 to GPIO output.
      ```
      # echo 1 > /sys/class/gpio/gpio490/value
      # cat /sys/class/gpio/gpio490/value
      # cat /sys/class/gpio/gpio491/value
      ```
   d. Verify write 0 to GPIO output.
      ```
      # echo 0 > /sys/class/gpio/gpio490/value
      # cat /sys/class/gpio/gpio490/value
      # cat /sys/class/gpio/gpio491/value
      ```
5. Exit Jailhouse

```bash
# ../tools/jailhouse disable
```

### 3.6.1.2.3 LS1046ARDB

Perform the following steps:

1. Execute `run jh_mmcboot` in U-Boot stage.
2. Wait for Linux OS to boot up and login in it.
3. Execute non-root Linux demo:

```bash
# cd /usr/share/jailhouse/scripts
# ./linux-demo-ls1046ardb.sh
```

4. Exit Jailhouse:

```bash
# ../tools/jailhouse disable
```

### 3.6.1.3 Running Jailhouse examples In Inmate

#### 3.6.1.3.1 i.MX 8M Plus LPDDR4 EVK

1. Execute `run jh_mmcboot` in U-Boot stage
2. Wait for Linux OS to boot up and login in it.
3. Execute GIC demo.

```bash
# cd /usr/share/jailhouse/scripts
# ./gic-demo-imx8mp.sh
```

4. Check the result on serial port:

```
Initializing the GIC...
Initializing the timer...
---
Timer fired, jitter: 2039 ns, min: 2039 ns, max: 2039 ns
Timer fired, jitter: 1039 ns, min: 1039 ns, max: 1039 ns
Timer fired, jitter: 879 ns, min: 879 ns, max: 879 ns
Timer fired, jitter: 959 ns, min: 959 ns, max: 959 ns
Timer fired, jitter: 1039 ns, min: 1039 ns, max: 1039 ns
Timer fired, jitter: 919 ns, min: 919 ns, max: 919 ns
Timer fired, jitter: 919 ns, min: 919 ns, max: 919 ns
Timer fired, jitter: 1079 ns, min: 1079 ns, max: 1079 ns
Timer fired, jitter: 919 ns, min: 919 ns, max: 919 ns
Timer fired, jitter: 959 ns, min: 959 ns, max: 919 ns
```

5. Execute UART demo:

```bash
# ./uart-demo-imx8mp.sh
```

6. Check the result on serial port:
7. Exit Jailhouse.

```
# ../tools/jailhouse disable
```

### 3.6.1.3.2 LS1028ARDB Jailhouse example in Inmate

Perform the following steps for running LS1028ARDB Jailhouse example In Inmate:

1. Execute `run jh_mmcboot` in U-Boot stage.
2. Wait for Linux OS to boot up and then log in.
3. Execute GIC demo using the command below:

   ```
   # cd /usr/share/jailhouse/scripts
   # ./gic-demo-ls1028ardb.sh
   ```

4. Execute UART demo using the command below:

   ```
   # ./uart-demo-ls1028ardb.sh
   ```

5. Execute ivshmem demo using the command below:

   ```
   # ./ivshmem-demo-ls1028ardb.sh
   ```

**Note:** If ivshmem case fails, then, reboot the board and test the case again.

Check the result on the second serial port:

3.6.1.3.3 LS1046ARDB Jailhouse example

Perform the below steps for running Jailhouse examples in Inmate on LS1046ARDB:

1. Execute run `jh_mmcboot` in U-Boot stage.
2. Wait for Linux OS to boot up and login it.
3. Execute GIC demo:
   ```
   # cd /usr/share/jailhouse/scripts
   # ./gic-demo-ls1046ardb.sh
   ```
4. Execute UART demo:
   ```
   # ./uart-demo-ls1046ardb.sh
   ```
5. Execute ivshmem demo:
   ```
   # ./ivshmem-demo-ls1046ardb.sh
   ```
6. Exit Jailhouse:
   ```
   # ../tools/jailhouse disable
   ```

3.6.2 Harpoon (RTOS on Cortex-A)

All information provided in this document is subject to legal disclaimers.
© 2023 NXP B.V. All rights reserved.
3.6.2.1 Overview

Harpoon RTOS provides an environment for developing real-time demanding applications on an RTOS running on one (or several) Cortex-A core(s) in parallel of a Linux distribution.

Harpoon leverages Jailhouse to partition the hardware and run the RTOS as a Linux guest.

The Harpoon RTOS is based on either FreeRTOS or Zephyr plus MCUXpresso drivers and provides several example applications:

- Audio application
- Industrial application
- Real-time latency test application

For details about Harpoon OS, refer to its user guide available at the following location:

3.7 RTOS and Baremetal on Cortex-M core

Regarding RTOS and Baremetal building, please refer to Section 3.2

Real-time Edge images has some demo for testing.

For more example, please refer to https://mcuxpresso.nxp.com/en/welcome

3.7.1 Booting Cortex-M Core RTOS Image

There are two ways to boot ARM Cortex-M Core: booting from U-Boot, or using RemoteProc to boot from Linux.

3.7.1.1 Booting Native RTOS Cortex-M Core image from U-Boot

U-boot command "bootaux" is used to boot Cortex-M Core RTOS Image from U-Boot.

For example, after the board is booted into the U-Boot console, use the following command to boot Arm Cortex-M core on i.MX 8M Mini EVK board or i.MX 8M Plus EVK board:

```bash
=> ext4load mmc 1:2 0x48000000 /examples/mcux-sdk/freertos-hello/release/ freertos_hello.bin; cp.b 0x48000000 0x7e0000 20000;
=> bootaux 0x7e0000
```

Or use the following command on i.MX93 EVK board:

```bash
=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/hello-world-cm/ release/hello_world_cm33_UART2.bin; cp.b 0xd0000000 0x201e0000 20000;
=> bootaux 0x1ffe0000
```

3.7.1.2 Using RemoteProc to boot RTOS Cortex-M Core Image

If you choose to use RemoteProc to start the remote core directly, execute run prepare_mcore in U-Boot before starting the Linux OS.

```bash
=> run prepare_mcore
```
Then, use the following command to use RPMSG dtb file to boot the kernel:

```bash
# On imx8mm-lpddr4-evk board
=> setenv fdtfile imx8mm-evk-rpmsg.dtb
=> boot

# On imx8mp-lpddr4-evk board
=> setenv fdtfile imx8mp-evk-rpmsg.dtb
=> boot
```

Then, after the Linux kernel boots up, run the commands for i.MX 8MP:

```bash
root@imx8mp-lpddr4-evk:~# echo -n imx8mp_m7_TCM_hello_world.elf > /sys/class/remoteproc/remoteproc0/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/class/remoteproc/remoteproc0/state
[ 19.668712] remoteproc remoteproc0: powering up imx-rproc
[ 19.670341] remoteproc remoteproc0: Booting fw image
  imx8mp_m7_TCM_hello_world.elf, size 153316
root@imx8mp-lpddr4-evk:~# [ 20.191036] remoteproc remoteproc0: remote processor imx-rproc is now up
```

For i.MX 8MM, run the following commands:

```bash
root@imx8mm-lpddr4-evk:~# echo -n imx8mm_m4_TCM_hello_world.elf > /sys/class/remoteproc/remoteproc0/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/class/remoteproc/remoteproc0/state
[ 209.654414] remoteproc remoteproc0: powering up imx-rproc
[ 209.656646] remoteproc remoteproc0: Booting fw image
  imx8mm_m4_TCM_hello_world.elf, size 146136
root@imx8mm-lpddr4-evk:~# [ 210.174456] remoteproc remoteproc0: remote processor imx-rproc is now up
```

After these steps are followed, the remote processor `imx-rproc` is up.
4 Heterogeneous Multicore Framework

4.1 Overview

Heterogeneous Multicore Framework provides a general software framework to support Heterogeneous AMP. It enables AMP to be interconnected and provides a unified resource management and life-cycle management.

It provides the below key functions to help users to accelerate solution development based on multicore platforms:

1. **Data communication between different operating systems**
   The following technical implementation can be used to pass common data between different operating systems. The data transfer can be between Cortex-M Core and Cortex-A Core, or between different Cortex-A Cores, or between multiple CPU Cores simultaneously.
   - **RPMsg**
     RPMsg is a standard intercore communication protocol, it is supported on Linux and RTOS.
   - **Heterogeneous Multicore VirtIO**
     Heterogeneous Multicore VirtIO applies para-virtualization VirtIO technology to build high-performance inter-core data path, customized data path is defined according to different use cases.

2. **Resource sharing between different operating systems**
Resource sharing enables sharing physical resources between different OSes. In general, one OS owns and controls hardware resource while the other OS uses a virtual device. The following mechanism is followed to build the control path and data path to access physical resource.

- **RPMSG**
  Use RPMSG to build control and data path crossing OS, physical resource is shared with another OS in terms of virtual device. Simplified Real-Time Messaging (SRTM) protocol provided in Real-time Edge is an implementation based on RPMsg. It is used to share the physical resources of the Cortex-M core with Cortex-A core in terms of virtual device in Linux.

- **Heterogeneous Multicore VirtIO**
  Heterogeneous Multicore VirtIO have a better performance than RPMSG, and it can also be used for resource sharing. POSIX compatible API can be used to access virtual device, and some existing VirtIO device drivers in Linux can be reused. Networking sharing is provided in Real-time Edge to share the same networking interface between multiple OSes.

3. **Unified Life-Cycle Management**
   Heterogeneous Multicore Framework provides unified Life Cycle Management both for Cortex-A Core and Cortex-M Core.

There are some sample application provided in Heterogeneous Multicore Framework, these application can be used to demo and evaluate the features in Heterogeneous Multicore Framework:

1. **hello_world**
   hello_world application is to demonstrate flexible Real-time System on MPU platforms, multiple images provided can be used to run single or multiple RTOS on Cortex-A Core or Cortex-M with or without ruing Linux simultaneously.

2. **RPMSG Applications**
   Heterogeneous Multicore Framework supports RPMSG communication between any Real-time Systems on MPU Platforms, such as:
   - RPMSG between RTOS on Cortex-M Core and Linux on Cortex-A core
   - RPMSG between RTOS on Cortex-A Core and Linux on Cortex-A Core
   - RPMSG between RTOS on Cortex-A Core and RTOS on Cortex-A Core
   - RPMSG between RTOS on Cortex-M Core and RTOS on Cortex-A Core

   The following applications provides filed trail for RPMSG related features
   - **rpmsg_str_echo**
     This demo demonstrates building up multiple RPMSG endpoints between RTOS and Linux. For example, on i.MX 8M Plus EVK board, images provided in Real-time Edge can be used to run three RTOS rpmsg_str_echo applications on two Cortex-A Core and one Cortex-M Core, the other two Cortex-A Cores run SMP Linux, then each RTOS will establish three RPMSG Channels with Linux.
   - **rpmsg_pingpong**
     This demo is to demonstrate RPMSG communication between RTOS and RTOS, one is RPMSG master and the other is RPMSG slave.

   - **rpmsg_perf**
     rpmsg_perf is a tool to evaluate RPMSG bandwidth performance between RTOS and Linux Kernel.

   - **RPMSG enhanced 8MB buffer**
     The application is to demonstrate how to change default RPMSG buffer size and count.

   - **UART Sharing based on RPMSG**
     This application is to demonstrate how to use RPMSG to share physical peripherals or other resource between different CPU Core or OS. This demo share physical UART controlled by Cortex-M Core with Cortex-A Core on which virtual UART device driver is provided in Linux.

3. **Heterogeneous Multicore VirtIO Applications**
   - **virtio_perf**
     virtio_perf is a tool to evaluate Heterogeneous Multicore VirtIO bandwidth performance between RTOS and Linux.
• VirtIO Networking Sharing

This application is to demonstrate how to use Heterogeneous Multicore VirtIO to share physical peripherals or other resource between different CPU Core or OS. The application provides networking sharing, physical networking interface is controlled by Cortex-M Core or Cortex-A Core, then it is shared with Cortex-A Core on which virtual NIC device driver is provided in Linux. Heterogeneous Multicore VirtIO is used to establish high performance data path between two sides.

The following table shows the support matrix on NXP platforms:

Table 25. Heterogeneous Multicore Application Support Matrix

<table>
<thead>
<tr>
<th>Feature</th>
<th>Sub-feature</th>
<th>Application</th>
<th>i.MX 8M Mini LPDDR4 EVK</th>
<th>i.MX 8M Plus LPDDR4 EVK</th>
<th>i.MX 93 EVK</th>
<th>i.MX 93 9x9 LPDDR4 QSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexible Real-time System</td>
<td>Flexible Real-time System</td>
<td>hello_world</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RAM Console</td>
<td>hello_world</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>networking stack on A-Core RTOS</td>
<td>lwip_ping</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unified Life Cycle Management</td>
<td>U-Boot booting Native RTOS on A-Core</td>
<td>hello_world</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>U-Boot booting Native RTOS on M-Core</td>
<td>hello_world</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Linux booting Native RTOS on M-Core</td>
<td>remoteproc</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RPMSG</td>
<td>rpmsg_str_echo</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>RPMSG</td>
<td>rpmsg_str_echo</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPMSG</td>
<td>rpmsg_pingpong</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPMSG</td>
<td>rpmsg_lite_str_echo_rtos</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPMsg Performance Evaluation</td>
<td>rpmsg_perf</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART Sharing based on RPMsg</td>
<td>rpmsg_lite_uart_sharing_rtos</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous Multicore VirtIO</td>
<td>virtio_perf</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heterogeneous Multicore VirtIO</td>
<td>virtio_net_backend</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
4.2 Building Heterogeneous Multicore RTOS Application

Heterogeneous Multicore Framework provides some RTOS applications in the repo: `heterogeneous-multicore`.

There are two methods to build Heterogeneous Multicore RTOS Applications, one method is to leverage Yocto, another method is to standalone method by using ARM gcc directly.

4.2.1 Build with Yocto

Real-time Edge supports to build all images by using Yocto, so please refer to "Section 3.2 Building, deploying, and releasing unified software" for how to leverage Yocto to build RTOS Applications on Cortex-A Core an

Cortex-M Core.

The followings are some Yocto quick commands:

Build all RTOS application running both on Cortex-A Core and Cortex-M Core:

```
bitbake packagegroup-real-time-edge-rtos
```

Build single Heterogeneous Multicore RTOS application separately:

```
bitbake APP-NAME-CORE
```

The "APP-NAME-CORE" could be the following applications with suffix ":ca" for A-Core application or ":cm" for M-Core application:

- hello-world-ca
- lwip-ping-ca
- rpmsg-str-echo-ca
- rpmsg-str-echo-cm
- rpmsg-pingpong-ca
- rpmsg-lite-uart-sharing-rtos
- virtio-net-backend-ca
- virtio-net-backend-cm
- virtio-perf-ca
- virtio-perf-cm

4.2.2 Build with Standalone Mode

Some RTOS applications are in the repository: `heterogeneous-multicore`, the repo is managed by "west" tool, so can use "west" to download all the software components and then use ARM gcc toolchain to build the application directly.

1. Download and install the toolchain

The toolchain for building RTOS on Cortex-A Core:

```
mkdir ~/toolchains/; cd ~/toolchains/
wget https://developer.arm.com/-/media/Files/downloads/gnu-a/10.3-2021.07/binrel/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf.tar.xz
tar xf gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf.tar.xz
```

The toolchain for building RTOS on Cortex-M Core:

```
cd ~/toolchains/
```
2. Download Source Code

This Heterogeneous Multicore project uses west to manage all related repos, west.yml provides the
description and revision for other projects used by Heterogeneous Multicore.

Use the following command to download all the source code:

```bash
tar xf gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2
```

```bash
west init -m https://github.com/nxp-real-time-edge-sw/heterogeneous-multicore.git workspace
cd workspace
west update
```

3. Build the RTOS application.

Each application provides a building script which can be used to build the application separately. In the
meanwhile, a common building helper script "build_all.sh" in the root directory of "heterogeneous-multicore"
can be used to build single or all application for all boards.

- **Build Single Application for Single Board**

  Take "rpmsg_str_echo" application as example, use the following command to build the application for
  Cortex-A Core on i.MX 8M Plus EVK:

  ```bash
  export ARMGCC_DIR=~/toolchains/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf
cd ~/workspace/heterogeneous-multicore/apps/rpmsg_str_echo/freertos/boards/evkmimx8mp_ca53/armgcc_aarch64
  ./build_ddr_release.sh
  ```

  Then the RTOS images "rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE.bin"
rpmmsg_str_echo_ca53_RTOS0_UART4.bin rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE.bin" can be found in directory "ddr_release".

  And use the following commands to build the application for Cortex-M Core on i.MX 8M Plus EVK:

  ```bash
  export ARMGCC_DIR=~/toolchains/gcc-arm-none-eabi-10-2020-q4-major
cd ~/workspace/heterogeneous-multicore/apps/rpmsg_str_echo/freertos/boards/evkmimx8mp_cm7/armgcc
  ./build_release.sh
  ```

  Then the RTOS image "rpmmsg_str_echo_cm7.bin" can be found in directory "release".

- **Build Multiple Applications**

  "build_all.sh" in the root directory of "heterogeneous-multicore" can be used to build single or all
  application for all boards.

  The following is help information for "build_all.sh" tool:

  ```bash
  ./build_all.sh core_type [directory]               - build all a-core or m-core applications in [directory]
  ./build_all.sh [core_type] [directory] clean       - clean all a-core or m-core or for both applications building in [directory]
  ```

  core

  ```bash
  For example:
  ./build_all.sh a-core
  ```

  ```bash
  applications in apps directory
  ./build_all.sh m-core apps/hello_world
  ```

  ```bash
  applications in apps/hello_world directory
  ./build_all.sh clean
  ```

  ```bash
  building in apps directory
  ./build_all.sh a-core clean
  ```

  ```bash
  applications building in apps directory
  ./build_all.sh m-core apps/hello_world clean
  ```

  ```bash
  applications building in apps/hello_world directory
  ```
4.3 Flexible Real-time System

On NXP MPU platforms, it supports Flexible Real-time System, the system can run single or multiple RTOS on Cortex-M Core and Cortex-A Core with or without running Linux on Cortex-A Core simultaneously.

The system provides RAM Console to make it easy to debug multiple OS in case of no enough physical UART Console can be used.

And the system also provides some common software stack, such as lwIP networking stack on Cortex-A Core or Cortex-M Core.
4.3.1 Heterogeneous Multicore RAM Console

RAM Console is a virtual debug console which can be used by RTOS, it provides common console APIs to print and save console log to a reserve memory region.

If multiple OSes run on MPU platform, some OSes can use physical UART as debug console, then the other OSes can use RAM Console, then can dump the other OS's RAM Console from the OS which is using physical UART Console.

It provides two methods to dump the RAM Console log: use U-Boot command or Linux Userspace tool.

4.3.1.1 Use RAM Console in RTOS

This chapter describes how to develop RTOS application based on RAM Console.

4.3.1.1.1 RAM Console Technical Details

RAM Console driver locates at debug console (utilities/debug_console) in the repository mcux-sdk.

There is a 64 bytes Console Header at the start of RAM Console log memory, the memory layout is as the following figure:

---

The RAM Console Header includes the following parts:

- 16 bytes fixed string flag: "RAM_CONSOLE"
- console_start_address: the memory physical address for the console_start of Console Buffer
- console_buffer_length: the length in bytes of Console Buffer from console_start to console_end
- console_cursor_position: current Console cursor related position from console_start, the cursor is started from console_start, then move towards console_end with console log increase, when it reaches to console_end, it will jump back to console_start, and then move towards console_end again.
4.3.1.1.2 Develop with RAM Console

Take the "hello_world" program for the "evkmimx8mm_ca53" board in the repository heterogeneous-multicore as an example. Then follow the steps listed below to use RAM Console as debug console in RTOS application.

1. Reserve a memory block for RAM Console, and add mmu mapping entry in app_mmu.h.
2. In application's CMakeLists.txt

   ```
   ...
   SET(DEBUG_CONSOLE_CONFIG "-DSDK_DEBUGCONSOLE=2 -DCONFIG_RAM_CONSOLE")
   ...
   include(utility_assert_ram_console)
   include(utility_ram_console)
   ...
   
   3. Call RamConsole_Init() with RAM Console memory block’s address and size to initialize RAM Console

   ```
   #ifdef CONFIG_RAM_CONSOLE
   RamConsole_Init(RAM_CONSOLE_ADDR, RAM_CONSOLE_SIZE);
   #else
   BOARD_InitDebugConsole();
   #endif
   ```

4. In general, while running multiple RTOS instances, all the RAM Console memory addresses can be printed out to physical debug console from the RTOS instances using physical debug console.

4.3.1.1.3 Dump RAM Console Log

RAM Console Log can be dumped from U-Boot command line or from Linux userspace. While running multiple OSes on MPU platform, the OS instances must be started using RAM Console first. Then start the OS using physical UART Cone. If the last OS using physical UART Cone is RTOS, it can dump the other OS's RAM Console log before from U-Boot command line start this last RTOS, otherwise it has to dump the console memory by using JTAG tools. But if the last OS using physical UART Cone is Linux, we can still dump the other OS's RAM Console log by using Linux userspace tool after Linux boots up.

- Dump from U-Boot

  In U-Boot command line, use "md" command to dump the whole RAM Console memory including RAM Console Header

  ```
  u-boot=> dcache flush; md C1FFF000
  c1fff000: 5f4d4152 534e4f43 00454c4f 00000000 RAM_CONSOLE.....
  c1fff010: c1fff040 00000000 00000fc0 00000000 ........
  c1fff020: 0000024d 00000000 00000000 00000000 M............
  c1fff030: 00000000 00000000 00000000 00000000 ............
  c1fff040: 6f430a0d 78657472 3335412d 5452203a ..Cortex-A53: RT
  c1fff050: 3a31534f 66544820 7720666c 646c726f OS1: Hello world
  c1fff060: 655220d3 742d6c61 2d65669 65766444 ! Real-time Edge
  c1fff070: 20666f20 584d494d 2d504d38 0d4b5645 on MIMX8MP-EVK.
  c1fff080: 6369740a 63617420 63697420 63617420 tic tac tic tac
  c1fff090: 63697420 63617420 63697420 63617420 tic tac tic tac
  c1fff0a0: 63697420 63617420 63697420 63617420 tic tac tic tac
  c1fff0b0: 63697420 63617420 63697420 63617420 tic tac tic tac
  c1fff0c0: 63697420 63617420 63697420 63617420 tic tac tic tac
  ```

- Dump from Linux

  There is a Linux userspace tool provided in the repository heterogeneous-multicore "tool" directory. Use this tool to dump RAM Console log.

  ```
  root@imx8mp-1pddr4-evk:~# ram_console_dump 0xC1FFF000
  ```
4.3.2 Heterogeneous Multicore hello_world

4.3.2.1 Overview

Heterogeneous Multicore hello_world application demonstrates flexible Real-time System on MPU platforms. It can run single or multiple RTOS on Cortex-A Core or Cortex-M with or without running Linux simultaneously.

Take i.MX 8M Plus Applications Processor as an example. It has four Cortex-A53 cores and one Cortex-M7 Core. It could run the following use cases on this MPU platform:

<table>
<thead>
<tr>
<th>ID</th>
<th>M7</th>
<th>A53</th>
<th>A53</th>
<th>A53</th>
<th>A53</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RTOS</td>
<td>A53</td>
<td>SMP Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>RTOS</td>
<td>A53</td>
<td>SMP Linux</td>
<td>RTOS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>RTOS</td>
<td>A53</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
<tr>
<td>3</td>
<td>RTOS</td>
<td>Linux</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
<tr>
<td>4</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
</tbody>
</table>

4.3.2.2 Technical Points

- **Debug Console**
  Take i.MX 8M Plus Applications Processor as an example, in general, UART4 is used for Cortex-M Core RTOS or Cortex-A Core RTOS, UART2 is used for Linux, the other RTOS instance can use RAM Console. The hello_world application could build all RTOS instances with any possible UART Console or RAM Console, the following images are for i.MX 8M Plus EVK, the RTOS0 provides images both for UART4 Console and RAM Console, can run any of them according to usecase setup.

```
├── evkmimx8mp
│   ├── hello_world_ca53_RTOS0_RAM_CONSOLE.bin
│   ├── hello_world_ca53_RTOS0_UART4.bin
│   ├── hello_world_ca53_RTOS1_RAM_CONSOLE.bin
│   ├── hello_world_ca53_RTOS2_RAM_CONSOLE.bin
│   └── hello_world_ca53_RTOS3_RAM_CONSOLE.bin
```

- **GIC Controller Initialization**
  If run multiple RTOS on MPU's Cortex-A, it can only setup GIC's distributor by the first RTOS to be run. For example, the following code in hello_world’s main.c to make sure that only RTOS0 to setup GIC’s distributor.

```c
#if (RTOSID == 0)
    GIC_Enable(1);
#else
    GIC_Enable(0);
```
So in general, RTOS0 need to run firstly before running the other RTOS instances.

- Memory Usage

For RTOS applications in the repository: heterogeneous-multicore, the memory resource used by RTOS kernel is defined in "os/freertos/Core_AArch64/boards/<PLAT_NAME>/rtos_memory.h", for example, in the following rtos_memory.h for i.MX 8M Plus, RTOS0, RTOS1 and RTOS2, each RTOS uses 16M bytes memory from 0xC0000000, 0xC1000000 or 0xC2000000.

```c
#ifndef _RTOS_MEMORY_H_
#define _RTOS_MEMORY_H_

/* Memory used by RTOS kernel */
#define M_INTERRUPTS_BASE       (0xC0000000 + 0x1000000 * RTOSID)
#define M_INTERRUPTS_LEN        0x00002000 /* 8 kB */
#define M_TEXT_BASE             (M_INTERRUPTS_BASE + M_INTERRUPTS_LEN)
#define M_TEXT_LEN              0x005FE000 /* ~6 MB */
#define M_DATA_BASE             (M_TEXT_BASE + M_TEXT_LEN)
#define M_DATA_LEN              0x005FE000 /* ~6 MB */
#define M_STACKS_BASE           (M_DATA_BASE + M_DATA_LEN)
#define M_STACKS_LEN            0x00002000 /* 8 kB */
#define M_STACKS_NC_BASE        (M_STACKS_BASE + M_STACKS_LEN)
#define M_STACKS_NC_LEN         0x003FF000 /* ~4 MB */

/* Memory used by RAM Console */
#define RAM_CONSOLE_ADDR        (M_STACKS_NC_BASE + M_STACKS_NC_LEN)
#define RAM_CONSOLE_SIZE        0x00001000 /* 4KB */
#define RTOS_MEM_LEN            0x01000000 /* 16 MB */

#endif
```

If run Linux kernel with RTOS simultaneously, need to reserve the memory space used by RTOS in device tree, for example, in imx8mp-evk-multicore-rtos.dts:

```dts
ca53_reserved: ca53@c0000000 {
  no-map;
  reg = <0 0xc0000000 0x0 0x3000000>;
};

m7_reserved: m7@80000000 {
  no-map;
  reg = <0 0x80000000 0 x10000000>;
};
```

4.3.2.3 Running flexible multicore hello_world application
4.3.2.3.1 Running use cases on i.MX 8M Plus LPDDR4 EVK

The following use cases can run on i.MX 8M Plus LPDDR4 EVK:

Table 27. Flexible Real-time System on i.MX 8M Plus

<table>
<thead>
<tr>
<th>ID</th>
<th>M7</th>
<th>A53</th>
<th>A53</th>
<th>A53</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RTOS</td>
<td>SMP Linux</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>RTOS</td>
<td>SMP Linux</td>
<td>RTOS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>RTOS</td>
<td>SMP Linux</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
<tr>
<td>3</td>
<td>RTOS</td>
<td>Linux</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
<tr>
<td>4</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
</tbody>
</table>

By default, Cortex-M Core's RTOS uses UART4 as debug Console. Refer to Section 3.7.1 for how to boot Cortex-M Core's RTOS image.

Cortex-A Core's RTOS could run with UART4 Console, RAM Console or UART2 Console, the following Cortex-A Core's RTOS images are provided:

```
├── evkmimx8mp
│   ├── hello_world_ca53_RTOS0_RAM_CONSOLE.bin
│   ├── hello_world_ca53_RTOS0_UART4.bin
│   ├── hello_world_ca53_RTOS1_RAM_CONSOLE.bin
│   ├── hello_world_ca53_RTOS2_RAM_CONSOLE.bin
│   ├── hello_world_ca53_RTOS3_RAM_CONSOLE.bin
│   └── hello_world_ca53_RTOS3_UART2.bin
```

RAM Console address for each RTOS:

- RTOS0: RAM console@0xc0fff000
- RTOS1: RAM console@0xc1fff000
- RTOS2: RAM console@0xc2fff000
- RTOS3: RAM console@0xc3fff000

If run hello_world_ca53_RTOS0_UART4.bin, it will print all the RAM Console's address except RTOS0.

Refer to Section 3.5.3 for how to boot RTOS on Cortex-A Core, and refer to Section 4.3.1.1.3 for how to dump RAM Console log.

Take Usecase #2 as example, follow the following steps to boot multiple OS on the platform:

- **Boot Cortex-M Core's RTOS**
  
  ```
  u-boot=> ext4load mmc 1:2 0x48000000 /examples/mcux-sdk/freertos-hello/release/freertos_hello.bin;
  u-boot=> cp.b 0x48000000 0x7e0000 20000;
  u-boot=> bootaux 0x7e0000
  ```

  Then will find the following log from UART4:

  ```
  Hello world.
  ```

- **Boot the first Cortex-A Core's RTOS on Core2**
  
  ```
  u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca53_RTOS0_RAM_CONSOLE.bin
  u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
  ```
Then check the RAM Console's log as follows:

```
// example log output
5f4d4152 534e4f43 00454c0f 00000000  RAM_CONSOLE.....
00000000 00000000 00000000 00000000 00000000 00000000
6f430a0d 78657472 3335412d 5452203a ..Cortex-A53: RT
6f430a0d 78657472 3335412d 6f73666f .RTOS2: RAM consol
```

• Boot the second Cortex-A Core's RTOS on Core3:

```
u-boot=> ext4load mmc 1:2 0xC1000000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca53_RTOS1_RAM_CONSOLE.bin
u-boot=> dcache flush; icache flush; cpu 3 release 0xC1000000
```

Then check the RAM Console's log as follows:

```
5f4d4152 534e4f43 00454c0f 00000000  RAM_CONSOLE.....
00000000 00000000 00000000 00000000 00000000 00000000
6f430a0d 78657472 3335412d 5452203a ..Cortex-A53: RT
6f430a0d 78657472 3335412d 6f73666f .RTOS2: RAM consol
```

• Boot SMP Linux from Cortex-A Core0 and Core1:

If M-Core is booting up, use `imx8mp-evk-multicore-rpmsg.dtb`
```
u-boot => setenv fdtfile imx8mp-evk-multicore-rpmsg.dtb
```

Otherwise, can also use `imx8mp-evk-multicore-rtos.dtb`:
```
u-boot => setenv fdtfile imx8mp-evk-multicore-rtos.dtb:
```

Then, boot up the kernel using the command below:
```
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => boot
```

After Linux kernel boots up, check Cortex-A Core's RTOS log from the RAM console:
```
root@imx8mp-lpddr4-evk:~# ram_console_dump 0xC0FFF000
RAM Console@0xc0fff000:
```
4.3.2.3.2 Running use cases for i.MX 8M Mini LPDDR4 EVK

The following use cases can run on the i.MX 8M Mini LPDDR4 EVK:

Table 28. Flexible Real-time System on i.MX 8M Mini

<table>
<thead>
<tr>
<th>ID</th>
<th>M4</th>
<th>A53</th>
<th>A53</th>
<th>A53</th>
<th>A53</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RTOS</td>
<td>SMP Linux</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>RTOS</td>
<td>SMP Linux</td>
<td>RTOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>RTOS</td>
<td>SMP Linux</td>
<td>RTOS</td>
<td>RTOS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RTOS</td>
<td>Linux</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
<tr>
<td>4</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
</tbody>
</table>

By default, RTOS of the Cortex-M Core uses UART4 as debug Console. Refer to Section 3.7.1 for how to boot Cortex-M Core's RTOS image.

RTOS of the Cortex-A Core can run with UART4 Console, RAM Console, or UART2 Console. The following Cortex-A Core's RTOS images are provided:

```bash
evkimx8mm
  hello_world_ca53_RTOS0_RAM_CONSOLE.bin
  hello_world_ca53_RTOS0_UART4.bin
  hello_world_ca53_RTOS1_RAM_CONSOLE.bin
  hello_world_ca53_RTOS2_RAM_CONSOLE.bin
  hello_world_ca53_RTOS3_RAM_CONSOLE.bin
  hello_world_ca53_RTOS3_UART2.bin
```

RAM Console address for each RTOS:

- RTOS0: RAM console@0x94bfff000
- RTOS1: RAM console@0x95bfff000
- RTOS2: RAM console@0x96bfff000
- RTOS3: RAM console@0x97bfff000
If run hello_world_ca53_RTOS0_UART4.bin, it will print all the RAM Console's address except RTOS0.

Refer to Section 3.5.3 for how to boot RTOS on Cortex-A Core, and refer to Section 4.3.1.1.3 for how to dump RAM Console log.

Take Usecase #2 as example, follow the following steps to boot multiple OS on the platform:

- **Boot Cortex-M Core's RTOS**

  ```markdown
  u-boot=> ext4load mmc 1:2 0x48000000 /examples/mcux-sdk/freertos-hello/release/freertos_hello.bin;
  u-boot=> cp.b 0x48000000 0x7e0000 20000;
  u-boot=> bootaux 0x7e0000
  ```

  Then the following log will be displayed from UART4:
  
  Hello world.

- **Boot the first Cortex-A Core's RTOS with RAM_Console on Core2**

  ```markdown
  u-boot=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca53_RTOS0_RAM_CONSOLE.bin
  u-boot=> dcache flush; icache flush; cpu 2 release 0x93C00000
  ```

  Or boot the first Cortex-A Core's RTOS with RAM_Console on UART4 if it is not used by Cortex-M Core RTOS:

  ```markdown
  u-boot=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca53_RTOS0_UART4.bin
  u-boot=> dcache flush; icache flush; cpu 2 release 0x93C00000
  ```

  Then check the RAM Console's log as follows:

  ```markdown
  u-boot=> dcache flush; md 0x94bff000
  94bff000: 5f4d4152 534e4f43 00454c4f 00000000 RAM_CONSOLE.....
  94bff010: 94bff040 00000000 00000fc0 00000000 @............
  94bff020: 00000000 00000000 00000000 00000000 ...............
  94bff030: 00000000 00000000 00000000 00000000 ...............
  94bff040: 6f430a0d 78657472 3335412d 5452203a ..Cortex-A53: RT
  94bff050: 3a30353f 6c654f0d 32206e6f20 77206f6c65676445 OS0: Hello world
  94bff060: 7420636f6e6365520a203a3263204d 44757368696374..RTOS2: RAM console
  94bff070: 20206e6f203030306669740a0d30 ....6047700..ti
  94bff080: 6f430a0d 78657472 3335412d 5452203a ..Cortex-A53: RT
  94bff090: 3a30353f 6c654f0d 32206e6f20 77206f6c65676445 OS0: Hello world
  94bff0a0: 7420636f6e6365520a203a3263204d 44757368696374..RTOS2: RAM console
  94bff0b0: 3a30353f 6c654f0d 32206e6f20 77206f6c65676445 OS0: Hello world
  94bff0c0: 7420636f6e6365520a203a3263204d 44757368696374..RTOS2: RAM console
  94bff0d0: 3a30353f 6c654f0d 32206e6f20 77206f6c65676445 OS0: Hello world
  94bff0e0: 617420636f6e6365520a203a3263204d 44757368696374..RTOS2: RAM console
  94bff0f0: 617420636f6e6365520a203a3263204d 44757368696374..RTOS2: RAM console
  94bff100: 6f430a0d 78657472 3335412d 5452203a ..Cortex-A53: RT
  ```

- **Boot the second Cortex-A core's RTOS on Core3**

  ```markdown
  u-boot=> ext4load mmc 1:2 0x94C00000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca53_RTOS1_RAM_CONSOLE.bin
  u-boot=> dcache flush; icache flush; cpu 3 release 0x94C00000
  ```

  Then check the log of RAM Console as follows:

  ```markdown
  u-boot=> dcache flush; md 0x95bff000
  95bff000: 5f4d4152 534e4f43 00454c4f 00000000 RAM_CONSOLE.....
  95bff010: 95bff040 00000000 00000fc0 00000000 @............
  95bff020: 00000000 00000000 00000000 00000000 ...............
  95bff030: 00000000 00000000 00000000 00000000 ...............
  95bff040: 6f430a0d 78657472 3335412d 5452203a ..Cortex-A53: RT
  ```
Boot SMP Linux from Cortex-A Core0 and Core1:

```bash
u-boot=> setenv fdtfile imx8mm-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk-ignore-unused
u-boot=> boot
```

After Linux kernel boots up, can still check the RTOS log of Cortex-A Core:

```bash
root@imx8mm-lpddr4-evk:~# ram_console_dump 0x95bff000
RAM Console@0x95bff000:
```

Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
RTOS1: RAM console@0x95bff000
RTOS2: RAM console@0x96bff000
RTOS3: RAM console@0x97bff000
```

```bash
root@imx8mm-lpddr4-evk:~# ram_console_dump 0x95bff000
RAM Console@0x95bff000:
```

Cortex-A53: RTOS1: Hello world! Real-time Edge on MIMX8MM-EVK
4.3.2.3.3 Running use cases on i.MX 93 EVK

The following use cases can run on i.MX 93 EVK:

<table>
<thead>
<tr>
<th>ID</th>
<th>M33</th>
<th>A55</th>
<th>A55</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RTOS</td>
<td>SMP Linux</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>RTOS</td>
<td>Linux</td>
<td>RTOS</td>
</tr>
<tr>
<td>2</td>
<td>RTOS</td>
<td>RTOS</td>
<td>RTOS</td>
</tr>
</tbody>
</table>

By default, RTOS of the Cortex-M core uses UART2 as debug console. Refer to Section 3.7.1 for how to boot Cortex-M Core's RTOS image.

RTOS of the Cortex-A core can run with UART2 console, RAM console, or UART1 console. The following RTOS images are provided:

```bash
└── mcimx93evk
    ├── hello_world_ca55_RTOS0_RAM_CONSOLE.bin
    │    ├── hello_world_ca55_RTOS0_UART2.bin
    │    └── hello_world_ca55_RTOS1_UART1.bin
    └── hello_world_cm33_UART2.bin
```

RAM console address for each RTOS is listed below:

| RTOS0: | RAM console@0xd0fff000
| RTOS1: | RAM console@0xd1fff000

If you run the `hello_world_ca53_RTOS0_UART4.bin`, it displays the addresses of all the RAM Consoles except RTOS0.

Refer to Section 3.5.3 for how to boot RTOS on Cortex-A Core, and refer to Section 4.3.1.1.3 for how to dump RAM Console log.

Taking Use Case #1 as an example, below are the steps to boot multiple operating systems on the platform:

- **Boot Cortex-M Core's RTOS**
  ```bash
  u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/hello-world/cm/release/hello_world_cm33_UART2.bin;
  u-boot=> cp.b 0xd0000000 0x201e0000 20000;
  u-boot=> bootaux 0x1ffe0000
  ```
  Then you will find the following log from UART4:
  ```
  Cortex-M33: RTOS0: Hello world! Real-time Edge on MIMX93-EVK
  tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac tic tac
  ```

- **Boot Cortex-A Core's RTOS on Core2**
  ```bash
  u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/hello-world-ca/ddr_release/hello_world_ca55_RTOS0_RAM_CONSOLE.bin
  u-boot=> dcache flush; icache flush; cpu 1 release 0xd0000000
  ```
  Then check the RAM Console's log as follows:
  ```bash
  u-boot=> dcache flush; md 0xd0fff000
d0fff000: 5f4d4152 534e4f43 00454f00 00000000 RAM_CONSOLE.....
d0fff010: d0fff040 00000000 0000fc00 00000000 @..............
  ```
**4.3.3 lwIP Networking Stack**

### 4.3.3.1 Overview

lwIP is a small independent implementation of the TCP/IP protocol suite, it is freely available under a BSD license.

### 4.3.3.2 Running lwIP Application on i.MX 8M Plus LPDDR4 EVK

Heterogeneous Multicore Framework provides one lwIP application in the repository heterogeneous-multicore.

The Heterogeneous-multicore LWIP Ping application runs on Cortex-A Core. It will initialize LWIP networking stack, and configure ENET port on i.MX 8M Plus LPDDR4 EVK with default IP address "192.168.0.100" and default gateway address "192.168.0.254", then ping the gateway.

1. **Hardware Setup**
   - Connect i.MX 8M Plus LPDDR4 EVK's ENET port to another board's ethernet port with ethernet cable, and configure another board's ethernet interface with IP address "192.168.0.254".

2. **Run the Application**
Boot up i.MX 8M Plus LPDDR4 EVK board, and boot lwIP application from U-Boot command line:

```
  u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/lwip-ping-ca/ddr_release/lwip_ping_ca53.bin
  u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
```

Then, the following log is displayed on UART4 Console:

```
Initializing PHY...

********************************************************************************
PING example
********************************************************************************
IPv4 Address : 192.168.0.100
IPv4 Subnet mask : 255.255.255.0
IPv4 Gateway : 192.168.0.254
********************************************************************************
ping: send
192.168.0.254

ping: recv
192.168.0.254
0 ms
```

4.4 RPMSG data communication

4.4.1 Overview

RPMsg (Remote Processor Messaging) protocol defines a standardized binary interface and is used for inter-core communication between Heterogeneous AMP on i.MX MPU platforms.

Currently Real-time Edge supports the following Heterogeneous AMP:

• Linux on Cortex-A core(s)
• RTOS on Cortex-M core
• RTOS on Cortex-A core(s)

Between these OS running different processes, Real-time Edge supports inter-core communication between Cortex-M core and Cortex-A core. It also supports RPMSG between heterogeneous AMP on different Cortex-A cores.

4.4.2 RPMSG performance evaluation

This RPMSG performance application provides a method to evaluate the RPMSG channel's benchmarks between Linux as RPMSG master and FreeRTOS as RPMSG remote.

4.4.2.1 Running RPMsg performance application on Cortex-M core

Follow the steps below to run the application on i.MX 8M Plus LPDDR4 EVK:
1. **Boot the FreeRTOS on Cortex-M core**

   ```
   u-boot=> load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/rpmsg-perf-cm/release/rpmsg_perf_cm7.bin
   u-boot=> cp.b 48000000 7e0000 20000
   u-boot=> bootaux 7e0000
   ```

   The FreeRTOS boots up and waits for RPMsg link up, the logs on UART4 console as the following:

   ```
   Cortex-M7: RPMsg performance with linux:
   INFO: rpmsg_init : RPMSG remote init ...
   INFO: rpmsg_remote_init : waiting for link establish ...
   ```

2. **Boot the Linux with the required DTB.**

   ```
   u-boot=> setenv fdtfile imx8mp-evk-rpmsg.dtb
   u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
   u-boot=> run bsp_bootcmd
   ```

   The Linux boot up and kick the FreeRTOS to establish the RPMsg link. 
   Then the Remote peer FreeRTOS console displays the RPMsg link status as the following:

   ```
   Cortex-M7: RPMsg performance with linux:
   INFO: rpmsg_init : RPMSG remote init ...
   INFO: rpmsg_remote_init : waiting for link establish ...
   INFO: rpmsg_remote_init : RPMSG link up
   ```

3. **Install the Linux rpmsg_perf driver module using the commands below:**

   ```
   The driver creates a char device rpmsg-perf30 as shown below, which is used by the user space tool rpmsg_perf to test the RPMsg benchmarks:
   ```

   ```
   root@imx8mp-lpddr4-evk:~# modprobe rpmsg_perf
   root@imx8mp-lpddr4-evk:~# ls /dev/rpmsg-perf<x>
   /dev/rpmsg-perf30
   ```

4. **Use the rpmsg_perf tool as listed below:**

   ```
   The example runs rpmsg_perf /dev/rpmsg-perf0 true true 64 60 in the usage:
   ```

   ```
   root@imx8mp-lpddr4-evk:~# rpmsg_perf /dev/rpmsg-perf0 true true 64 60
   [ 1643.799911] rpmsg_perf: packet size: 64, sent packets: 4075370, time: 60 s, rate: 67 kpps
   ```

   It means that Linux sends 64 Bytes packets to the FreeRTOS side during the given period 60s. The FreeRTOS RPMsg remote receives these packets using no_copy version APIs, and the performance is about 67 kpps.

4.4.3 RPMSG between Cortex-A Core and Cortex-M Core

*Figure 27* shows RPMSG communication between RTOS running on Cortex-M core and Linux running Cortex-A core.
On i.MX MPU platforms, RPMSG builds virtual queue by leveraging Vring of VirtIO in shared memory of DDR. MU (Message Unit) is a hardware component in MPU platform that provides inter-core interrupt between Cortex-M core and Cortex-A core, so RPMG uses MU as a mailbox notification mechanism.

In Linux, RPMSG communication is based on VirtIO driver and MU mailbox drivers. The RPMsg-Lite is an open-source component developed by NXP Semiconductors. It is a lightweight implementation of the RPMSG protocol. RPMsg-Lite is used on RTOS. It includes VirtIO driver, mailbox driver, and RPMSG driver. RPMsg-Lite is also enabled on RTOS running on Cortex-A cores.

Details about RPMsg-Lite can be found in the [RPMsg-Lite User's Guide](https://developer.nxp.com).

### 4.4.3.1 RPMSG with enhanced 8MB Vring buffer

### 4.4.3.2 RPMSG merits

The RPMSG bus implements 2 virtqueues for transmitting and receiving respectively, and currently each virtqueue can support up to 256 RPMSG buffers with hardcode size 512B.

This feature increases the total number of RPMSG buffer to 8192 (4096 per direction) and extends the buffer size to 1024B.

### 4.4.3.3 Building and running the RPMSG demo (Cortex-A and Cortex-M core)

To build and run the demo for RPMSG between Cortex-A and Cortex-M cores, follow the steps listed below:

1. Enable RPMSG 8M buffer support in Real-time Edge software using the below commands:

   ```bash
   $ cd yocto-real-time-edge/sources/meta-real-time-edge
   # Open file "conf/distro/include/real-time-edge-base.inc" add "rpmsg_8m_buf"
   to "DISTRO_FEATURES" like this:
   DISTRO_FEATURES:append:mx8mm-nxp-bsp = " rpmsg_8m_buf"
   ```

2. Build the image using the commands below:

   ```bash
   $ cd yocto-real-time-edge
   $ DISTRO=nxp-real-time-edge MACHINE=imx8mm-lpddr4-evk source real-time-edge-setup-env.sh -b build-imx8mm-real-time-edge
   ```
REALTIMEEDGEUG

NXP Semiconductors

Real-time Edge Software User Guide

$ bitbake nxp-image-real-time-edge
3. Program the full SD card image. For this, use SD card with capacity of at least 4 GB.
$ bzip2 -d -c nxp-image-real-time-edge-imx8mm-lpddr4-evk.wic.bz2 | pv | sudo
dd of=/dev/sdx bs=1M && sync
# Note: find the right SD Card device name in your host machine and replace
the “sdx”.
4. Start up M-core firmware under U-Boot:
a. If you choose to run the binary in DRAM:
=> ext4load mmc 1:2 0x80000000 /examples/mcux-sdk/rpmsg-lite-str-echortos-8m-cm/ddr_release/rpmsg_lite_str_echo_rtos.bin
=> dcache flush
=> bootaux 0x80000000
b. If you choose to run the binary in TCM:
=> ext4load mmc 1:2 0x48000000 /examples/mcux-sdk/rpmsg-lite-str-echortos-8m-cm/release/rpmsg_lite_str_echo_rtos.bin
=> cp.b 0x48000000 0x7e0000 0x20000
=> bootaux 0x7e0000
5. Boot up Linux with RPMSG DTB:
=> setenv fdtfile imx8mm-evk-rpmsg-8m-buf.dtb
=> run bsp_bootcmd
6. After Linux boots up, load imx_rpmsg_tty.ko
root@imx8mm-lpddr4-evk:~# modprobe imx_rpmsg_tty
Linux imx_rpmsg_tty driver sends a “hello world!” message when probed, and it is displayed on the
FreeRTOS console.
7. Test string transmitting through device “ttyRPMSG30” from Linux prompt, the FreeRTOS console displays
the received string. For example execute the following command:
root@imx8mm-lpddr4-evk:~# echo “any-string” > /dev/ttyRPMSG30
8. In this demo, the single RPMSG buffer size is 1024B and the RPMSG header overhead is 16B, so the
transmitting string will be split into up to 1008B fragments. Use the following commands to generate a file
larger than 1KB to verify:
root@imx8mm-lpddr4-evk:~# for i in {1..300}; do echo -n `seq -s "" 0 1 9` >>
num.txt; done
root@imx8mm-lpddr4-evk:~# echo `cat num.txt` > /dev/ttyRPMSG30
The log displays the message shown below:
RPMSG String Echo FreeRTOS RTOS API Demo...
Nameservice sent, ready for incoming messages...
Get Message From Master Side : "hello world!" [len : 12]
Get Message From Master Side : "012345678901234567890123456789012345678901234567
89012345678901234567890123456789012345678901234567890123456789012345678901234567
...
89012345678901234567890123456789012345678901234567890123456789012345678901234567
" [len : 1008]
Get Message From Master Side : "8" [len : 1]
Get Message From Master Side : "901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123456
...
78901234567890123456789012345678901234567890123456789012345678901234567890123456
" [len : 1008]
Get Message From Master Side : "7" [len : 1]
Get Message From Master Side : "890123456789012345678901234567" [len : 30]
REALTIMEEDGEUG

User guide

All information provided in this document is subject to legal disclaimers.

Rev. 2.7 — 18 December 2023

© 2023 NXP B.V. All rights reserved.

101 / 400


4.4.4 RPMSG between Cortex-A Core and Cortex-A Core

Heterogeneous Multicore Framework provides RPMSG communication between Cortex-A Core and Cortex-A Core:

- RPMSG between Linux on Cortex-A Core and RTOS on Cortex-A Core
- RPMSG between RTOS on Cortex-A Core and RTOS on Cortex-A Core

There is no MU hardware mailbox that can be used between different Cortex-A cores. Therefore, a Generic Software mailbox is created for message notification between Cortex-A cores. The Generic Software mailbox uses shared memory to simulate MMIO registers that are used by the mailbox driver. Two unused SPI interrupts in GIC are used as notification interrupts between Cortex-A cores. RPMg-Lite is also enabled on RTOS of Cortex-A cores.

4.4.4.1 RPMSG between Cortex-A Linux and Cortex-A RTOS

The following diagram illustrates the software setup for RPMSG between Linux on Cortex-A Core and RTOS on Cortex-A Core.

![Diagram showing RPMSG between Linux and RTOS on different Cortex-A cores](image)

Figure 28. RPMSG between Linux and RTOS on different Cortex-A cores

4.4.4.1.1 Building the RPMSG demo on i.MX 8M Mini

Please refer to RTEDGEYOCTOUG to set up Yocto environment and build the `nxp-image-real-time-edge`. All demo applications are located in the `/examples` directory of the rootfs.
Use the following command to compile the demo separately:

```
bitbake rpmsg-str-echo-ca
```

The demo is located in the `tmp/deploy/images/imx8mm-lpddr4-evk/examples/` directory.

### 4.4.4.1.2 Running the RPMSG demo

1. Open 2 terminal emulators to connect UART2 and UART4, respectively with the following setup:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit

2. Start up FreeRTOS on the selected A-core under U-Boot:
   ```
   => ext4load mmc 1:2 93c00000 /examples/heterogeneous-multicore/rpmsg-str-echo-ca/ddr_release/
      rpmsg_str_echo_ca53_RTOS0_UART4.bin
   => dcache flush; icache flush;
   => cpu 3 release 93c00000
   ```

3. Boot up Linux with RPMSG DTB:

   ```
   => setenv fdtfile imx8mm-evk-rpmsg-ca53.dtb
   => run bsp_bootcmd
   ```

4. After Linux boots up, load `imx_rpmsg_tty.ko`, it will create 3 ttyRPMSG under `/dev`
   ```
   root@imx8mm-lpddr4-evk:~# modprobe imx_rpmsg_tty
   root@imx8mm-lpddr4-evk:~# ls /dev/ttyRPMSG*
   /dev/ttyRPMSG3 /dev/ttyRPMSG4 /dev/ttyRPMSG5
   ```

5. Use minicom to open a console connecting one of the device `ttyRPMSG` on the Linux prompt as shown below:
   ```
   root@imx8mm-lpddr4-evk:~# minicom -D /dev/ttyRPMSG3
   ```

   Observe that the input string should then be echoed back on the console.

### 4.4.4.2 RPMSG between Cortex-A RTOS and Cortex-A RTOS

The following diagram illustrates the software setup for RPMSG between different RTOS on different Cortex-A Core.
4.4.4.2.1 Running the RPMsg pingpong application

This RPMsg pingpong application demonstrates the RPMsg communication between two FreeRTOSes running respectively on 2 Cortex-A core islands.

Follow the steps below to run the demo on i.MX 8M Plus LPDDR4 EVK:

1. First, boot the remote peer FreeRTOS on core2:

   u-boot=> load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/rpmsg-pingpong-remote-ca/ddr_release/rpmsg_pingpong_remote_ca53_UART4.bin
   u-boot=> dcache flush; icache flush;
   u-boot=> cpu 2 release 0xC0000000

   The Remote peer boots up and waits for RPMsg link up, the logs on UART4 console are displayed as the following:

   RPMsg Ping-Pong FreeRTOS Demo: remote: running at 0xc0000000
   master RAM_CONSOLE at 0xc1fff000
   INFO: rpmsg_init : RPMSG remote init ...
   INFO: rpmsg_remote_init : waiting for link establish ...

2. As a second step, boot the Master peer FreeRTOS on core3, which uses the RAM console as output.

   u-boot=> load mmc 1:2 0xC1000000 /examples/heterogeneous-multicore/rpmsg-pingpong-master-ca/ddr_release/rpmsg_pingpong_master_ca53_RAM_CONSOLE.bin
   u-boot=> dcache flush; icache flush;
   u-boot=> cpu 3 release 0xC1000000

   The Master boots up and kicks the Remote peer to start the pingpong tests.
   Then the remote peer FreeRTOS console displays the test result as the following:

   RPMsg Ping-Pong FreeRTOS Demo: remote: running at 0xc0000000
INFO: rpmsg_init            : RPMSG remote init ...  
INFO: rpmsg_remote_init     : waiting for link establish ...  
INFO: rpmsg_remote_init     : RPMSG link up  
Waiting for ping ...  
Received ping (0) ... sending pong (1)  
Received ping (2) ... sending pong (3)  
Received ping (4) ... sending pong (5)  
Received ping (6) ... sending pong (7)  
Received ping (8) ... sending pong (9)  
Received ping (10) ... sending pong (11)  
Received ping (12) ... sending pong (13)  
Received ping (14) ... sending pong (15)  
Received ping (16) ... sending pong (17)  
Received ping (18) ... sending pong (19)  
Received ping (20) ... sending pong (21)  
Received ping (22) ... sending pong (23)  
Received ping (24) ... sending pong (25)  
Received ping (26) ... sending pong (27)  
Received ping (28) ... sending pong (29)  
Received ping (30) ... sending pong (31)  
Received ping (32) ... sending pong (33)  
Received ping (34) ... sending pong (35)  
Received ping (36) ... sending pong (37)  
Received ping (38) ... sending pong (39)  
Received ping (40) ... sending pong (41)  
Received ping (42) ... sending pong (43)  
Received ping (44) ... sending pong (45)  
Received ping (46) ... sending pong (47)  
Received ping (48) ... sending pong (49)  
Received ping (50) ... sending pong (51)  
Received ping (52) ... sending pong (53)  
Received ping (54) ... sending pong (55)  
Received ping (56) ... sending pong (57)  
Received ping (58) ... sending pong (59)  
Received ping (60) ... sending pong (61)  
Received ping (62) ... sending pong (63)  
Received ping (64) ... sending pong (65)  
Received ping (66) ... sending pong (67)  
Received ping (68) ... sending pong (69)  
Received ping (70) ... sending pong (71)  
Received ping (72) ... sending pong (73)  
Received ping (74) ... sending pong (75)  
Received ping (76) ... sending pong (77)  
Received ping (78) ... sending pong (79)  
Received ping (80) ... sending pong (81)  
Received ping (82) ... sending pong (83)  
Received ping (84) ... sending pong (85)  
Received ping (86) ... sending pong (87)  
Received ping (88) ... sending pong (89)  
Received ping (90) ... sending pong (91)  
Received ping (92) ... sending pong (93)  
Received ping (94) ... sending pong (95)  
Received ping (96) ... sending pong (97)  
Received ping (98) ... sending pong (99)  
Received ping (100) ... sending pong (101)  

RPMsg demo ends
The Master peer also displays the result on the RAM console:

```
u-boot=> dcache flush; md c1fff000 260
c1fff000: 734d5052 69502067 502d676e 20676e6f  RPMsg Ping-Pong
    c1fff040: 2034f46 73676772 6e6970 32027469 3a202020 534d5052 : RPMS
    c1fff060: 616d616e6a 207468656c6c6564657273 696f6e6974696e6720696e697369696f20686f76657273697a6572736f6c6520736f6363657373 697320686f76657273697a6572736f6c65 73657276656e636520736f6363657373 697320686f76657273697a6572736f6c65 207368617420736861742073686174 73657276656e636520736f6363657373 697320686f76657273697a6572736f6c65 207368617420736861742073686174 73657276656e636520736f6363657373 697320686f76657273697a6572736f6c65 207368617420736861742073686174 73657276656e636520736f6363657373 697320686f76657273697a6572736f6c65 207368617420736861742073686174
```
4.4.5 Complex RPMSG on MPU

4.4.5.1 Overview

In order to demonstrate typical RPMSG usecase on MPU platform, Heterogeneous Multicore Framework provides a complex RPMSG str-echo application in the repository: heterogeneous-multicore.

The Figure 30 shows the application setup:

```c
clfff750: 6572202e 76696563 70206465 206766ef . received pong
clfff760: 29373728 65530a0d 6e69646e 69702067 (77)...Sending ping (78) received pong
clfff770: 2820676e 20203837 202e2e2e 65636572 ping (80) ... received pong
clfff780: 6e677669 6e677669 76696563 70206465 (81)...Sending ping (82) received pong
clfff790: 6e6e6553 206766ef 676e6970 730382820 Sending ping (83)...Sending ping (84) received pong
clfff7a0: 206766ef 23911382 5565a0d 6e66646e ong (85)...Sending ping (86) received pong
clfff7b0: 6e6e6553 206766ef 676e6970 730382820 Sending ping (87)...Sending ping (88) ... received pong (89)
clfff7c0: 6e6e6553 206766ef 676e6970 20820676e (90)...Sending ping (91)...Sending ping (92) ... received pong
clfff7d0: 76696563 70206465 6572202e 76696563 ping (93)...Sending ping (94) ... received pong (95) 6ed pong (96)
clfff7e0: 0d666964 69702067 20820676e 76696563 (97)...Sending ping (98)...Sending ping (99) ... received pong
clfff7f0: 76696563 70206465 6572202e 76696563 Ping (100)...Received pong (101)...
clfff80: 6e66646e 69702067 730382820 5020320d 206766ef 676e6970 730382820 Sending ping (102)...Sending ping (103) ._received pong (104)
clfff81: 206766ef 23911382 5565a0d 6e66646e ong (105)...Sending ping (106) ... received pong (107)
clfff82: 6e6e6553 206766ef 676e6970 730382820 Sending ping (108)...Sending ping (109) ... received pong (110)
clfff83: 6e66646e 69702067 730382820 5020320d 206766ef 676e6970 730382820 Sending ping (111)...Sending ping (112) .received pong (113)
clfff84: 206766ef 23911382 5565a0d 6e66646e ong (114)...Sending ping (115) ... received pong (116)
clfff85: 6e6e6553 206766ef 676e6970 730382820 Sending ping (117)...Sending ping (118) ... received pong (119)
clfff86: 6e6e6553 206766ef 676e6970 730382820 Sending ping (120)...Sending ping (121) ... received pong (122)
clfff87: 6e66646e 69702067 730382820 5020320d 206766ef 676e6970 730382820 Sending ping (123)...Sending ping (124) .received pong (125)
clfff88: 206766ef 23911382 5565a0d 6e66646e ong (126)...Sending ping (127) ... received pong (128)
clfff89: 6e6e6553 206766ef 676e6970 730382820 Sending ping (129)...Sending ping (130) ... received pong (131)
clfff8a: 6e66646e 69702067 730382820 5020320d 206766ef 676e6970 730382820 Sending ping (132)...Sending ping (133) .received pong (134)
clfff8b: 206766ef 23911382 5565a0d 6e66646e ong (135)...Sending ping (136) ... received pong (137)
clfff8c: 6e6e6553 206766ef 676e6970 730382820 Sending ping (138)...Sending ping (139) ... received pong (140)
clfff8d: 6e66646e 69702067 730382820 5020320d 206766ef 676e6970 730382820 Sending ping (141)...Sending ping (142) .received pong (143)
clfff8e: 206766ef 23911382 5565a0d 6e66646e ong (144)...Sending ping (145) ... received pong (146)
clfff8f: 6e6e6553 206766ef 676e6970 730382820 Sending ping (147)...Sending ping (148) ... received pong (149)
clfff90: 6e66646e 69702067 730382820 5020320d 206766ef 676e6970 730382820 Sending ping (150)...Sending ping (151) .received pong (152)
clfff91: 206766ef 23911382 5565a0d 6e66646e ong (153)...Sending ping (154) ... received pong (155)
clfff92: 6e6e6553 206766ef 676e6970 730382820 Sending ping (156)...Sending ping (157) ... received pong (158)
clfff93: 6e66646e 69702067 730382820 5020320d 206766ef 676e6970 730382820 Sending ping (159)...Sending ping (160) .received pong (161)
clfff94: 206766ef 23911382 5565a0d 6e66646e ong (162)...Sending ping (163) ... received pong (164)
clfff95: 6e6e6553 206766ef 676e6970 730382820 Sending ping (165)...Sending ping (166) ... received pong (167)
clfff96: 6e66646e 69702067 730382820 5020320d 206766ef 676e6970 730382820 Sending ping (168)...Sending ping (169) .received pong (170)
clfff97: 000a0d73 00000000 00000000 .......... 00000000 s............
```
Figure 30. Complex RPMSG Setup on MPU

Rmmsg-str-echo application setup RPMSG communication between FreeRTOS and Linux, FreeRTOS runs RPMSG master endpoint, and Linux runs RPMSG remote endpoint. By default, it creates three endpoints on both the master side and remote side. Each endpoint is one-to-one connected with the other side. Therefore there are three RPMSG channels between the master and remote side. Application on FreeRTOS receives data from the remote side and then sends it back to the same RPMSG channel. On the Linux side, if data is sent to the master side, the same data is received or echoed back from the same channel.

4.4.5.2 Running the Complex str-echo application

This section describes the steps for running the Complex str-echo application on i.MX 8M Plus LPDDR4 EVK and i.MX 8M Mini LPDDR4 EVK boards.

4.4.5.2.1 Running the application on i.MX 8M Plus LPDDR4 EVK

The following RTOS images are provided to run the application:

```
rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE.bin
rpmsg_str_echo_ca53_RTOS0_UART4.bin
rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE.bin
rpmsg_str_echo_cm7.bin
```

There are two RTOS images provided for RTOS0 on Cortex-A53, one uses RAM console, the other uses UART4 console. The RAM Console image must be run if running RTOS on Cortex M7 core simultaneously because Cortex M7 Core images use UART4 console by default.

Use the following steps to run the whole setup on i.MX 8M Plus LPDDR4 EVK:
1. **Boot the First Cortex-A Core RTOS:**

```bash
u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/rpmsg-str-echo-ca/ddr_release/rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
```

2. **Boot the Second Cortex-A Core RTOS:**

```bash
u-boot=> ext4load mmc 1:2 0xC1000000 /examples/heterogeneous-multicore/rpmsg-str-echo-ca/ddr_release/rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE.bin
u-boot=> dcache flush; icache flush; cpu 3 release 0xC1000000
```

3. **Boot Cortex-M Core RTOS:**

```bash
u-boot=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/rpmsg-str-echo-cm/release/rpmsg_str_echo_cm7.bin
u-boot=> cp.b 0x48000000 0x7e0000 20000;
u-boot=> bootaux 0x7e0000
```

The below log is displayed for UART4 console:

```
INFO: rpmsg_init : RPMSG init ...
INFO: rpmsg_init : waiting for link establish ...
```

4. **Boot up Linux using the commands:**

```bash
u-boot=> setenv fdtfile imx8mp-evk-multicore-rpmsg.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

5. **When Linux is up, install tty driver module**

```bash
root@imx8mp-lpddr4-evk:~# modprobe imx_rpmsg_tty
[ 21.770356] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.3: new channel: 0x400 -> 0x3!
[ 21.770576] Install rpmsg tty driver!
[ 21.773539] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.4: new channel: 0x401 -> 0x4!
[ 21.773804] Install rpmsg tty driver!
[ 21.774034] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.5: new channel: 0x402 -> 0x5!
[ 21.774156] Install rpmsg tty driver!
[ 21.774275] imx_rpmsg_tty virtio1.rpmsg-virtual-tty-channel-1.-1.6: new channel: 0x400 -> 0x6!
[ 21.774360] Install rpmsg tty driver!
[ 21.774443] imx_rpmsg_tty virtio1.rpmsg-virtual-tty-channel-1.-1.7: new channel: 0x401 -> 0x7!
[ 21.774530] Install rpmsg tty driver!
[ 21.774586] imx_rpmsg_tty virtio1.rpmsg-virtual-tty-channel-1.-1.8: new channel: 0x402 -> 0x8!
[ 21.774663] Install rpmsg tty driver!
[ 21.774726] imx_rpmsg_tty virtio2.rpmsg-virtual-tty-channel-1.-1.0: new channel: 0x400 -> 0x0!
[ 21.774810] Install rpmsg tty driver!
[ 21.774880] imx_rpmsg_tty virtio2.rpmsg-virtual-tty-channel-1.-1.1: new channel: 0x401 -> 0x1!
[ 21.774960] Install rpmsg tty driver!
[ 21.775022] imx_rpmsg_tty virtio2.rpmsg-virtual-tty-channel-1.-1.2: new channel: 0x402 -> 0x2!
[ 21.775111] Install rpmsg tty driver!
```
6. Then check Cortex-A RTOS0's RAM Console Log:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump 0xC0FFF000
RAM Console@0xc0fff000:

Cortex-A53: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
RTOS1: RAM console@0xc1fff000
INFO: rmsg_init : RPMSG remote init ...
INFO: rpmsg_remote_init : waiting for link establish ...
INFO: rpmsg_remote_init : RPMSG link up
ept3: Get Message From Master Side : "hello world!" [len : 12]
ept4: Get Message From Master Side : "hello world!" [len : 12]
ept5: Get Message From Master Side : "hello world!" [len : 12]
> ............................................................
```

7. Check Cortex-A RTOS1's RAM Console Log:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump 0xC1FFF000
RAM Console@0xc1fff000:

Cortex-A53: RTOS1: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
INFO: rmsg_init : RPMSG remote init ...
INFO: rpmsg_remote_init : waiting for link establish ...
INFO: rpmsg_remote_init : RPMSG link up
ept6: Get Message From Master Side : "hello world!" [len : 12]
ept7: Get Message From Master Side : "hello world!" [len : 12]
ept8: Get Message From Master Side : "hello world!" [len : 12]
> ............................................................
```

The following log for Cortex-M Core's UART4 Log:

```
Cortex-M7: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
INFO: rmsg_init : RPMSG remote init ...
INFO: rpmsg_remote_init : waiting for link establish ...
INFO: rpmsg_remote_init : RPMSG link up
ept0: Get Message From Master Side : "hello world!" [len : 12]
ept1: Get Message From Master Side : "hello world!" [len : 12]
ept2: Get Message From Master Side : "hello world!" [len : 12]
```

It creates the following RPMSG devices:

```
root@imx8mp-lpddr4-evk:~# ls /dev/ttyRPMSG*
```

/dev/ttyRPMSG0 ~ 2 are three endpoints connected to Cortex-M Core RTOS, /dev/ttyRPMSG3 ~ 5 are three endpoints connected to Cortex-A Core RTOS0, /dev/ttyRPMSG6 ~ 8 are three endpoints connected to Cortex-A Core RTOS1.

8. Test RPMSG Communication:

Use "echo" or "minicom" to verify the RPMSG communication between the two RTOS.
For example, use "echo" and send a sample string to Cortex-M Core's endpoint:

```
root@imx8mp-lpddr4-evk:~# echo "adfad" > /dev/ttyRPMSG2
```

Then in Cortex-M Core's Console, the below string is received:

```
ept2: Get Message From Master Side : "adfad" [len : 5]
ept2: Get New Line From Master Side
```

Or use minicom to open one RPMSG endpoint: it will echo back the character inputted in minicom console:

```
root@imx8mp-lpddr4-evk:~# minicom -D /dev/ttyRPMSG6
```
9. Then input some character typed in using the minicom console. This character is sent to the RTOS endpoint from Linux. Then the application running on RTOS sends the character back to Linux, finally all character are echoed back in the minicom console. For example, if you input the characters "dadfddedddd", the below log would be displayed:

Welcome to minicom 2.8
OPTIONS: I18n
Compiled on Jan 1 2021, 17:45:55.
Port /dev/ttyRPMSG6, 08:36:41
Press CTRL-A Z for help on special keys
dadfddedddd

Known Issues:

- Do not turn data cache off in U-Boot while booting Cortex-M Core RTOS.

4.4.5.2.2 Running the application on i.MX 8M Mini LPDDR4 EVK

The following RTOS images are provided to run the application:

- rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE.bin
- rpmsg_str_echo_ca53_RTOS0_UART4.bin
- rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE.bin
- rpmsg_str_echo_cm4.bin

There are two RTOS images provided for RTOS0 on Cortex-A53, one uses RAM Console, the other uses UART4 Console, it needs to run RAM Console image if run RTOS on Cortex M4 Core simultaneously because Cortex M4 Core images uses UART4 Console by default.

Follow the following steps to run the whole setup on i.MX 8M Mini LPDDR4 EVK:

1. Boot the First Cortex-A Core RTOS

   u-boot=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/rpmsg-str-echo-ca/ddr_release/rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE.bin
   u-boot=> dcache flush; icache flush; cpu 2 release 0x93C00000

2. Boot the Second Cortex-A Core RTOS

   u-boot=> ext4load mmc 1:2 0x94C00000 /examples/heterogeneous-multicore/rpmsg-str-echo-ca/ddr_release/rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE.bin
   u-boot=> dcache flush; icache flush; cpu 3 release 0x94C00000

3. Boot Cortex-M Core RTOS

   u-boot=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/rpmsg-str-echo-cm/release/rpmsg_str_echo_cm4.bin
   u-boot=> cp.b 0x48000000 0x7e0000 20000;
   u-boot=> bootaux 0x7e0000

The log for UART4 Console will be the following log:

   Cortex-M4: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
   INFO: rpmsg_init : RPMSG init ...
   INFO: rpmsg_init : waiting for link establish ...

4. Boot Linux up

   u-boot=> setenv fdtfile imx8mm-evk-multicore-rpmsg.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot

5. When Linux is up, install tty driver module

```
root@imx8mm-lpddr4-evk:~# modprobe imx_rpmsg_tty
[  21.770356] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.3: new
  channel: 0x400 -> 0x3!
[  21.770576] Install rpmsg tty driver!
[  21.773539] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.4: new
  channel: 0x401 -> 0x4!
[  21.773804] Install rpmsg tty driver!
[  21.774034] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.5: new
  channel: 0x402 -> 0x5!
[  21.774156] Install rpmsg tty driver!
  channel: 0x400 -> 0x6!
[  21.774360] Install rpmsg tty driver!
  channel: 0x401 -> 0x7!
[  21.774530] Install rpmsg tty driver!
  channel: 0x402 -> 0x8!
[  21.774663] Install rpmsg tty driver!
[  21.774726] imx_rpmsg_tty virtio2.rpmsg-virtual-tty-channel-1.-1.0: new
  channel: 0x400 -> 0x0!
[  21.774810] Install rpmsg tty driver!
  channel: 0x401 -> 0x1!
[  21.774960] Install rpmsg tty driver!
  channel: 0x402 -> 0x2!
[  21.775111] Install rpmsg tty driver!
```

6. Then check Cortex-A RTOS0's RAM console log:

```
root@imx8mm-lpddr4-evk:~# ram_console_dump 0x94BFF000
RAM Console@0x94bff000:
Cortex-A53: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
INFO: rpmsg_init             : RPMSG remote init ... 
INFO: rpmsg_remote_init      : waiting for link establish ... 
INFO: rpmsg_remote_init      : RPMSG link up 
  ept3: Get Message From Master Side : "hello world!" [len : 12]
  ept4: Get Message From Master Side : "hello world!" [len : 12]
  ept5: Get Message From Master Side : "hello world!" [len : 12]
> ............................................................
```

7. And check Cortex-A RTOS1's RAM console Log:

```
root@imx8mm-lpddr4-evk:~# ram_console_dump 0x95BFF000
RAM Console@0x95bff000:
Cortex-A53: RTOS1: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
INFO: rpmsg_init             : RPMSG remote init ... 
INFO: rpmsg_remote_init      : waiting for link establish ... 
INFO: rpmsg_remote_init      : RPMSG link up 
  ept6: Get Message From Master Side : "hello world!" [len : 12]
  ept7: Get Message From Master Side : "hello world!" [len : 12]
  ept8: Get Message From Master Side : "hello world!" [len : 12]
```
The following is the log displayed for Cortex-M Core's UART4 console:

Cortex-M4: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
INFO: rpmsg_init : RPMSG remote init ...
INFO: rpmsg_remote_init : waiting for link establish ...
INFO: rpmsg_remote_init : RPMSG link up
ept0: Get Message From Master Side : "hello world!" [len : 12]
ept1: Get Message From Master Side : "hello world!" [len : 12]
ept2: Get Message From Master Side : "hello world!" [len : 12]

The following RPMSG devices are created:

```
root@imx8mm-lpddr4-evk:~# ls /dev/ttyRPMSG*
/dev/ttyRPMSG0  /dev/ttyRPMSG1  /dev/ttyRPMSG2  /dev/ttyRPMSG3  /dev/
ttyRPMSG4  /dev/ttyRPMSG5  /dev/ttyRPMSG6  /dev/ttyRPMSG7  /dev/ttyRPMSG8
```

/dev/ttyRPMSG0 ~ 2 are three endpoints connected to Cortex-M Core RTOS, /dev/ttyRPMSG3 ~ 5 are three endpoints connected to Cortex-A Core RTOS0, /dev/ttyRPMSG6 ~ 8 are three endpoints connected to Cortex-A Core RTOS1.

8. Test RPMSG Communication

Use "echo" or "minicom" to verify the RPMSG communication between the two real time operating systems. For example, use "echo" send some string to Cortex-M Core's endpoint:

```
root@imx8mm-lpddr4-evk:~# echo "adfad" > /dev/ttyRPMSG2
```

Then in Cortex-M Core's Console will find the string is received:

```
ept2: Get Message From Master Side : "adfad" [len : 5]
ept2: Get New Line From Master Side
```

Or use minicom to open one RPMSG endpoint. It then echoes back the character typed in the minicom console:

```
root@imx8mm-lpddr4-evk:~# minicom -D /dev/ttyRPMSG6
```

9. Then input some character typed in using the minicom console. These characters are sent to the RTOS endpoint from Linux. Then the application running on RTOS sends the character back to Linux. Finally all characters are echoed back to the minicom console. For example, if you input the characters "dadfddededddd", following would be the log displayed:

```
Welcome to minicom 2.8
OPTIONS: I18n
Compiled on Jan 1 2021, 17:45:55.
Port /dev/ttyRPMSG6, 08:36:41
Press CTRL-A Z for help on special keys
dadfddededddd
```

Known Issues:

- Do not turn data cache off in U-Boot while booting Cortex-M Core RTOS.
4.5 RPMSG based resource sharing

4.5.1 Overview

On NXP MPU platforms, in general, RTOS runs on Cortex-M Core(s) and Linux runs on Cortex-A Core(s). In some use cases, considering power management and real-time performance, Cortex-M Core owns and controls physical resources or peripherals, but needs to share these physical resources or peripherals with Cortex-A Core(s). The rpmsg_lite_uart_sharing_rtos is a FreeRTOS example to share physical UART owned by Cortex-M Core with Cortex-A Core.

4.5.2 Software architecture and design

This chapter describes different software architectures based on different technologies.

4.5.3 Resource sharing based on SRTM

This example uses the Simplified Real-Time Messaging (SRTM) protocol to communicate between Cortex-A and Cortex-M Cores. SRTM is used for communication among SoCs/processors in the same SoC. The figure below shows the software architecture for resource sharing based on SRTM.

![Resource Sharing Software Architecture](image)

SRTM runs on Cortex-M Core which owns the hardware resources. To share these, it provides an application protocol based on RPMSG.

Virtual Device Drivers run on the resource user. The drivers provide standard device service on Cortex-A, which needs to use the hardware resources shared by SRTM.

4.5.3.1 UART sharing design details

The UART sharing example is designed with the following features:
RTOS on Cortex-M Core owns and fully controls the physical UART ports.
SRTM service runs on RTOS and provides physical device sharing service to Linux.
Virtual UART driver on Linux provides standard UART device service to applications.
Multiple virtual UART ports are provided in Linux.
Each virtual UART port in Linux can map to a dedicated physical UART on FreeRTOS.
Multiple virtual UART ports can be mapped to the same physical UART.

Supported Platforms: i.MX 8M Mini LPDDR4 EVK, i.MX 93 EVK

It includes the following software components:

- Physical UART driver on FreeRTOS
- SRTM UART sharing service on FreeRTOS
- rpmsg_lite_uart_sharing_rtos application on FreeRTOS
- Virtual UART driver in Linux

The following figure illustrates the software architecture of a UART sharing design.

Figure 32. UART sharing software architecture

In order to support multiple virtual UART on a single physical UART, a multiple virtual UART protocol is used.
The example described in this document follows the packet format described in the below table.
Table 30. Packet Format

<table>
<thead>
<tr>
<th>Fields</th>
<th>Start Flags (4 bytes)</th>
<th>Address (1 byte)</th>
<th>Payload Size (1 byte)</th>
<th>Payload (n bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>24 55 54 2C</td>
<td>x</td>
<td>n</td>
<td>xxxxxx...</td>
</tr>
<tr>
<td>ASCII</td>
<td>$ U T ,</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The packet header includes fields that indicate start flags, address, and payload size. It is 6 bytes by default.

“Start flags” field is used to figure out the start of data packets, user can configure start flags with specified characters and size. The default start flags are 4 bytes: "$UT,“.

The “Address” field is reused by receive from transmit directions. For receive direction (blue colored path in Figure 32), it is the destination address or ID of target virtual device. For transmit direction (orange colored path in Figure 32), it is the source address or ID from which virtual device is transmitted.

“Payload Size” is the size of payload data, it is one byte, so the maximum payload size is 255 bytes. "Payload" is the actual data exchanged within protocol and it follows the packet header.

4.5.4 Source code files and configuration

1. **Source code files:**
   The source files for different software components are listed in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Software component</th>
<th>Source Files/Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS application:</td>
<td>mcux-sdk-examples</td>
<td>evkmx8mm/multicore_examples/</td>
</tr>
<tr>
<td>rpmmsg_lite_uart_sharing_</td>
<td></td>
<td>rpmsg_lite_uart_sharing_rtos/</td>
</tr>
<tr>
<td>rtos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRTM Service</td>
<td>mcu-sdk</td>
<td>components/srtm/services/</td>
</tr>
<tr>
<td></td>
<td></td>
<td>srtm_uart_service.c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>srtm_uart_service.h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>srtm_uart_adapter.c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>srtm_uart_adapter.h</td>
</tr>
<tr>
<td>Virtual UART driver</td>
<td>real-time-edge-linux</td>
<td>drivers/tty/rpmsg tty.c</td>
</tr>
</tbody>
</table>

2. **Linux Virtual UART driver**
   By default, Real-time Edge kernel builds the virtual UART driver as module (rpmsg tty.ko) by enabling the configure item: CONFIG_RPMSG_TTY=m.

3. **Virtual UART and physical UART mapping**
   The UART Sharing Service supports three modes of mapping between virtual UART and physical UART:
   a. **Virtual UART to physical UART 1:1 mapping**
      • Virtual UARTs on A-core have 1:1 mapping to physical UARTs on the M-core.
      • Each physical UART connects to a different device.
      • Each virtual UART uses a dedicated RMSG endpoint.
b. Virtual UART to physical UART n:1 mapping
   - Multiple Virtual UARTs on A-core maps to a single physical UARTs on M-core.
   - Physical UART connects to a device or another board.
   - Each virtual UART uses a dedicated RPMSG Endpoint.
   - Multiple UART Header is used to establish multiple virtual UART channels on a single physical UART connect. For details about multiple UART Headers, refer to the section, "Section 4.5.3.1".

Figure 33. Virtual UART to physical UART 1:1 mapping

Figure 34. Virtual UART to physical UART n:1 mapping

c. Virtual UART to physical UART flexible mapping: This mapping mode can support virtual UART to physical UART 1:1 mapping and n:1 mapping simultaneously. The following figure shows flexible mapping between two i.MX 8M Mini boards.
Figure 35. Virtual UART to physical UART flexible mapping

The mapping between virtual UART and physical UART is configured in Linux device tree, as shown in a dts node example below:

```c
uart_rpbus_3: uart-rpbus-3 {
    compatible = "fsl,uart-rpbus";
    bus_id = <3>; /* use uart3 */
    flags=<IMX_SRTM_UART_SUPPORT_MULTI_UART_MSG_FLAG>;
    status = "okay";
};
```

This dts node is configured for virtual UART3.

**Note:**
- The “bus_id” specifies the physical UART instance ID that this virtual UART maps to. If the property of “bus_id” is not configured, the message sent from Linux to this virtual UART is displayed on M-core’s debug console directly.
- Physical UART ID is configured in the FreeRTOS application “rpmsg_lite_uart_sharing_rtos”.
- On i.MX 8M Mini LPDDR4 EVK, physical UART3 can be used, so all virtual UART ports are mapped to physical UART3 by default.
- On i.MX 93 EVK, physical LPUART5 can be used, so all virtual UART ports are mapped to physical LPUART5 by default.
- If flags is set with the value IMX_SRTM_UART_SUPPORT_MULTI_UART_MSG_FLAG, the multiple virtual UART is mapped to a single physical UART instance specified by bus_id (that implies that multiple virtual UART protocol packet headers are used).
- If flags is not set, this virtual UART is mapped 1:1 with physical UART instance specified by bus_id. By default, there are 11 virtual UARTs in the dtb file `imx8mm-evkrpmmsg.dtb` for i.MX 8M Mini LPDDR4 EVK and `imx93-11x11-evk-rpmsg.dtb` for i.MX 93 EVK.
- The virtual UART 0 to 9 are n:1 mapped to physical UART.
- The virtual UART 10 has no bus_id and displays messages sent from Linux to M-core’s debug console.
4.5.5 Building and running the demo on i.MX 8M Mini LPDDR4 EVK

4.5.5.1 Hardware setup for i.MX 8M Mini EVK

Use flying wire to connect UART3 between two i.MX 8M Mini EVK boards. UART3’s pin is provided in J1003 connector; use the following pin connection between the two boards.

Table 32. PIN connection between two i.MX 8M Mini boards

<table>
<thead>
<tr>
<th>i.MX 8M Mini Board1</th>
<th>Connection</th>
<th>i.MX 8M Mini Board2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>Function</td>
<td>Pin</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>6</td>
<td>GND</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>UART3_TXD</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>UART3_RXD</td>
<td>8</td>
</tr>
</tbody>
</table>

4.5.5.2 Building the demo images

The demo images "rpmsg_lite_uart_sharing_rtos.bin" are by default compiled with the i.MX 8M Mini LPDDR4 EVK target image compiling, and are installed into the "/examples" directory of the target rootfs.

Or the image can be built separately by using the following Yocto command:

```
DISTRO=nxp-real-time-edge MACHINE=imx8mm-lpddr4-evk bitbake rpmsg-lite-uart-sharing-rtos
```

The image can be found on directory "<image-build-dir>/tmp/deploy/images/imx8mmevk/examples/" on building host.

4.5.5.3 Running the i.MX 8M Mini EVK demo

1. Connect two i.MX 8M Mini EVK boards by following the steps in section of "Hardware Setup".
2. Connect two i.MX8M Mini EVK boards to your PC via USB cable between the USB-UART connector and the PC USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number, two debug consoles for each board, one for the Linux debug console and another for the FreeRTOS debug console.
4. Deploy Real-time Edge release root files in SD card and modify the on-board switch to boot from MicroSD card.
5. Power on the board and enter into U-Boot command line, then execute the following command:

   ```
   u-boot => setenv fdtfile imx8mm-evk-rpmsg.dtb
   To make changes permanent, execute the following commands once (after setenv above):
   u-boot => saveenv
   ```

6. Then, use the following command to download and run FreeRTOS image:

   ```
   u-boot => ext4load mmc 1:2 0x48000000 /examples/mcux-sdk/rpmsg-lite-uart-sharing-rtos/release/rpmsg_lite_uart_sharing_rtos.bin;
   u-boot => cp.b 0x48000000 0x7e0000 20000; bootaux 0x7e0000
   ```

Then, FreeRTOS debug console would display the following log:

```
####################################################################
RPMSG UART SHARING DEMO
Build Time: Mar 2 2022--09:38:19
####################################################################
```
Wait the Linux kernel boot up to create the link between M core and A core.

7. Then boot Linux kernel by executing the following command:

```
u-boot => setenv jh_clk clk_ignore_unused
u-boot => boot
```

After the Linux kernel boots up, in the FreeRTOS, an extra line of log as shown below indicates that RPMSG connection between Cortex-A core and Cortex-M core has been established:

```
Task A is working now.
```

Execute the above steps (1 to 7) on each i.MX 8M Mini EVK board.

8. After Linux boots up, enter Linux command line, use the following commands to test the demo:

a. Check device files are available:

```
root@imx8mm-lpddr4-evk:~# ls /dev/ttyRPMSG*
```

There should be 11 device files from "/dev/ttyRPMSG0" to "/dev/ttyRPMSG10" if the default dtb file "imx8mm-evk-rpmsg.dtb" is used. The "//dev/ttyRPMSG0" to "/dev/ttyRPMSG9" have n:1 mapping to physical UART3, "/dev/ttyRPMSG10" is without "bus_id" and displays the message sent from Linux to M-core's debug console.

b. Check each virtual UART from "/dev/ttyRPMSG0" to "/dev/ttyRPMSG9" is connected to peer virtual UART between two boards, for example, uses "minicom" to open and configure the same virtual UART on both boards:

```
root@imx8mm-lpddr4-evk:~# minicom -s -D /dev/ttyRPMSG6
```

Then configure the UART as shown in the following figures:

![Figure 36. Configuring RPMSG Virtual UART Step1](image)
Save the settings and back to minicom main window, then input any characters in one board's minicom window. These characters would be displayed on the other board's minicom window.

4.5.6 Building and running the demo on i.MX 93 EVK

4.5.6.1 Hardware setup for i.MX 93 EVK

Use flying wire to connect LPUART5 between two i.MX 93 EVK boards. LPUART5’s pin is provided in J1001 connector. Use the following pin connection between the two boards.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Connection</th>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>GND</td>
<td>&lt;-&gt;</td>
<td>30</td>
<td>GND</td>
</tr>
<tr>
<td>28</td>
<td>LPUART5_RX</td>
<td>&lt;-&gt;</td>
<td>27</td>
<td>LPUART5_TX</td>
</tr>
<tr>
<td>27</td>
<td>LPUART5_TX</td>
<td>&lt;-&gt;</td>
<td>28</td>
<td>LPUART5_RX</td>
</tr>
</tbody>
</table>

4.5.6.2 Building the demo images

The demo image "rpmsg_lite_uart_sharing_rtos.bin" is by default compiled with the i.MX 93 EVK target image compiling, and are installed into the "/examples" directory of the target rootfs.

Or the image can be built separately by using the following Yocto command:

```
DISTRO=nxp-real-time-edge MACHINE=imx93evk source real-time-edge-setup-env.sh -b <build_dir>
bitbake rpmsg-lite-uart-sharing-rtos-mcimx93evk
```

The image can be found on directory "<image-build-dir>/tmp/deploy/images/imx93evk/examples/" on building host.
4.5.6.3 Running the i.MX 93 demo

1. Connect two i.MX 93 EVK boards by following the steps listed in Section 4.5.6.1.
2. Connect two i.MX 93 EVK boards to your PC via USB cable between the USB-UART connector and the PC USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number, four debug consoles for each board. Use the third one for the Linux debug console and the fourth one for the FreeRTOS debug console.
4. Deploy Real-time Edge release root files in SD card and modify the on-board switch to boot from MicroSD card.
5. Power on the board and enter into U-Boot command line. Then, execute the following command:

   ```
   u-boot => setenv fdtfile imx93-11x11-evk-uart-sharing-cm33.dtb
   To make changes permanent, execute the following commands once (after setenv above):
   u-boot => saveenv
   ```

6. Then, use the following command to download and run FreeRTOS image:

   ```
   u-boot => ext4load mmc 1:2 0x80000000 /examples/rpmsg-lite-uart-sharing-rtos-mcimx93evk/release/rpmsg_lite_uart_sharing_rtos.bin
   u-boot => cp.b 0x80000000 0x201e0000 0x10000
   u-boot => bootaux 0x1ffe0000 0
   ```

   Then, FreeRTOS debug console would display the following log:

   ```
   ####################  RPMSG UART SHARING DEMO  ####################
   Build Time: Apr 5 2011 23:00:00
   Start SRTM communication
   *****************************************
   Wait for the Linux kernel boot up to create the link between M core and A core.
   *******************************************
   ```

7. And then, boot Linux kernel by executing the following command:

   ```
   u-boot => setenv jh_clk clk_ignore_unused
   u-boot => boot
   ```

8. After the Linux kernel boots up, in the FreeRTOS console, an extra line of log as shown below indicates that RPMSG connection between Cortex-A core and Cortex-M core has been established:

   ```
   Task A is working now.
   ```

   Execute the above steps (1 to 7) on each i.MX 93 EVK board.

9. After Linux boots up, enter Linux command line, use the following commands to test the demo:

   a. Check device files are available:

   ```
   root @imx93evk:~# ls /dev/ttyRPMSG*
   ```

   There should be 11 device files from "/dev/ttyRPMSG0" to "/dev/ttyRPMSG10" if the default dtb file `imx93-11x11-evk-uart-sharing-cm33.dtb` is used. The "/dev/ttyRPMSG0" to "/dev/ttyRPMSG9" have n:1 mapping to physical LPUART5, "/dev/ttyRPMSG10" is without "bus_id" and displays the message sent from Linux to M-core's debug console.

   b. Check each virtual UART from "/dev/ttyRPMSG0" to "/dev/ttyRPMSG9" is connected to peer virtual UART between two boards, for example, execute the following on the first board:

   ```
   • Use "minicom" to open and configure the same virtual UART on both boards:
   ```

   ```
   root@imx93evk:~# minicom -s -D /dev/ttyRPMSG6
   ```
Then configure the UART as shown in the following figures:

![Configure RPMSG Virtual UART Step1](image1)

Figure 38. Configure RPMSG Virtual UART Step1

![Configure RPMSG Virtual UART Step2](image2)

Figure 39. Configure RPMSG Virtual UART Step2

Save the settings and go back to minicom main window, then input any characters in one board's minicom window. Then, the characters would be displayed on the other board's minicom window.

4.5.7 Building and running the demo on i.MX 93 QSB

4.5.7.1 Hardware setup for i.MX 93 QSB

Use flying wire to connect LPUART5 between two i.MX 93 QSB boards. LPUART5's pin is provided in J1401 connector. Use the following pin connection between the two boards.

Table 34. PIN connection between two i.MX 93 QSB boards

<table>
<thead>
<tr>
<th>I.MX 93 QSB Board1</th>
<th>Connection</th>
<th>I.MX 93 QSB Board2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>Function</td>
<td>Pin</td>
</tr>
<tr>
<td>30</td>
<td>GND</td>
<td>&lt;-&gt;</td>
</tr>
<tr>
<td>28</td>
<td>LPUART5_RX</td>
<td>&lt;-&gt;</td>
</tr>
</tbody>
</table>
Table 34. PIN connection between two i.MX 93 QSB boards...continued

<table>
<thead>
<tr>
<th>i.MX 93 QSB Board1</th>
<th>Connection</th>
<th>i.MX 93 QSB Board2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>Function</td>
<td>Pin</td>
</tr>
<tr>
<td>27</td>
<td>LPUART5_TX</td>
<td>&lt;-&gt;</td>
</tr>
</tbody>
</table>

4.5.7.2 Building the demo images

The demo image "rpmsg_lite_uart_sharing_rtos.bin" is by default compiled with the i.MX 93 QSB target image compiling, and are installed into the "/examples" directory of the target rootfs.

Or the image can be built separately by using the following Yocto command:

```bash
DISTRO=nxp-real-time-edge MACHINE=imx93-9x9-lpddr4-qsb source real-time-edge-setup-env.sh -b <build_dir>
bitbake rpmsg-lite-uart-sharing-rtos-mcimx93qsb
```

The image can be found on directory "<image-build-dir>/tmp/deploy/images/imx93-9x9-lpddr4-qsb/examples/" on building host.

4.5.7.3 Running the demo on i.MX 93 QSB

1. Connect two i.MX 93 QSB boards by following the steps listed in Section 4.5.7.1.
2. Connect two i.MX 93 QSB boards to your PC via USB cable between the USB-UART connector and the PC USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number, four debug consoles for each board. Use the third one for the Linux debug console and the fourth one for the FreeRTOS debug console.
4. Deploy Real-time Edge release root files in SD card and modify the on-board switch to boot from MicroSD card.
5. Power on the board and enter into U-Boot command line. Then, execute the following command:

```bash
u-boot => setenv fdtfile imx93-9x9-qsb-uart-sharing-cm33.dtb
```

To make changes permanent, execute the following commands once (after `setenv` above):

```bash
u-boot => saveenv
```

6. Then, use the following command to download and run FreeRTOS image:

```bash
u-boot => ext4load mmc 1:2 0x80000000 /examples/rpmsg-lite-uart-sharing-rtos-mcimx93qsb/release/rpmsg_lite_uart_sharing_rtos.bin
u-boot => cp.b 0x80000000 0x201e0000 0x10000
u-boot => bootaux 0x1ffe0000 0
```

Then, FreeRTOS debug console would display the following log:

```
################################ RPSMG UART SHARING DEMO ################################
Build Time: Oct 30 2023 11:20:34
Start SRTM communication
*******************************
Wait for the Linux kernel boot up to create the link between M core and A core.
```

7. And then, boot Linux kernel by executing the following command:

```bash
u-boot => setenv jh_clk clk_ignore_unused
```
8. After the Linux kernel boots up, in the FreeRTOS console, an extra line of log as shown below indicates that RPMSG connection between Cortex-A core and Cortex-M core has been established:

| Task A is working now. |

Execute the above steps (1 to 7) on each i.MX 93 QSB board.

9. After Linux boots up, enter Linux command line using the following commands to test the demo:

a. Check the device files are available:

```
root@imx93-9x9-lpddr4-qsb:~# ls /dev/ttyRPMSG*
```

There should be 11 device files from "/dev/ttyRPMSG0" to "/dev/ttyRPMSG10" if the default dtb file `imx93-9x9-qsb-uart-sharing-cm33.dtb` is used. The "/dev/ttyRPMSG0" to "/dev/ttyRPMSG9" have n:1 mapping to physical LPUART5, "/dev/ttyRPMSG10" is without "bus_id" and displays the message sent from Linux to M-core's debug console.

b. Check each virtual UART from "/dev/ttyRPMSG0" to "/dev/ttyRPMSG9" is connected to peer virtual UART between two boards, for example, use "minicom" to open and configure the same virtual UART on both boards:

```
root@imx93-9x9-lpddr4-qsb:~# minicom -s -D /dev/ttyRPMSG6
```

Then configure the UART as shown in the following figures:

![Figure 40. Configure RPMSG Virtual UART Step1](image)

Figure 40. Configure RPMSG Virtual UART Step1
4.6  Heterogeneous Multicore VirtIO and networking sharing

4.6.1  Heterogeneous Multicore VirtIO

Heterogeneous Multicore VirtIO leverages para-virtualization VirtIO technology to build resource sharing between Heterogeneous asymmetric multiprocessing (AMP). The main difference from para-virtualization VirtIO is that Heterogeneous Multicore VirtIO does not use and depend on any hypervisor. Therefore, it can be used for resource sharing between Cortex-A and Cortex-M cores, or between multiple Cortex-A cores.

The VirtIO is a standard for para-virtualization to provide high-performance IO device virtualization for VM. The Figure 42 shows the architecture of the VirtIO solution.

Save the settings and go back to minicom main window, then input any characters in one board's minicom window. Then, the characters would be displayed on the other board's minicom window.

![Configure RPMSG Virtual UART Step2](image)
The frontend VirtIO driver runs in the guest kernel space, and the backend VirtIO device runs in the Hypervisor. The Virtqueue and Vring via shared memory provide data transfer capability. The Hypervisor emulates the device logic by VMExit and injecting vCPU IRQ. Therefore, para-virtualization VirtIO is used for resource sharing between Virtual Machine guest OS and host OS, and it depends on the Hypervisor to run VirtIO backend.

The Heterogeneous Multicore VirtIO in Real Time Edge leverages VirtIO technology. However, it runs VirtIO backend on any CPU core including Cortex-A core and Cortex-M core. The VirtIO frontend runs on any other CPU core. This technology uses VirtIO to establish communication between the Frontend and Backend, which run on different CPU cores. Therefore, VirtIO can be used to share hardware resources between different CPU cores.

In the current implementation, the backend runs RTOS and owns the hardware resource, such as peripherals. The frontend runs on Linux. As there is no Hypervisor providing VMExit and vCPU IRQ injecting mechanism, the hardware or software mailbox between the frontend and backend is needed. The Figure 43 shows the architecture of the Heterogeneous Multicore VirtIO.

Heterogeneous Multicore VirtIO leverages shared memory to build Vring structure and data buffers. The shared memory should be coherent from Frontend and Backend.
4.6.2 Heterogeneous Multicore VirtIO performance evaluation

A VirtIO transmission device is introduced for evaluating the performance between the frontend and backend through virtqueues. There are 2 virtqueues for transmitting and receiving directions separately, and the device configuration registers are used to configure and control the test cases.

The Table 35 lists the combination of supported cases.

<table>
<thead>
<tr>
<th>Direction</th>
<th>Pkt size</th>
<th>Frontend Buffer copy</th>
<th>Backend Buffer copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX (Linux -&gt; FreeRTOS)</td>
<td>Max 2KB</td>
<td>Y/N</td>
<td>Y/N</td>
</tr>
<tr>
<td>RX (FreeRTOS -&gt; Linux)</td>
<td>Max 2KB</td>
<td>Y/N</td>
<td>Y/N</td>
</tr>
</tbody>
</table>

4.6.3 Heterogeneous Multicore VirtIO network sharing

The following figure shows the heterogeneous Multicore VirtIO network sharing architecture.

The virtual networking frontend runs on Cortex-A core, the frontend in Linux reuses the existing "drivers/net/virtio_net.c" driver by selecting kernel configuration item "CONFIG_VIRTIO_NET". RTOS frontend driver is not enabled in this release.

The Virtual Networking Backend can run in RTOS on Cortex-A core or Cortex-M core. The virtio-net backend drivers use Heterogeneous Multicore VirtIO to communicate with virtio-net frontend. It includes two data paths to handle data packets receiving/transmitting of frontend and one control path to handle control requirements from the frontend.

A Virtual Switch in the backend is used to switch packets from different ports. The switch ports include one "remote port" and many "local ports". In general, a "remote port" is a physical Ethernet port such as physical ENET port, which is used to receive/transmit packets from/to the physical Ethernet port. Here, "local port" refers to the virtual software port, such as the local port for virtio-net backend used to receive/transmit from/to virtio-net
backend. In fact, the packet is from/to virtio-net frontend through data path of Heterogeneous Multicore VirtIO
and another type of “local port” is used to connect to “virt-net” on RTOS to provide virtual Ethernet interface for
RTOS locally.

In the current implementation, an Ethernet L2 switch functions as the Virtual Switch. Each “local port” has
different MAC address, so the packets received from “remote port” can be switched to the destination “local
port” according to destination MAC address in the networking packets. The packets whose destination MAC
address does not match any “local port” are discarded, except broadcast packets. For the packets received
from the “local port”, the switch tries to check whether destination MAC address matches the address of other
“local ports”. If a matching entry is found, the packet is switched to the matched “local port”, so that the virtual
switch can implement switching packets between “local ports” locally. In case a match is not found, the packets
are sent to the external by “remote port”. That is to say Virtual Switch supports “local switch” and “remote
switch”.

The Virtual Switch can connect to multiple “local port”, that is to say single physical Ethernet port, controlled by
CPU Core running backend, can be shared with multiple OS running on different CPU Core by though multiple
frontend, and local virtual Ethernet driver in backend also provide Ethernet service for backend CPU Core
locally.

Note: In this release, “virt-net” on backend is not included and cannot provide local Ethernet service for
backend CPU Core. The “virt-net” frontend on RTOS is also not included in this release.

4.6.4 Building Heterogeneous Multicore VirtIO backend firmware

Refer to RTEDGEYOCTOUG to set up Yocto environment and build the nxp-image-real-time-edge. All demo
applications are located in the /examples directory of the rootfs.

The command below is used to compile the demo separately.

```
bitbake <multicore-app-name>
```

Where: 

"<multicore-app-name>" can be "virtio-perf-ca", "virtio-perf-cm", "virtio-net-backend-ca" or "virtio-net-
backend-cm".

The backend firmware is located in the directory:

tmp/deploy/images/imx8mm-lpddr4-evk/examples/heterogeneous-multicore/

For example, for i.MX8MM, the following binary images are located:

• "virtio-perf-cm/release/virtio_perf_cm4.bin"
• "virtio-perf-ca/ddr_release/virtio_perf_ca53.bin"
• "virtio-net-backend-cm/release/virtio_net_backend_cm4.bin"
• "virtio-net-backend-ca/ddr_release/virtio_net_backend_ca53.bin"

4.6.5 Building Heterogeneous Multicore VirtIO Frontend linux images

Refer to the RTEDGEYOCTOUG to set up Yocto environment and build the nxp-image-real-time-edge,
Linux image. Use the dtb files, which are located in rootfs images.

Alternatively, use the command below to compile the demo separately.

```
bitbake linux-imx
```

Then copy the following images to board’s first FAT32 partition:

• arch/arm64/boot/Image: Kernel Image
4.6.6 Running VirtIO performance testing on i.MX 8M Mini EVK

Perform the following steps for VirtIO performance testing:

1. **Set up the UART console for Frontend and Backend:**
   Connect the DEBUG UART slot on the board to your PC through the USB Cable. This step creates two USB serial ports on the PC. Open 2 UART consoles to connect these two USB serial ports respectively, using the following setup:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit

   One USB serial port is used for Linux that runs VirtIO frontend, the other can be used for RTOS that runs the VirtIO backend.

2. **Boot backend on Cortex-A core or Cortex-M core**
   Heterogeneous Multicore VirtIO Backend can run on Cortex-A core or Cortex-M core to evaluate different use cases.
   a. **Run the backend on Cortex-M core**
      On the U-Boot command line, execute the following commands to boot Cortex-M core with backend firmware:
      ```
      => ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/virtio-perf-cm/release/virtio_perf_cm4.bin
      => cp.b 0x48000000 0x7e00000 0x20000
      => bootaux 0x7e0000
      => setenv fdtfile imx8mm-evk-virtio-perf-cm4.dtb
      => setenv mmcargs $mmcargs clk_ignore_unused
      => run bsp_bootcmd
      ```
      Or Run the backend on Cortex-A53 by executing the following command in U-Boot command line:
      ```
      => ext4load mmc 1:2 0x93c000000 /examples/heterogeneous-multicore/virtio-perf-ca/ddr_release/virtio_perf_ca53.bin
      => dcache flush; icache flush;
      => cpu 3 release 0x93c00000
      => setenv fdtfile imx8mm-evk-virtio-perf-ca53.dtb
      ```
3. Use the “vt_test.sh” tool in Linux to start the performance testing use case. The following is help information of the tool.

```
root@imx8mm-lpddr4-evk:~# vt_test.sh -h
-s: Packet size: max 2048 Bytes, default: 64 Bytes
-r: Regression times: default: 1000
-t: Test type: 0: TX; 1: RX
-b: Backend copy buffer option: 0: not copy; 1: copy
-f: Frontend copy buffer option: 0: not copy; 1: copy
-h: This USAGE info
```

a. “-s” specifies the packet size to be used for testing, such as “-s 64”, it uses 64-byte packets for testing.
b. “-t” specifies the testing direction:
   - “-t 0” means the test sends packets from frontend (Linux) to backend (RTOS on A-Core or M-Core),
   - “-t 1” means the test sends packets from backend (RTOS on A-Core or M-Core) to frontend (Linux).
c. “-r” specifies the regression times:
   - “-r 100000” indicates that the test case sends 100000 packets with the direction from backend to frontend or frontend to backend, which is specified by the “-t” parameter.
d. “-b” specifies whether there is a memory copy in the backend:
   - For the memory copy case, use “-b 1”, for enabling memory copy from Vring buffer to user application buffer when receiving packet from frontend, or copy from user application buffer to Vring buffer when transmitting packets to frontend.
   - For no memory copy case “-b 0”, specifies there will be no memory copy in backend for each packet receiving or transmitting.
e. “-f” specifies whether there is a memory copy in the frontend.

For example, see the command below:

```
vtest.sh -s 64 -r 1000000 -t 0 -b 0 -f 0
```

The above test case transmits 1000000 packets from frontend (Linux) to backend (RTOS on A-Core or M-Core), each packet size is 64 bytes, there is no memory copy both on frontend and backend. The test log is as follows:

```
root@imx8mm-lpddr4-evk:~# vt_test.sh -s 64 -r 1000000 -t 0 -b 0 -f 0
[20.561527] ***********************************************
[20.561539] Front-end: interrupt mode
[20.561543] Back-end: interrupt mode
[20.561544] Front-end: do NOT copy buffer
[20.561546] Back-end: do NOTcopy buffer
[20.561547] Test case: RX
[20.561547] pkt_size: <64>
[20.561547] regress times: <1000000>
[21.868494] tx_test: pkt_size (64 B), pkt_cnt (1000000), period (1298108 µs)
```

The log shows that 1000000 packets are transmitted from frontend to backend in 1298108 µs. Therefore, the performance is 770 kpps or 394 Mbit/s.

4.6.7 Running VirtIO network sharing

VirtIO Network Sharing supports running on i.MX 8M Mini, i.MX 8M Plus, and i.MX 93 platforms.

1. Set up the UART console for the frontend and backend
Connect the DEBUG UART slot on the board to your PC through the USB Cable. On the PC, this step creates two USB serial ports (port0 and port1) for i.MX 8M Mini EVK board, and four USB serial ports (port0 ~ port3) for i.MX 8M Plus EVK board and i.MX93 EVK board. Open two UART consoles for UART port0 and port1 on i.MX 8M Mini EVK board or port2 and port3 on i.MX 8M Plus EVK board and i.MX93 EVK board UART with the following setup:

- 115200
- No parity
- 8 data bits
- 1 stop bit

The first UART console is used for Linux that runs the VirtIO frontend. The UART console is used for RTOS, which runs the VirtIO backend.

2. Hardware setup
Connect ENET port on EVK board to a networking switch or another board by using an Ethernet cable. For i.MX 8M Mini EVK board, there is a single Ethernet port on the board. So, use this port for testing. For i.MX 8M Plus EVK board and i.MX93 EVK board, there are two Ethernet ports are on the board. The first one, which is close to DEBUG USB port is ENET port and it is used for VirtIO Networking Sharing. So connect this port to the networking link. Another Ethernet port is an ENET QoS port and it is not used for this demo.

3. Booting backend and frontend
Heterogeneous Multicore VirtIO backend can run on Cortex-A core or Cortex-M core.

- Running Virtio Networking Backend on i.MX 8M
  a. Run the backend on a Cortex-M4 core

  On U-Boot command prompt, execute the following commands to boot Cortex-M core with firmware:

  ```
  => ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/virtio-net-backend-cm/release/virtio_net_backend_cm4.bin
  => cp.b 0x48000000 0x7e0000 0x20000
  => bootaux 0x7e0000
  ```

  Then boot Linux kernel:

  ```
  => setenv fdtfile imx8mm-evk-virtio-net-cm4.dtb
  => setenv mmcargs $mmcargs mem=2048MB clk_ignore_unused
  => run bsp_bootcmd
  ```

  b. Or run the backend on the Cortex-A53 core

  Execute the following command in the U-Boot command line:

  ```
  => ext4load mmc 1:2 0x93c00000 /examples/heterogeneous-multicore/virtio-net-backend-ca/ddr_release/virtio_net_backend_ca53.bin
  => dcache flush; icache flush;
  => cpu 3 release 0x93c00000
  ```

  Then, boot the Linux kernel:

  ```
  => setenv fdtfile imx8mm-evk-virtio-net-ca53.dtb
  => setenv mmcargs $mmcargs maxcpus=3 clk_ignore_unused
  => run bsp_bootcmd
  ```

- Running Virtio Networking Backend on i.MX 8MP
  a. Run the backend on Cortex-M7 core

  U-Boot command prompt, execute the following commands to boot Cortex-M core with firmware:

  ```
  => ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/virtio-net-backend-cm/release/virtio_net_backend_cm7.bin
  => cp.b 0x48000000 0x7e0000 0x20000
  => bootaux 0x7e0000
  ```
Then, boot Linux kernel:

```bash
=> setenv fdtfile imx8mp-evk-virtio-net-cm7.dtb
=> setenv mmcargs $mmcargs mem=2048MB clk_ignore_unused
=> run bsp_bootcmd
```

b. Or running the backend on Cortex-A53 core
Execute the following command in U-Boot command line:

```bash
=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/virtio-net-backend-ca/ddr_release/virtio_net_backend_ca53.bin
=> dcache flush; icache flush;
=> cpu 3 release 0xC0000000
```

Then boot Linux kernel:

```bash
=> setenv fdtfile imx8mp-evk-virtio-net-ca53.dtb
=> setenv mmcargs $mmcargs maxcpus=3 clk_ignore_unused
=> run bsp_bootcmd
```

- Running Virtio Networking Backend on i.MX93
  
a. Running the backend on Cortex-M33

On U-Boot command prompt, execute the following commands to boot Cortex-M core with firmware:

```bash
=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/virtio-net-backend-cm/release/virtio_net_backend_cm33.bin
=> cp.b 0xd0000000 0x201e0000 20000
=> bootaux 0x1ffe0000
```

Then boot Linux kernel:

```bash
=> setenv fdtfile imx93-11x11-evk-virtio-net-cm33.dtb
=> setenv mmcargs $mmcargs clk_ignore_unused
=> run bsp_bootcmd
```

b. Or running the backend on Cortex-A55
Execute the following command in the U-Boot command line:

```bash
=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/virtio-net-backend-ca/ddr_release/virtio_net_backend_ca55.bin
=> dcache flush && icache flush
=> cpu 1 release 0xd0000000
```

Then boot Linux kernel:

```bash
=> setenv fdtfile imx93-11x11-evk-virtio-net-ca55.dtb
=> setenv mmcargs $mmcargs maxcpus=1 clk_ignore_unused
=> run bsp_bootcmd
```

4. Evaluate Networking Sharing

After backend starts, the second UART console displays the following backend log:

```bash
Starting Virtio networking backend...
virtio network device initialization succeed!
Switch enabled with enet remote port succeed!
ENET: PHY link is up with speed 1000M full-duplex
```

After the kernel boots up, use "ifconfig" and "ping" commands to verify the virtual networking interface. In the following log, "eth0" is virtio_net interface, but it may be different on different platform. So, use "ethtool" to find out the virtio_net interface that is using "virtio_net" driver, and the default MAC address of virtio_net interface is "00:04:9f:00:01:02".

```bash
root@imx8mm-lpddr4-evk:~# ifconfig
eth0: flags=4163<UP,BROADCAST,ModelState:RUNNING,MULTICAST> mtu 1500
```

---

**User guide**

© 2023 NXP B.V. All rights reserved.

**Rev. 2.7 — 18 December 2023**

134 / 400
inet 192.168.1.107 netmask 255.255.255.0 broadcast 192.168.1.255
inet6 fd08:d7d5:e652::733 prefixlen 128 scopeid 0x0<global>
inet6 fd08:d7d5:e652:0:201:2ff:fe03:405 prefixlen 64 scopeid 0x0<global>
inet6 fe80::201:2ff:fe03:405 prefixlen 64 scopeid 0x20<link>
ether 00:04:9f:00:01:02 txqueuelen 1000 (Ethernet)
RX packets 54 bytes 5544 (5.4 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 143 bytes 20887 (20.3 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@imx8mm-lpddr4-evk:~# ethtool -i eth0
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: b8400000.virtio_net
supports-statistics: yes
supports-test: no
supports-eprom-access: no
supports-register-dump: no
supports-priv-flags: no

root@imx8mm-lpddr4-evk:~# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.888 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.541 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=2.13 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=2.29 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=1.73 ms

Use the following command to change the MAC address of virtio_net:
root@imx8mm-lpddr4-evk:~# ifconfig eth0 hw ether 00:04:9f:00:01:03

4.7 Unified Life Cycle management

4.7.1 Overview

Heterogeneous Multicore Framework provides unified Life Cycle Management for both Cortex-A and Cortex-M cores.

Real-time Edge supports bootstrapping the native Zephyr and native FreeRTOS on the Cortex-A core and Cortex-M core with U-Boot command and with the RemoteProc under Linux as listed in the below table.

<table>
<thead>
<tr>
<th>Core Type</th>
<th>U-Boot</th>
<th>RemoteProc on Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>NativeZephyr on M core</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NativeFreeRTOS on M core</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>NativeZephyr on A core</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>NativeFreeRTOS on A core</td>
<td>Y</td>
<td>-</td>
</tr>
</tbody>
</table>
4.7.2 Booting Native RTOS Cortex-A core image from U-Boot

Downloading the Pure RTOS Cortex-A Core image into DDR memory firstly, and then use U-Boot command "go" to boot the image from Core0 or use U-Boot command "cpu" to boot the image from the other Cortex-A Cores.

Refer to Section 3.5.3 for details.

4.7.3 Booting Native RTOS Cortex-M Core image from U-Boot

U-boot command "bootaux" is used to boot Cortex-M Core RTOS Image from U-Boot.

For example, after the board is booted into the U-Boot console, use the following command to boot Arm Cortex-M core on i.MX 8M Mini EVK board or i.MX 8M Plus EVK board:

```
=> ext4load mmc 1:2 0x48000000 /examples/mcux-sdk/freertos-hello/release/freertos_hello.bin; cp.b 0x48000000 0x7e0000 20000;
=> bootaux 0x7e0000
```

Or use the following command on i.MX93 EVK board:

```
=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/hello-world-cm/release/hello_world_cm33_UART2.bin; cp.b 0xd0000000 0x201e0000 20000;
=> bootaux 0x1ffe0000
```

4.7.4 Using RemoteProc to boot RTOS Cortex-M Core Image

If you choose to use RemoteProc to start the remote core directly, execute run prepare_mcore in U-Boot before starting the Linux OS.

```
=> run prepare_mcore
```

Then, use the following command to use RPMSG dtb file to boot the kernel:

```
# On imx8mm-lpddr4-evk board
=> setenv fdtfile imx8mm-evk-rpmsg.dtb
=> boot

# On imx8mp-lpddr4-evk board
=> setenv fdtfile imx8mp-evk-rpmsg.dtb
=> boot
```

Then, after the Linux kernel boots up, run the commands for i.MX 8MP:

```
root@imx8mp-lpddr4-evk:~# echo -n imx8mp_m7_TCM_hello_world.elf > /sys/class/remoteproc/remoteproc0/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/class/remoteproc/remoteproc0/state
[ 19.668712] remoteproc remoteproc0: powering up imx-rproc
[ 19.670341] remoteproc remoteproc0: Booting fw image
  imx8mp_m7_TCM_hello_world.elf, size 153316

root@imx8mp-lpddr4-evk:~# [ 20.191036] remoteproc remoteproc0: remote processor imx-rproc is now up
```
For i.MX 8MM, run the following commands:

```
root@imx8mm-lpddr4-evk:~# echo -n imx8mm_m4_TCM_hello_world.elf > /sys/class/remoteproc/remoteproc0/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/class/remoteproc/remoteproc0/state
[ 209.654414] remoteproc remoteproc0: powering up imx-rproc
[ 209.656646] remoteproc remoteproc0: Booting fw image
imx8mm_m4_TCM_hello_world.elf, size 146136
root@imx8mm-lpddr4-evk:~# [ 210.174456] remoteproc remoteproc0: remote processor imx-rproc is now up
```

After these steps are followed, the remote processor `imx-rproc` is up.
5 Heterogeneous Multi-SoC Framework

This section describes the features of heterogeneous Multi-SoC framework and how to implement it using NXP hardware platforms.

5.1 Introduction

Heterogeneous Multi-SoC Framework enables the usage of a combination of MPU and i.MX RT1180 as an Industrial Switch. This extends the MPU hardware capability with the i.MX RT1180 hardware capability, thereby providing switch functionality, TSN functionality, and the capability of supporting different Industrial Protocols. The i.MX RT1180 can be used to run real-time tasks such as industrial protocols in the real-time domain, whereas the MPU can process compute-heavy tasks, in the non-real-time domain.

The external Ethernet ports of i.MX RT1180 can be exposed to the MPU side as standard Ethernet interfaces as data path. Different interfaces such as SPI or I2C can be used as the management interface between MPU and i.MX RT1180.

5.2 Software architecture

The Linux Distributed Switch Architecture (DSA) framework is used to expose the i.MX RT1180 NETC switch ports to MPU side. In this architecture, one of the NETC switch ports or ENETC port is used as the data interface. Different interfaces (for example LPSPI, I2C or message unit) can be used as management interfaces.

For more information regarding Linux DSA, refer to https://docs.kernel.org/networking/dsa/dsa.html.

![Diagram of using switch port as DSA CPU port which is connected to MPU](image)

**Note:** The industrial protocols listed in the box of i.MX RT1180 are not supported yet.
Figure 46. Using ENETC port as DSA CPU port which is connected to MPU

Note: The configuration illustrated in this figure is not supported in this release yet.

The software architecture implementation includes:

- NETC DSA switch driver on Linux
- Device driver of DSA control interface on Linux DSA
- Service driver of DSA control interface on i.MX RT1180
- NETC DSA switch configuration on i.MX RT1180
  - Using NETC switch port as DSA CPU port
  - Using ENETC port as DSA CPU port (not supported in this release yet)

The external Ethernet ports of i.MX RT1180 are exposed to MPU with Linux DSA framework. The ports can be viewed as standard Ethernet ports which could support the below operations:

- Binding Linux IP address to a specific port
- Broadcast on a specific port and others

To identify the source traffic of incoming traffic, the outer VLAN tag is used as the DSA tag to indicate which port the traffic is coming from. The VID field of the VLAN tag is encoded to include the source/destination port. Below is the description of the 12-bit VID field:

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSV</td>
<td>VBID</td>
<td>SWITCH_ID</td>
<td>VBID</td>
<td></td>
<td></td>
<td>PORT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The above custom VID definition is coming from the Linux tag_8021q source code (net/dsa/tag_8021q.c) which is used by the Linux NETC DSA driver. As a result, VLAN ID 3072 to 3076 as well as 3088 are reserved by the Linux NETC DSA driver and will not be allowed to be used by the user.

5.2.1 Using one of the i.MX RT1180 switch ports as DSA CPU port

Consider the case when one of the i.MX RT1180 switch ports is used as DSA CPU port that is connected to MPU. In such a situation, the packets received on the external ports of i.MX RT1180 are filtered using Ingress Port Filter Table. The incoming packets will be filtered according to the ingress port ID, and forwarded to the DSA CPU port by either stream forwarding (by-passing the 802.1Q bridge forwarding) or 802.1Q bridge
forwarding using Ingress Stream Table. The outer VLAN tag is added by using Egress Treatment Table on egress DSA CPU port.

The packets received on the DSA CPU port are filtered according to the DSA tag, which is viewed a VID from i.MX RT1180 perspective. This is done by using Ingress Port Filter Table and the packets are directed to the port mapped to the VID by using Ingress Stream Table. The outer VLAN tag is removed on ingress DSA CPU port by also using Ingress Stream Table.

5.2.2 Using i.MX RT1180 ENETC port as DSA CPU port

Note: This configuration is not supported in this release yet.

For the case using the ENETC port as the DSA CPU port that is connected to MPU, the packets received on the external ports of i.MX RT1180 are forwarded to the CPU of i.MX RT1180 by 802.1Q bridge forwarding. The software bridge stack running on the CPU of i.MX RT1180 will add the DSA tag and forward the packets to the ENETC port.

The packets received on the DSA CPU port are received by the software bridge stack. The software bridge stack removes the DSA tag and then forwards the packets to the switch port accordingly.

5.3 Running the Heterogeneous Multi-SoC Framework

This section describes the process to implement and run the heterogeneous Multi-SoC framework on supported MPU platforms and i.MX RT1180 EVK.

5.3.1 Building and running on MPU + i.MX RT1180 EVK

The currently supported MPU platforms include i.MX 8M Plus EVK and i.MX 93 EVK.

Note: For i.MX RT1180 EVK, the supported board revision is SCH-50577 REV C2 (700-50577 REVC), which has i.MX RT1180 Rev. B0 chip.

5.3.1.1 Hardware setup for i.MX 8M Plus EVK and i.MX RT1180 EVK

1. SPI connection between i.MX 8M Plus EVK and i.MX RT1180 EVK

On i.MX 8M Plus EVK, ECSPi2 pins are available on J21 connector, but due to signal incompatibility with the i.MX RT1180 EVK LPSPi3 pins on J44 connector, board rework is needed on i.MX 8M Plus EVK to replace a translating transceiver NBT0104 U55 which has limited capacitive loading for 70 pF with NTS0104GU12, which has internal pull-up of 10K and bigger capacitive loading. Refer to the schematic shown in the Figure 47 below:
Figure 47. Hardware rework for SPI signals on i.MX 8M Plus EVK
Connect ECSPI2 pins on J21 connector i.MX RT1180 EVK LPSPI3 pins by following the connection in Table 38.

Table 38. Pin connection between i.MX 8M Plus EVK and i.MX RT1180 EVK

<table>
<thead>
<tr>
<th>i.MX 8M Plus EVK</th>
<th>Connection</th>
<th>i.MX RT1180 EVK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>Function</td>
<td>Pin</td>
</tr>
<tr>
<td>21</td>
<td>ECSPI2_MISO</td>
<td>10</td>
</tr>
<tr>
<td>19</td>
<td>ECSPI2_MOSI</td>
<td>8</td>
</tr>
<tr>
<td>23</td>
<td>ECSPI2_SCLK</td>
<td>12</td>
</tr>
<tr>
<td>24</td>
<td>ECSPI2_SS0</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>GND</td>
<td>14</td>
</tr>
</tbody>
</table>

2. Ethernet connection between i.MX 8M Plus EVK and i.MX RT1180 EVK
Using Ethernet cable to connect i.MX 8M Plus EVK ENET2 port (eth1 in Linux) on J8 RJ45 connector and i.MX RT1180 EVK NETC switch port 3 (ENET3) on J31 RJ45 connector.

5.3.1.2 Hardware setup for i.MX 93 EVK and i.MX RT1180 EVK
1. LPSPI connection between i.MX 93 EVK and i.MX RT1180 EVK
Using flying wire to connect i.MX 93 EVK LPSPI3 pins on J1001 connector and i.MX RT1180 EVK LPSPI3 pins on J44 connector by following the pin connection in below table:

Table 39. Pin connection between i.MX 93 EVK and i.MX RT1180 EVK

<table>
<thead>
<tr>
<th>i.MX 93 EVK</th>
<th>Connection</th>
<th>i.MX RT1180 EVK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>Function</td>
<td>Pin</td>
</tr>
<tr>
<td>19</td>
<td>LPSPI3_SOUT</td>
<td>10</td>
</tr>
<tr>
<td>21</td>
<td>LPSPI3_SIN</td>
<td>8</td>
</tr>
</tbody>
</table>
Table 39. Pin connection between i.MX 93 EVK and i.MX RT1180 EVK...continued

<table>
<thead>
<tr>
<th></th>
<th>Function</th>
<th>Connection</th>
<th></th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>LPSPI3_CLK</td>
<td>&lt;-&gt;</td>
<td>12</td>
<td>LPSPI3_CLK</td>
</tr>
<tr>
<td>24</td>
<td>LPSPI3_PCS0</td>
<td>&lt;-&gt;</td>
<td>6</td>
<td>LPSPI3_PCS0</td>
</tr>
<tr>
<td>25</td>
<td>GND</td>
<td>&lt;-&gt;</td>
<td>14</td>
<td>GND</td>
</tr>
</tbody>
</table>

2. **Ethernet connection between i.MX 93 EVK and i.MX RT1180 EVK**

Using Ethernet cable to connect i.MX 93 EVK ENET_QOS port (eth1 in Linux) on J501 RJ45 connector and i.MX RT1180 EVK NETC switch port 3 (ENET3) on J31 RJ45 connector. In DSA terminology, the ENET_QOS Ethernet controller on i.MX 93 is referred to as the ‘master’ interface, while the NETC switch port 3 on i.MX RT1180 is referred to as ‘CPU’ port.

5.3.1.3 Building the DSA switch application for i.MX RT1180 EVK

The demo application images `dsa_switch.elf` and `dsa_switch.bin` for i.MX RT1180 EVK are installed into target roofs with the MPU Yocto image build, and are available in `/examples/heterogeneous-multi-soc/dsa-switch-evkmimirxrti1180-cm33/` directory.

Also, the demo application images can be deployed into the Yocto build directory by using the following Yocto commands:

```bash
# For i.MX 8M Plus EVK
$ DISTRO=nxp-real-time-edge MACHINE=imx8mp-lpddr4-evk source real-time-edge-setup-env.sh -b <build_dir>
$ bitbake dsa-switch-evkmimirxrti1180-cm33
# For i.MX 93 EVK
$ DISTRO=nxp-real-time-edge MACHINE=imx93evk source real-time-edge-setup-env.sh -b <build_dir>
$ bitbake dsa-switch-evkmimirxrti1180-cm33
```

After the Yocto bitbake command is used, the demo application images can be found in the below directory on the build host:

```bash
<build_dir>/tmp/deploy/images/<MACHINE>/examples/heterogeneous-multi-soc/dsa-switch-evkmimirxrti1180-cm33
```

The demo applications are available in both release and ram release mode. The release image boots from the external QuadSPI NOR flash and then relocates to internal memory (Code TCM, System TCM and OCRAM) and needs to be flashed in the external NOR flash. The ram release image boots from internal memory and needs to be loaded directly into internal memory by host debug tools.

5.3.1.4 Bringing up MPU and i.MX RT1180 EVK

1. Make SPI and Ethernet connections between i.MX 8M Plus EVK or i.MX 93 EVK and i.MX RT 1180 EVK by following the steps in the Section Section 5.3.1.1 or Section 5.3.1.2.

2. Make a serial connection between J23 connector on i.MX 8M Plus EVK or J1401 connector on i.MX 93 EVK and USB connector on host PC using USB cable. Open the terminal application on host PC, such as PuTTY or TeraTerm, and connect to the 3rd debug console which is used by Linux running on MPU.

3. Make a serial connection between J53 connector on i.MX RT1180 EVK and USB connector on host PC using USB cable. Open another terminal application on host PC, and connect to the debug console which is used by FreeRTOS running on i.MX RT1180 EVK. Note that the terminal settings ‘Implicit CR in every LF’ should be enabled for correct display of the logs.
4. Power on i.MX RT1180 EVK board, and flash the binary image `release/dsa_switch.bin` on i.MX RT1180 EVK and power cycle the board. Alternatively, one can download the elf image `ram_release/dsa_switch.elf` by host debug tools (e.g. using JTAG debugger connected to J37 connector). After the demo application is running, there should be continuous logs displayed in the console.

5. Power on the MPU board and enter U-Boot command line. Then, execute the following commands:

```bash
# For i.MX 8M Plus EVK
u-boot=> setenv fdtfile imx8mp-evk-revb4-dsa.dtb
u-boot=> boot

# For i.MX 93 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-dsa.dtb
u-boot=> boot
```

After Linux is up on MPU, refer to the Section 5.3.1.5 for the detailed usage on how to show and configure various NETC switch features on i.MX RT1180.

5.3.1.5 Runtime usage on MPU and i.MX RT1180 EVK

This section describes the major i.MX RT1180 NETC switch features that can be queried or configured on MPU + i.MX RT1180 EVK heterogeneous multi-SoC architecture using Linux DSA (Distributed Switch Architecture).

5.3.1.5.1 i.MX RT1180 NETC switch interface under Linux

On MPU, after Linux is up, on successful initialization of the Linux NETC DSA driver, the i.MX RT1180 NETC front panel switch port 0 (ENET0), port 1 (ENET1), port 2 (ENET2) should have a network device interface attached with the `swpX` name format.

<table>
<thead>
<tr>
<th>i.MX RT1180 EVK NETC switch port</th>
<th>Label on i.MX RT1180 EVK</th>
<th>Linux network device name on MPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 0</td>
<td>ENET0 (J28)</td>
<td>swp0</td>
</tr>
<tr>
<td>Port 1</td>
<td>ENET1 (J29)</td>
<td>swp1</td>
</tr>
<tr>
<td>Port 2</td>
<td>ENET2 (J30)</td>
<td>swp2</td>
</tr>
<tr>
<td>Port 3</td>
<td>ENET3 (J31)</td>
<td>N/A</td>
</tr>
<tr>
<td>Port 4 (internal)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Note:** Linux DSA does not currently create user network device for CPU port. The i.MX RT1180 NETC switch port 3 is used as CPU port. Therefore, it is not visible to the user as a normal network device in Linux.

One can use `ifconfig -a` command to show these network device interfaces.

**Example - NETC switch port interfaces** `swpX` shown by `ifconfig -a`

```bash
# ifconfig -a
[...]
swp0: flags=4098<BC,D,M> mtu 1500
    ether 00:04:9f:08:49:31 txqueuelen 1000 (Ethernet)
    RX packets 0  bytes 0  (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0  (0.0 B)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

swp1: flags=4098<BC,D,M> mtu 1500
    ether 00:04:9f:08:49:31 txqueuelen 1000 (Ethernet)
    RX packets 0  bytes 0  (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
```
Also, the `ip link show` command uses the `swpX@eth1` name format to also indicate the associated master Ethernet interface (`eth1`) on MPU for the DSA switch port.

Example - NETC switch port interfaces `swpX` and DSA master interface `eth1` shown by `ip link show`.

```
# ip link show
[..]
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1504 qdisc mq state UP mode DEFAULT group default qlen 1000
   link/ether 00:04:9f:08:49:31 brd ff:ff:ff:ff:ff:ff
4: swp0@eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
   group default qlen 1000
   link/ether 00:04:9f:08:49:31 brd ff:ff:ff:ff:ff:ff
5: swp1@eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
   group default qlen 1000
   link/ether 00:04:9f:08:49:31 brd ff:ff:ff:ff:ff:ff
6: swp2@eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
   group default qlen 1000
   link/ether 00:04:9f:08:49:31 brd ff:ff:ff:ff:ff:ff
```

### 5.3.1.5.2 Maximum frame size configuration

The Linux NETC DSA driver correlates between Layer-2 maximum frame size and Layer-3 MTU. Typical settings of Layer-2 maximum frame size is 1518 bytes for standard untagged frames and the corresponding Layer-3 MTU is 1500 bytes.

**Example 1:** Setting Layer-3 MTU of NETC switch port 1 (`swp1`) to 1000 bytes

```
# ifconfig swp1 mtu 1000
```

**Note:**

- Setting the Layer-3 MTU for an external NETC switch port on i.MX RT1180 EVK to a value larger than 1500 bytes also changes the Layer-3 MTU of both the CPU port on i.MX RT1180 EVK and the master Ethernet interface (`eth1`) on MPU accordingly.

As the maximum MTU of `eth1` is 1600, the maximum MTU value that can be configured on an external NETC switch port is 1596 bytes. However, the NETC Ethernet MAC supports configurable maximum frame size up to 2000 bytes. The 4 extra bytes in MTU for both the CPU port on i.MX RT1180 EVK and the master Ethernet interface (`eth1`) on MPU are used to store the 4-bytes 802.1Q DSA tag.

### 5.3.1.5.3 Single port mode for i.MX RT1180 NETC switch ports

After Linux is up on MPU, by default all the i.MX RT1180 NETC switch ports available in Linux works in single port mode. In this configuration mode the traffic received on other switch ports is forwarded to the CPU port.
(i.MX RT1180 NETC switch port 3). Each of the other external switch port interface can be used independently to send and receive packets.

**Example 1: Single port configuration of the Linux NETC DSA driver**

```c
/* configure IP address on external switch interfaces */
# ip addr add 192.168.1.1/24 dev swp0
# ip addr add 192.168.2.1/24 dev swp1
# ip addr add 192.168.3.1/24 dev swp2
/* master interface to be brought up first - up by default */
# ip link set eth1 up
/* bring up the switch slave interfaces */
# ip link set swp0 up
# ip link set swp1 up
# ip link set swp2 up
```

Assuming there is one remote host connected to each of the external switch ports, one can ping the remote host.

```c
/* Assuming the IP is 192.168.1.2 for the remote host connected to swp0 */
# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=1.75 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=1.77 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=1.76 ms
[...]
/* Assuming the IP is 192.168.2.2 for the remote host connected to swp1 */
# ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=3.23 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=1.75 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=1.75 ms
[...]
/* Assuming the IP is 192.168.3.2 for the remote host connected to swp2 */
# ping 192.168.3.2
PING 192.168.3.2 (192.168.3.2) 56(84) bytes of data.
64 bytes from 192.168.3.2: icmp_seq=1 ttl=64 time=2.80 ms
64 bytes from 192.168.3.2: icmp_seq=2 ttl=64 time=1.76 ms
64 bytes from 192.168.3.2: icmp_seq=3 ttl=64 time=1.77 ms
[...]
```

**5.3.1.5.4 Bridge mode for i.MX RT1180 NETC switch ports**

In this configuration mode, the external switch ports (DSA slave interfaces) are added to a bridge for L2 forwarding. The `eth1` interface is brought up as the DSA master interface.

**Example 1: Bridge configuration of the Linux NETC DSA driver**

```c
/* bring up master interface before the slave ports - up by default */
# ip link set eth1 up
/* bring up the switch slave interfaces */
# ip link set swp0 up
# ip link set swp1 up
# ip link set swp2 up
```
/* create bridge */
# ip link add name br0 type bridge
/* add the external switch ports to the bridge */
# ip link set dev swp0 master br0
# ip link set dev swp1 master br0
# ip link set dev swp2 master br0
/* configure IP address and bring up the bridge */
# ip addr add 192.168.2.1/24 dev br0
# ip link set dev br0 up

Assuming there is a remote host connected to one of the external switch ports, one can ping the remote host.

/* Assuming the IP is 192.168.2.2 for the remote host connected to
* one of the external switch ports (swp0, swp1 and swp2)
*/
# ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=1.76 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=1.80 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=1.77 ms
[…]

At the same time, the L2 forwarding works using the external switch ports (swp0, swp1 and swp2). Assuming there is one remote host connected to each of the external switch ports, each remote host can ping other hosts.

5.3.1.5.5 i.MX RT1180 NETC switch port statistics counters

The NETC DSA switch driver supports ethtool -S swpX statistics reporting for each external DSA slave switch port through the associated net devices. Note that the first 4 stats (tx_packets, tx_bytes, rx_packets, rx_bytes) are counted by Linux networking stack and the other stats are directly read from NETC switch hardware.

Example 1: Query the port statistics counters of NETC switch port 1 (swp1)

# ethtool -S swp1
NIC statistics:
  tx_packets: 232
tx_bytes: 30985
  rx_packets: 1178
  rx_bytes: 76913
  in-bytes: 150749
  in-valid-bytes: 150749
  in-pause-frames: 0
  in-valid-frames: 1846
  in-vlan-frames: 0
  in-uc-frames: 814
  in-mc-frames: 85
  in-bc-frames: 947
  in-frames: 1846
[…]
  out-bytes: 381591
  out-valid-bytes: 381591
  out-pause-frames: 0
  out-valid-frames: 4539
  out-vlan-frames: 0
  out-uc-frames: 810
  out-mc-frames: 3673
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>out-bc-frames</td>
<td>56</td>
</tr>
<tr>
<td>out-frames</td>
<td>4539</td>
</tr>
<tr>
<td>q0-rejected-bytes</td>
<td>0</td>
</tr>
<tr>
<td>q0-rejected-frames</td>
<td>0</td>
</tr>
<tr>
<td>q0-dequeue-bytes</td>
<td>0</td>
</tr>
<tr>
<td>q0-dequeue-frames</td>
<td>0</td>
</tr>
<tr>
<td>q0-dropped-bytes</td>
<td>0</td>
</tr>
<tr>
<td>q0-dropped-frames</td>
<td>0</td>
</tr>
<tr>
<td>q0-frames</td>
<td>0</td>
</tr>
<tr>
<td>q7-rejected-bytes</td>
<td>0</td>
</tr>
<tr>
<td>q7-rejected-frames</td>
<td>0</td>
</tr>
<tr>
<td>q7-dequeue-bytes</td>
<td>0</td>
</tr>
<tr>
<td>q7-dequeue-frames</td>
<td>0</td>
</tr>
<tr>
<td>q7-dropped-bytes</td>
<td>0</td>
</tr>
<tr>
<td>q7-dropped-frames</td>
<td>0</td>
</tr>
<tr>
<td>q7-frames</td>
<td>0</td>
</tr>
</tbody>
</table>
5.3.1.5.6 VLAN configuration

By default, the DSA switch application running on i.MX RT1180 EVK configures the NETC switch as VLAN filtering enabled. The bridge tool from iproute2 package can be used to manipulate the VLAN filter table.

To make the bridge VLAN aware, run the below command to toggle the `vlan_filtering` property on the bridge that already exists.

```
# ip link set dev br0 type bridge vlan_filtering 1
```

By default, only the default `pvid` (1) of the bridge is installed on all the switch ports. So, all VLAN-tagged traffic, except that tagged with VID 1, will be dropped.

**Note:** Since the default `pvid` (port-based VLAN) is 1, all untagged traffic will also get internally processed by the switch as having VID 1. So

- Untagged traffic is treated the same as traffic tagged with VID 1, or any other value that the `pvid` may have.
- Deleting VID 1 from the VLAN table will effectively block untagged traffic too.

**Note:** VLAN ID 3072 to 3076 as well as 3088 are reserved by the Linux NETC DSA driver and won’t be allowed to be used by the user.

**Example 1:** To add a new VLAN filter entry by which both the NETC switch port 1 (`swp1`) and port 2 (`swp2`) can accept and transmit tagged traffic with VLAN ID 100:

```
# bridge vlan add dev swp1 vid 100
# bridge vlan add dev swp2 vid 100
```

**Example 2:** To delete the VLAN filter entry added in previous example.

```
# bridge vlan delete dev swp1 vid 100
# bridge vlan delete dev swp2 vid 100
```

**Example 3:** To display the current VLAN filter table:

```
# bridge vlan show
```

It is also possible for the switch to tag untagged traffic with a different VLAN ID on ingress using the `pvid` option.

**Example 4:** To add new VLAN filter entries by which both the NETC switch port 1 (`swp1`) and port 2 (`swp2`) can accept and transmit tagged traffic with VLAN ID 100 and 200, also configure the PVID of NETC switch port 1 (`swp1`) to 100 and configure the PVID of NETC switch port 2 (`swp2`) to 200.

```
# bridge vlan add dev swp1 vid 100 pvid
# bridge vlan add dev swp2 vid 100
# bridge vlan add dev swp1 vid 200
# bridge vlan add dev swp2 vid 200 pvid
```

Using the above configuration, when an untagged packet enters `swp1`, it gets internally processed by the switch as having VID 100. If it is forwarded to `swp2` as a result of FDB lookup and exits `swp2`, it is tagged with VLAN ID 100. Similarly, when an untagged packet enters `swp2`, it gets internally processed by the switch as having VID 200. If it is forwarded to `swp1` as a result of FDB lookup and exits `swp1`, it is tagged with VLAN ID 200.
In the above example, if the VLAN tag added by PVID on the original untagged packet is not desired, it can be stripped on egress using the `untagged` option.

**Example 5**: To strip the VLAN tag added by PVID on untagged source traffic:

```
# bridge vlan add dev swp1 vid 100 pvid untagged
# bridge vlan add dev swp2 vid 100 untagged
# bridge vlan add dev swp1 vid 200 untagged
# bridge vlan add dev swp2 vid 200 pvid untagged
```

Using the above configuration, when an untagged packet enters `swp1`, it gets internally processed by the switch as having VID 100. If it is forwarded to `swp2` as a result of FDB lookup and exits `swp2`, it is still untagged. Similarly, when an untagged packet enters `swp2`, it gets internally processed by the switch as having VID 200. If it is forwarded to `swp1` as a result of FDB lookup and exits `swp1`, it is still untagged.

### 5.3.1.5.7 FDB configuration

By default, hardware MAC learning is enabled. FDB table entries `<MAC, VID, PORT>` are added or updated in the FDB table when packets with new unique `<MAC + VID>` are received on the NETC switch port.

**Example 1**: To display the current FDB table entries related to NETC switch port `swp1`

```
# bridge fdb show | grep swp1
```

**Example 2**: To add a static FDB entry associated with NETC switch port `swp1`

```
# bridge fdb add dev swp1 11:22:33:44:55:66 vlan 1 master static
```

Note that the VLAN filter entry for the VLAN ID specified for `vlan` option must have been added before adding the FDB entry for that VLAN ID.

**Example 3**: To delete a static FDB entry

```
# bridge fdb del dev swp1 11:22:33:44:55:66 vlan 1 master static
```
6 Real-time networking

6.1 Time Sensitive Networking (TSN) on NXP platforms

Time Sensitive Networking (TSN) is an extension to traditional Ethernet networks, providing a set of standards compatible with IEEE 802.1 and 802.3. These extensions intend to address the limitations of standard Ethernet in sectors ranging from industrial and automotive applications to live audio and video systems. Applications running over traditional Ethernet must be designed to be very robust in order to withstand corner cases such as packet loss, delay, or even reordering. TSN aims to provide guarantees for deterministic latency and packet loss under congestion. Therefore, it allows critical and non-critical traffic to be converged in the same network.

This chapter describes the process and use cases for implementing TSN features on the i.MX 8M LPDDR4 Plus, i.MX 8DXL LPDDR4 EVK, i.MX 93 EVK, i.MX 93 9x9 QSB, and LS1028ARDB boards.

6.1.1 TSN hardware capability

Table 41. TSN hardware capability on different platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>802.1Qbv (Enhancements for Scheduled Traffic)</th>
<th>802.1Qbu and 802.3br (Frame Preemption)</th>
<th>802.1Qav (Credit Based Shaper)</th>
<th>802.1AS (Precision Time Protocol)</th>
<th>802.1CB (Frame Replication and Elimination for Reliability)</th>
<th>802.1Qci (Per Stream Filtering and Policing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENETC (LS1028A)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Felix switch (LS1028A)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Stmac (i.MX 8DXL, i.MX 8M Plus, i.MX 93)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

6.1.2 TSN configuration

The table below describes the TSN configuration tools support on different platforms

Table 42. TSN configuration tool support on different hardware platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>802.1Qbv (Enhancements for Scheduled Traffic)</th>
<th>802.1Qbu and 802.3br (Frame Preemption)</th>
<th>802.1Qav (Credit Based Shaper)</th>
<th>802.1AS (Precision Time Protocol)</th>
<th>802.1CB (Frame Replication and Elimination for Reliability)</th>
<th>802.1Qci (Per Stream Filtering and Policing)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENETC (LS1028A)</td>
<td>tc-taprio, tsntool</td>
<td>ethtool tsntool</td>
<td>tc-cbs tsntool</td>
<td>ptp4l</td>
<td>N/A</td>
<td>tc-flower tsntool</td>
</tr>
<tr>
<td>Felix switch (LS1028A)</td>
<td>tc-taprio, tsntool</td>
<td>ethtool tsntool</td>
<td>tc-cbs tsntool</td>
<td>ptp4l, Gen AVB/TSN stack</td>
<td>tsntool</td>
<td>tc-flower tsntool</td>
</tr>
<tr>
<td>Stmac (i.MX 8DXL, i.MX 8M Plus, i.MX 93)</td>
<td>tc-taprio</td>
<td>ethtool</td>
<td>tc-cbs</td>
<td>ptp4l, Gen AVB/TSN stack</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
6.1.2.1 Using Linux traffic control (tc)

Enable the following configurations in kernel when using Linux traffic control (tc):

| [*] Networking support --->
| Networking options --->
| [*] QoS and/or fair queueing ---> |
| <*> Credit Based Shaper (CBS) |
| <*> Time Aware Priority (taprio) Scheduler |
| <*> Multi-queue priority scheduler (MQPRIO) |
| [*] Actions ---> |
| <*> Traffic Policing |
| <*> Generic actions |
| <*> Redirecting and Mirroring |
| <*> SRB Editing |
| <*> Vlan manipulation |
| <*> Frame gate entry list control tc action |

On IS1028A platform, ENETC QoS driver needs to be set to support tc configuration.

| Symbol: FSL_ENETC_QOS [=y] |
| Device Drivers ---> |
| [*] Network device support ---> |
| [*] Ethernet driver support ---> |
| [*] Freescale devices |
| [*] ENETC hardware Time-sensitive Network support |

1. The below link provides details for using tc-taprio to set Qbv:

2. The below link provides details for using tc-cbs to set Qav:

3. The below link provides details for using tc-flower to set Qci and ACL:

6.1.2.2 Tsntool

Tsntool is a tool to set the TSN capability of the Ethernet ports of TSN Endpoint and TSN switch. It is used on LS1028A platform. You should enable TSN, ENETC_TSN, and MSCC_FELIX_SWITCH_TSN to support tsntool configuration on LS1028A.

| Symbol: TSN [=y] |
| [*] Networking support ---> |
| Networking options ---> |
| [*] 802.1 Time-Sensitive Networking support |
| Symbol: ENETC_TSN [=y] && FSL_ENETC_PTP_CLOCK [=y] && FSL_ENETC_HW_TIMESTAMPING [=y] |
| Device Drivers ---> |
| [*] Network device support ---> |
| [*] Ethernet driver support ---> |
| [*] Freescale devices |
| <*> ENETC PF driver |
| <*> ENETC VF driver |
| <*> ENETC MIO IO driver |
| <*> ENETC PTP clock driver |
Enable PKTGEN in the kernel to use pktgen for testing using the commands below:

Symbol: NET_PKTGEN [=y]
[*] Networking support --->
   Networking options --->
   Network testing --->
   <*> Packet Generator (USE WITH CAUTION)

Refer to Tsntool User Manual for the details.

6.1.2.2.1 Tsntool User Manual

Tsntool is a tool to set the TSN capability of the Ethernet ports of TSN Endpoint and TSN switch. This document describes how to use tsntool for NXP’s LS1028ARDB hardware platform.

Note: Tsntool supports only the LS1028ARDB platform.

6.1.2.2.1.1 Getting the source code

Github of the tsntool code is mentioned below.

https://github.com/nxp-qoriq/tsntool

6.1.2.2.1.2 Tsntool commands

The Table 43 lists the TSN tool commands and their description.

Table 43. TSN tool commands and their description

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>help</td>
<td>Lists commands support</td>
</tr>
<tr>
<td>version</td>
<td>Shows software version</td>
</tr>
<tr>
<td>verbose</td>
<td>Debugs on/off for tsntool</td>
</tr>
<tr>
<td>quit</td>
<td>Quits prompt mode</td>
</tr>
<tr>
<td>qbvset</td>
<td>Sets time gate scheduling config for &lt;ifname&gt;</td>
</tr>
<tr>
<td>qbvget</td>
<td>Gets time scheduling entries for &lt;ifname&gt;</td>
</tr>
<tr>
<td>cbstreamidset</td>
<td>Sets stream identification table</td>
</tr>
<tr>
<td>cbstreamIDGET</td>
<td>Gets stream identification table and counters</td>
</tr>
<tr>
<td>qcisfiset</td>
<td>Sets stream filter instance</td>
</tr>
<tr>
<td>qcisfget</td>
<td>Gets stream filter instance</td>
</tr>
<tr>
<td>qcisgiset</td>
<td>Sets stream gate instance</td>
</tr>
<tr>
<td>qcisgget</td>
<td>Gets stream gate instance</td>
</tr>
</tbody>
</table>
### Table 43. TSN tool commands and their description...continued

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qcisficounterget</td>
<td>Gets stream filter counters</td>
</tr>
<tr>
<td>qcifmiset</td>
<td>Sets flow metering instance</td>
</tr>
<tr>
<td>qcifmiget</td>
<td>Gets flow metering instance</td>
</tr>
<tr>
<td>cbsset</td>
<td>Sets TCs credit-based shaper configuration</td>
</tr>
<tr>
<td>cbsget</td>
<td>Gets TCs credit-based shaper status</td>
</tr>
<tr>
<td>qbuset</td>
<td>Sets one 8-bits vector showing the preemptable traffic class</td>
</tr>
<tr>
<td>qbuset</td>
<td>Not supported</td>
</tr>
<tr>
<td>tsdset</td>
<td>Not supported</td>
</tr>
<tr>
<td>tsdget</td>
<td>Not supported</td>
</tr>
<tr>
<td>ctset</td>
<td>Sets cut through queue status (specific for ls1028 switch)</td>
</tr>
<tr>
<td>cbgen</td>
<td>Sets sequence generate configure (specific for ls1028 switch)</td>
</tr>
<tr>
<td>cbrec</td>
<td>Sets sequence recover configure (specific for ls1028 switch)</td>
</tr>
<tr>
<td>dscpset</td>
<td>Sets queues map to DSCP of QoS tag (specific for ls1028 switch)</td>
</tr>
<tr>
<td>sendpkt</td>
<td>Not supported</td>
</tr>
<tr>
<td>regtool</td>
<td>Registers read/write of bar0 of PFs (specific for ls1028 enetc)</td>
</tr>
<tr>
<td>ptpool</td>
<td>ptpool get/set ptp timestamp. Useful commands:</td>
</tr>
<tr>
<td></td>
<td>#get ptp0 clock time ptpool -g</td>
</tr>
<tr>
<td></td>
<td>#get ptp1 clock time ptpool -g -d /dev/ptp1</td>
</tr>
<tr>
<td>dscpset</td>
<td>Set queues map to DSCP of QoS tag (specific for ls1028 switch)</td>
</tr>
<tr>
<td>qcicapget</td>
<td>Gets max capability of the qci instance</td>
</tr>
<tr>
<td>tsncapget</td>
<td>Gets tsn capability of the device</td>
</tr>
</tbody>
</table>

#### 6.1.2.2.1.3 Tsntool commands and parameters

This section lists the tsntool commands along with the parameters and arguments, with which they can be used. Table 44 lists the qbvset parameter and its arguments.

### Table 44. qbvset

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swap0.</td>
</tr>
<tr>
<td>--entryfile &lt;filename&gt;</td>
<td>A file script to input gatelist format. It has the following arguments:</td>
</tr>
<tr>
<td></td>
<td># 'NUMBER' 'GATE_VALUE' 'TIME_LONG'</td>
</tr>
<tr>
<td></td>
<td>• NUMBER: # 't' or 'T' head. Plus entry number. Duplicate entry number will result in an error.</td>
</tr>
<tr>
<td></td>
<td>• GATE_VALUE: # format: xxxxxxxxb . # The MSB corresponds to traffic class 7. The LSB corresponds to traffic class 0. # A bit value of 0 indicates closed, whereas, a bit value of 1 indicates open.</td>
</tr>
<tr>
<td></td>
<td>• TIME_LONG: # nanoseconds. Do not input 0 time long.</td>
</tr>
<tr>
<td></td>
<td>t0 11101111b 10000 t1 11011111b 10000</td>
</tr>
<tr>
<td><strong>Note:</strong> Entryfile parameter must be set. If not set, there will be a vi text editor prompt, &quot;require to input the gate list&quot;.</td>
<td></td>
</tr>
<tr>
<td>--basetime &lt;value&gt;</td>
<td>AdminBaseTime</td>
</tr>
<tr>
<td></td>
<td>A 64-bit hex value means nanosecond until now.</td>
</tr>
</tbody>
</table>
### Table 44. qbvset...continued

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--cycletime &lt;value&gt;</td>
<td>AdminCycleTime</td>
</tr>
<tr>
<td>--cycleextend &lt;value&gt;</td>
<td>AdminCycleTimeExtension</td>
</tr>
<tr>
<td>--enable</td>
<td>--disable</td>
</tr>
<tr>
<td></td>
<td>• disable: disables the qbv for this port.</td>
</tr>
<tr>
<td></td>
<td>By default, the value is set to enable, if user does not provide any input.</td>
</tr>
<tr>
<td>--maxsdu &lt;value&gt;</td>
<td>queueMaxSDU</td>
</tr>
<tr>
<td>--initgate &lt;value&gt;</td>
<td>AdminGateStates</td>
</tr>
<tr>
<td>--configchange</td>
<td>ConfigChange. Default set to 1.</td>
</tr>
<tr>
<td>--configchangetime &lt;value&gt;</td>
<td>ConfigChangeTime</td>
</tr>
</tbody>
</table>

### Table 45. qbvget

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
</tbody>
</table>

### Table 46. cbstreamidset

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--enable</td>
<td>--disable</td>
</tr>
<tr>
<td></td>
<td>• disable: Disables the entry for this index.</td>
</tr>
<tr>
<td></td>
<td>By default, this field is set to enable if there is no enable or disable input.</td>
</tr>
<tr>
<td>--index &lt;value&gt;</td>
<td>Index entry number in this controller. Mandatory parameter.</td>
</tr>
<tr>
<td></td>
<td>This value corresponds to tsnStreamIdHandle on switch configuration.</td>
</tr>
<tr>
<td>--device &lt;string&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--streamhandle &lt;value&gt;</td>
<td>tsnStreamIdHandle</td>
</tr>
<tr>
<td>--infacoutput &lt;value&gt;</td>
<td>tsnStreamIdInFacOutputPortList</td>
</tr>
<tr>
<td>--outfacoutput &lt;value&gt;</td>
<td>tsnStreamIdOutFacOutputPortList</td>
</tr>
<tr>
<td>--infacinport &lt;value&gt;</td>
<td>tsnStreamIdInFacInputPortList</td>
</tr>
<tr>
<td>--outfacinport &lt;value&gt;</td>
<td>tsnStreamIdOutFacInputPortList</td>
</tr>
<tr>
<td>--nullstreamid</td>
<td>-</td>
</tr>
<tr>
<td>-sourcemacvid</td>
<td>-</td>
</tr>
<tr>
<td>-destmacvid</td>
<td>-</td>
</tr>
<tr>
<td>-ipstreamid</td>
<td>-</td>
</tr>
<tr>
<td>--nulldmac &lt;value&gt;</td>
<td>tsnCpeNullDownDestMac</td>
</tr>
</tbody>
</table>
### Table 46. cbstreamidset  ...continued

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--nulltagged &lt;value&gt;</td>
<td>tsnCpeNullDownTagged</td>
</tr>
<tr>
<td>--nullvid &lt;value&gt;</td>
<td>tsnCpeNullDownVlan</td>
</tr>
<tr>
<td>--sourcemac &lt;value&gt;</td>
<td>tsnCpeSmacVlanDownSrcMac</td>
</tr>
<tr>
<td>--sourcetagged &lt;value&gt;</td>
<td>tsnCpeSmacVlanDownTagged</td>
</tr>
<tr>
<td>--sourcevid &lt;value&gt;</td>
<td>tsnCpeSmacVlanDownVlan</td>
</tr>
</tbody>
</table>

### Table 47. cbstreamidget

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--index &lt;value&gt;</td>
<td>Index entry number in this controller. Mandatory to have.</td>
</tr>
</tbody>
</table>

### Table 48. qcisfiset

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
</tbody>
</table>
| --enable | --disable   | • enable: enable the entry for this index  
• disable: disable the entry for this index  
By default, this field is set to enable if there is no enable or disable input. |
| --maxsdu <value>       | Maximum SDU size.                                                           |
| --flowmeterid <value>  | Flow meter instance identifier index number.                               |
| --index <value>        | StreamFilterInstance.index entry number in this controller.               |
|                        | This value corresponds to tsnStreamIdHandle of cbstreamidset command on switch configuration. |
| --streamhandle <value> | StreamHandleSpec                                                            |
|                        | This value corresponds to tsnStreamIdHandle of cbstreamidset command.       |
| --priority <value>     | PrioritySpec                                                                |
| --gateid <value>       | StreamGateInstanceID                                                       |
| --oversizenable        | StreamBlockedDueToOversizeFrameEnable                                        |
| --oversize             | StreamBlockedDueToOversizeFrame                                            |

### Table 49. qcisfiget

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--index &lt;value&gt;</td>
<td>Index entry number in this controller. Mandatory to have.</td>
</tr>
</tbody>
</table>
### Table 50. qcisgiset

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--index &lt;value&gt;</td>
<td>Index entry number in this controller. Mandatory to have.</td>
</tr>
</tbody>
</table>
| --enable | --disable | • enable: enable the entry for this index. PSFPGateEnabled  
• disable: disable the entry for this index. By default, this field is set to enable if there is no enable or disable input. |
| --configchange                | configchange                                                                |
| --enblkinvrx                  | PSFPGateClosedDueToInvalidRxEnable                                         |
| --blkinvrx                    | PSFPGateClosedDueToInvalidRx                                               |
| --initgate                    | PSFPAdminGateStates                                                        |
| --initipv                     | AdminIPV                                                                    |
| --cycletime                   | Default not set. Get by gatelistfile.                                      |
| --cycletimeext                | PSFPAdminCycleTimeExtension                                                 |
| --basetime                    | PSFPAdminBaseTime                                                          |
|                               | A 64-bit hex value means nanosecond until now.                            |
|                               | OR a value input format as: Seconds.decimalSecond                          |
|                               | Example: 115.000125 means 115 seconds and 125 μs.                         |
| --gatelistfile                | PSFPAdminControlList. A file input the gate list: `NUMBER' 'GATE_VALUE' 'IPV' 'TIME_LONG' 'OCTET_MAX'  
• NUMBER: # 't' or 'T' head. Plus entry number. Duplicate entry number will result in an error.  
• GATE_VALUE: format: xb: The MSB corresponds to traffic class 7. The LSB corresponds to traffic class 0. A bit value of 0 indicates closed, A bit value of 1 indicates open.  
• IPV: # 0~7  
• TIME_LONG: in nanoseconds. Do not input time long as 0.  
• OCTET_MAX: The maximum number of octets that are permitted to pass the gate. If zero, there is no maximum.  
Example: t0 1b -1 50000 10 |

### Table 51. qcisgiget

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--index &lt;value&gt;</td>
<td>Index entry number in this controller. Mandatory to have.</td>
</tr>
</tbody>
</table>

### Table 52. qcifmiset

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--index &lt;value&gt;</td>
<td>Index entry number in this controller. Mandatory to have.</td>
</tr>
<tr>
<td>--disable</td>
<td>If not set disable, then to be set enable.</td>
</tr>
<tr>
<td>--cir &lt;value&gt;</td>
<td>cir. kbit/s.</td>
</tr>
</tbody>
</table>
### Table 52. qcifmiset...continued

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--cbs &lt;value&gt;</td>
<td>cbs. octets.</td>
</tr>
<tr>
<td>--eir &lt;value&gt;</td>
<td>eir.kbit/s.</td>
</tr>
<tr>
<td>--ebs &lt;value&gt;</td>
<td>ebs.octets.</td>
</tr>
<tr>
<td>--cf</td>
<td>cf. couple flag.</td>
</tr>
<tr>
<td>--cm</td>
<td>cm. color mode.</td>
</tr>
<tr>
<td>--dropyellow</td>
<td>drop yellow.</td>
</tr>
<tr>
<td>--markred_enable</td>
<td>mark red enable.</td>
</tr>
<tr>
<td>--markred</td>
<td>mark red.</td>
</tr>
</tbody>
</table>

### Table 53. qcifmiget parameter

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--index &lt;value&gt;</td>
<td>Index entry number in this controller. Mandatory to have.</td>
</tr>
</tbody>
</table>

### Table 54. qbuset parameter

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--preemptable &lt;value&gt;</td>
<td>8-bit hex value. Example: 0xfe The MS bit corresponds to traffic class 7. The LS bit to traffic class 0. A bit value of 0 indicates express. A bit value of 1 indicates preemptable.</td>
</tr>
</tbody>
</table>

### Table 55. cbsset command

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--tc &lt;value&gt;</td>
<td>Traffic class number.</td>
</tr>
<tr>
<td>--percentage &lt;value&gt;</td>
<td>Set percentage of tc limitation.</td>
</tr>
<tr>
<td>--all <a href="">tc-percent:tc-percent...</a></td>
<td>Not supported.</td>
</tr>
</tbody>
</table>

### Table 56. cbsget

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as eno0/swp0</td>
</tr>
<tr>
<td>--tc &lt;value&gt;</td>
<td>Traffic class number.</td>
</tr>
</tbody>
</table>
Table 57. `regtool`

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
</table>
| Usage: `regtool { pf number } { offset } [ data ]` | `pf number`: pf number for the pci resource to act on  
`offset`: offset into pci memory region to act upon  
`data`: data to be written |

Table 58. `ctset`

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--device &lt;ifname&gt;</code></td>
<td>An interface such as <code>swp0</code></td>
</tr>
<tr>
<td><code>--queue_stat &lt;value&gt;</code></td>
<td>Specifies which priority queues have to be processed in cut-through mode of operation. Bit 0 corresponds to priority 0, Bit 1 corresponds to priority 1 so-on.</td>
</tr>
</tbody>
</table>

Table 59. `cbgen`

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--device &lt;ifname&gt;</code></td>
<td>An interface such as <code>swp0</code></td>
</tr>
</tbody>
</table>
| `--index <value>` | Index entry number in this controller. Mandatory to have.  
This value corresponds to `tsnStreamIdHandle` of `cbstreamidset` command. |
| `--iport_mask <value>` | `INPUT_PORT_MASK`: If the packet is from input port belonging to this port mask, then it's a known stream and Sequence generation parameters can be applied |
| `--split_mask <value>` | `SPLIT_MASK`: Port mask used to add redundant paths (or ports). If split is enabled (`STREAM_SPLIT`) for a stream. This is OR'ed with the final port mask determined by the forwarding engine. |
| `--seq_len <value>` | `SEQ_SPACE_LOG2`: Minimum value is 1 and maximum value is 28.  
`tsnSeqRecSeqSpace = 2**SEQ_REC_SPACE_LOG2`  
For example, if this value is 12, then valid sequence numbers are from 0x0 to 0xFFF. |
| `--his_len <value>` | `SEQ_HISTORY_LEN`: Refer to `SEQ_HISTORY`, Min 1 and Max 32. |
| `--rtag_pop_en` | `REDTAG_POP`: If True, then the redundancy tag is popped by rewriter. |

Table 60. `cbrec`

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--device &lt;ifname&gt;</code></td>
<td>An interface such as <code>swp0</code></td>
</tr>
</tbody>
</table>
| `--index <value>` | Index entry number in this controller. Mandatory to have.  
This value corresponds to `tsnStreamIdHandle` of `cbstreamidset` command. |
| `--seq_len <value>` | `SEQ_SPACE_LOG2`: Min value is 1 and maximum value is 28.  
`tsnRecRecSeqSpace = 2**SEQ_REC_SPACE_LOG2`  
For example, if this value is 12, then valid sequence numbers are from 0x0 to 0xFFF. |
| `--his_len <value>` | `SEQ_HISTORY_LEN`: Refer to `SEQ_HISTORY`, Min 1 and Max 32. |
| `--rtag_pop_en` | `REDTAG_POP`: If True, then the redundancy tag is popped by rewriter. |
### Table 61. dscpset

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as swp0</td>
</tr>
<tr>
<td>--disable</td>
<td>Disables DSCP to traffic class for frames</td>
</tr>
<tr>
<td>--index</td>
<td>DSCP value</td>
</tr>
<tr>
<td>--cos</td>
<td>Priority number of queue which is mapped to</td>
</tr>
<tr>
<td>--dpl</td>
<td>Drop level which is mapped to</td>
</tr>
</tbody>
</table>

### Table 62. qcipget

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as swp0</td>
</tr>
</tbody>
</table>

### Table 63. tsncapget

<table>
<thead>
<tr>
<th>Parameter &lt;argument&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--device &lt;ifname&gt;</td>
<td>An interface such as swp0</td>
</tr>
</tbody>
</table>

### 6.1.2.2.1.4 Input tips

While providing the command input, user can use the following shortcut keys to make the input faster:

- **When a user inputs a command, use the **TAB** key to help list the related commands.**
  
  For example:
  ```
  tsntool> qbv
  Then press **TAB** key, to get all related qbv* start commands. If there is only one choice, it is filled as the whole command automatically.
  ```

- **When you want to input parameters and do not remember the parameter name, you can just input “--”. Then pressing **TAB** key displays all the parameters.**

- **If you input only half the parameter’s name, pressing the **TAB** key lists all the related names.**

- **History**: Press the up arrow “↑”. User gets the command history and can re-use the command.

### 6.1.2.2.1.5 Non-interactive mode

Tsntool also supports non-interactive mode.

For example:

In the interactive mode:

```
tsntool> qbuset --device eno0 --preemptable 0xfe
```

In non-interactive mode:

```
tsntool qbuset --device eno0 --preemptable 0xfe
```
6.1.2.3 Remote configuration using NETCONF/YANG

1. Overview

The NETCONF protocol defines a mechanism for device management and configuration retrieval and modification. It enables a client to adjust to the specific features of any network equipment by using a remote procedure call (RPC) paradigm and a system to expose device (server) capabilities.

YANG is a standards-based, extensible, hierarchical data modeling language. YANG is used to model the configuration and state data used by NETCONF operations, RPCs, and server event notifications.

2. Support for different platforms in Real-time Edge

<table>
<thead>
<tr>
<th>TSN offload</th>
<th>Real-time Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LS1028A</td>
</tr>
<tr>
<td></td>
<td>libtsn</td>
</tr>
<tr>
<td>802.1Qbv  (Time Aware Shaper)</td>
<td>Y</td>
</tr>
<tr>
<td>802.1Qbu/802.3br (Frame Preemption)</td>
<td>Y</td>
</tr>
<tr>
<td>802.1Qav (Credit Based Shaper)</td>
<td>-</td>
</tr>
<tr>
<td>802.1CB (Frame Replication and Elimination for Reliability)</td>
<td>-</td>
</tr>
<tr>
<td>802.1Qci (Per-Stream Filtering and Policing)</td>
<td>Y</td>
</tr>
<tr>
<td>IP config</td>
<td>Y</td>
</tr>
<tr>
<td>MAC config</td>
<td>Y</td>
</tr>
<tr>
<td>VLAN config</td>
<td>Y</td>
</tr>
</tbody>
</table>

3. Installation and configuration

Netopeer is a set of NETCONF tools built on the libnetconf library. The sysrepo-tsn (https://github.com/nxp-real-time-edge-sw/real-time-edge-sysrepo.git) helps to configure TSN features, including Qbv, Qbu, Qci, and stream identification via network, without logging in to device. For details of configuring TSN features via Netopeer, refer to NETCONF/YANG. Some application scenarios for tsn refer to Application scenarios.

6.1.2.4 Web-based configuration

6.1.2.4.1 Setting up web server

The Web UI allows the remote control of the YANG model and also get devices information by websockets. This demo is already added to tsntool in the folder tsntool/demos/cnc/.

In case user wants to setup the web server step by step, follow the below steps one by one:

1. Install related libraries: Suppose user is installing the demo on a Centos PC or Ubuntu PC as the WebServer. CNC demo requires python3 and related libraries: pyang, libnetconf, and libssh.
For Ubuntu:

$ sudo apt install -y libtool python-argparse libtool-bin python-sphinx libffi-dev
$ sudo apt install -y libxml2-dev libcurl4-openssl-dev xsltproc python-setuptools
$ sudo apt install -y zlibc-dev libssl-dev python-libxml2 libaugeas-dev
$ sudo apt install -y libreadline-dev python-dev pkg-config libxml2-dev
$ sudo apt install -y cmake openssh-server
$ sudo apt install -y python3-sphinx python3-setuptools python3-libxml2
$ sudo apt install -y libnss-mdns avahi-utils
$ pip3 install flask-restful
$ pip3 install websockets

For Centos 7.2:

$ sudo yum install libxml2-devel libxslt-devel openssl-devel libgcrypt dbus-devel
$ sudo yum install doxygen libevent readline.x86_64 ncurses-libs.x86_64
$ sudo yum install ncurses-devel.x86_64 libssh.x86_64 libssh2-devel.x86_64
$ sudo yum install libssh2.x86_64 libssh2-devel.x86_64
$ sudo yum install nss-mdns avahi avahi-tools

2. Install pyang

$ git clone https://github.com/mbj4668/pyang.git
$ cd pyang
$ git checkout b92b17718de53758c4c8a05b6818ea66fc0cd4d8 -b fornetconf1
$ sudo python setup.py install

3. Install libssh:

$ git clone https://git.libssh.org/projects/libssh.git
$ cd libssh
$ git checkout fe18ef279881b65434e3e44fc4743e4b1c7cb891 -b fornetconf1
$ mkdir build; cd build/
$ cmake ..
$ make
$ sudo make install

Note: There is a version issue for libssh installation on Ubuntu below version 16.04. Apt-get install libssh may get version 0.6.4. But libnetconf needs a version of 0.7.3 or later. Remove the default one and reinstall by downloading the source code and installing it manually.

4. Get tsntool source code on the web server PC:

```
git clone https://github.com/nxp-qoriq/tsntool
cd tsntool/demos/cnc/
```

5. Install libnetconf:

In the below command segments,
- PATH-to-tsntool is the path to the tsntool source code.

```
$ git clone https://github.com/CESNET/libnetconf.git
$ cd libnetconf
$ git checkout 8e934324e4b1e0ba6077b537e55636ed7c85aed -b fornetconf1
$ git am 0003-update-the-hostkeys-of-ssh-session-connection.patch
$ autoreconf --force --install
$ ./configure
$ make
```
6. Install python library:
   In the below command segments,
   • PATH-to-libnetconf is the path to the libnetconf source code.
   • PATH-to-tsntool is the path to the tsntool source code.

   $ cd PATH-to-libnetconf/

   The libnetconf needs to add two patches based on the below commit point to fix the demo python support. Ensure that the commit id is 313fdadd15427f7287801b92fe81ff84c08dd970.

   $ git checkout 313fdadd15427f7287801b92fe81ff84c08dd970 -b cnc-server
   $ cp PATH-to-tsntool/demos/cnc/*patch .
   $ git am 0001-lnctool-to-make-install-transapi-yang-model-proper.patch
   $ git am 0002-automatic-python3-authorizing-with-root-password-non.patch
   $ cd PATH-to-libnetconf/python
   $ python3 setup.py build; sudo python3 setup.py install

   Note:
   If rebuilding python lib, user need to remove the build folder by command rm build -rf before rebuilding. On the boards Real-time Edge supports, avahi-daemon and netopeer server are required. Remember to also add the netopeer2-server run at boards.

7. To start the web server on webserver PC, input the command below at shell into the folder: PATH-to-tsntool/demos/cnc/:

   sudo python3 cnc.py

8. Start topoagent server on the boards supported
   • Make sure the netopeer2-server run at boards (Not necessary for topology discovery).
   • Make sure the lldpd daemon is running at boards.
   • Make sure the avahi-daemon is running at boards.
   • Start the topology server on boards:

   ifconfig eno2 up
   ip link add name switch type bridge vlan_filtering 1
   ip link set switch up
   ip link set swp0 master switch && ip link set swp0 up
   ip link set swp1 master switch && ip link set swp1 up
   ip link set swp2 master switch && ip link set swp2 up
   ip link set swp3 master switch && ip link set swp3 up

   #Stop lldpd service.
   pkill lldpd
   #Start lldpd and limit interfaces to use. Use all ports except the control port.
   lldpd -1 swp0,swp1,swp2,swp3
   #If the hostname is not real-time-edge-$boardname, change to real-time-edge-$boardname.
   avahi-set-host-name real-time-edge-ls1028ardb
   cd /home/root/samples/cncdemo/
   python3 topoagent.py

9. Use the web browser to track the topology and configuration of the devices. Input the IP of web server with the port 8180 at browser. For example:

   http://10.193.20.147:8180

   Note:
**TSN configuration debug:**
- It is recommended to track the boards using tsntool to check the real tsn configuration for comparison.
- For tsn configuration, it is also recommended to track if the netopeer2-server is running at board or not.

**Limitations of Web UI are:**
- The server setup on a Centos PC or Ubuntu PC could be more compatible.
- Supports Qbv, Qbu, and Qci in current version.
- For Qci setting, Stream-gate entry should be set ahead of setting the Stream-filter as sysrepo required. Or else, user will get failure for setting Stream-filter without a stream gate id link to.
- The boards and the web server PC are required to be in the same IP domain since the bridge may block the probe frames.

### 6.1.2.4.2 Remote configuration

This section describes the steps for remote configuration.

- **Overview**
  The Web UI allows the remote control of the YANG model. The user can connect to the http server and input TSN parameters on the web UI. Click "Yes, confirm" button to send the parameters to the board as shown in Figure 48.

![Figure 48. A sample setup for remote configuration](image)

- **User Interface**
  Click the device displayed on the home page, and an interface description table appears. Click the interface to jump to the configuration page.

- **Qbv Configuration:** Selecting ‘qvb’ option setting displays the options as shown in Figure 49.
**Figure 49. Qbv Configuration**

- **Qbu Configuration**: Selecting 'qvb' option setting displays the options as shown in Figure 50.

**Figure 50. Qbu configuration**

- **Qci Configuration**: Selecting 'qci' option setting displays the options as shown in Figure 51.
Figure 51. Qci configuration

The qci interface allows user to select the configuration for "stream identify", "stream filter", "stream gate", and "flow metering".

Note:
1. Configure the "stream identify" first, then configure the "stream gate" and "flow metering", configure the "stream filter" at last.
2. "index" in "stream filter" configuration and "streamhandle" in "stream identify" should be the same value.
3. "flow meter index" in "stream filter" and "index" in "flow metering" should be the same value (63-246).

6.1.2.4.3 Dynamic remote configuration

The dynamic TSN configuration is used for the TSN configuration dynamically. Users do not need to log into each TSN node to specify the TSN parameters for TSN configuration. They only need to select the path, the base time, and then specify the cycle time. Then, the schedule mapping component calculates the TSN configuration parameters according to the user input and the path selected. The configuration parameters are delivered to each node by YANG models.

6.1.2.4.3.1 TSN working flow

This section provides an example of the TSN configuration working flow, which is described below:

After topology discovery and device registration, the network topology can be displayed over the web-browser. The user should select the nodes, specify the stream, input the timing requirement through the stream reservation component, and schedule configuration component. The results are passed down to the schedule mapping component to calculate the mapping from customer input to the TSN configuration. The configuration is instantiated using the YANG model and is delivered to different nodes for actual configuration.

The major components include:
- TSN network topology discovery
- Schedule mapping
- NETCONF/YANG configuration
• TSN Protocol Driver and TSN configuration
• Dashboard for stream management and customer input parse

Figure 52 shows the architecture diagram.

The Figure 52 illustrates the three layers of the TSN architecture. The first layer is the TSN network layer. The second layer is the service layer, which runs on the on-field controller/server. The third layer is an optional service that runs in the cloud or on-field server.

The TSN network layer includes TSN switches and endpoints that form the TSN network. For example, it includes the LS1028A TSN switch and TSN endpoints such as LS1028 ENETC TSN and i.MX 8M Plus TSN endpoint. The different components run on each of the nodes. For example, the topology discovery component collects the network topology, YANG model performs the TSN register configuration, and NETCONF server parses the YANG model for TSN configuration.

The second layer is the on-field controller layer. It is the server running on-field to host the services of the industrial board, topology discovery and schedule mapping.

The third layer runs on the cloud, which could host the services running on the on-field controller. This layer is an optional layer.

6.1.2.4.3.2 Topology discovery

The topology discovery component is used to discover network connections by running LLDP on each TSN network node. The connection information is delivered to topology discovery service running on the on-field server.
6.1.2.4.3.3 Path selection

Path selection implements an algorithm to select the path between the selected talker and listener. If there are multiple paths, the dashboard displays all paths and the user can select one of the paths for the stream. Set a different VLAN ID for the selected path, and the stream with this VID can flow in the path.

6.1.2.4.3.4 Path delay

Clock synchronization and path delay calculation are two prerequisites for schedule mapping. Clock synchronization uses gPTP to synchronize the clock of the system. The example described in this document uses linuxptp PMC tool to get the path delay.

Figure 53 shows a sample configuration to show the PMC running environment on LS1028ARDB boards.

6.1.2.4.3.5 Schedule mapping

The schedule mapping component is a critical component to convert the customer requirement to TSN register configuration. This component performs the following:
- Gets the user input and converting the input into TSN parameters.
- Gets the path and path delay from the link object of the NetworkGraph file.
- Gets the old TSN configuration for each node and calculates a new configuration to meet the user's requirements.

6.1.2.4.3.6 Dashboard configuration demo

The figure below shows the dashboard for the configuration demo.
Registering a stream

Click “Check Path” button, input the start device in “first device” input box, and end device in “Second device” input box. Then click the “submit” button, path is described in the Figure 55.

Figure 55. Stream Registration

Click “Register Stream” button, then select the path in path select. Fill VLAN ID, Stream ID, Priority, and then click “Add” button. The Figure 56 shows the output a stream table.

Figure 56. Registering a Stream
Configure stream identification

Click one stream ID in stream table, jump to stream configuration page. Select stream identify and fill information in input boxes. The stream MAC information and VLAN ID identify a stream according to the 802.1CB definition. This information is used by the PSFP configuration. Therefore, the stream identify page should be configured before configuring QcI and CFQ. Refer Figure 57.
Configure Qbv and Qci On Stream

Select Qbv, and then fill basetime, cycletime, and gate open time in the respective input textboxes. Select enable Qci button to configur both Qci gate control on input port and Qbv control list on the output port. The CNC server calculates the gate open time slot on each board and get a minimum time delay. Each path node tries to open gate with a minimum time delay. Refer to Figure 58.
Configuring CQF

The CQF configuration is based on the 802.1Qch definition to configure Qbv and Qci. The CQF configuration cannot be mixed with the previous Qbv configuration. In CQF configuration, the cycle time and gate open time for all streams should be the same, and cycle time must be an integer multiple of gate open time. Packets are delayed for a gate open time on each path node. Refer Figure 59.
6.1.3 TSN on i.MX 8DXL / i.MX 8M Plus / i.MX 93

The following sections describe TSN configuration on i.MX 8DXL, i.MX 8M Plus, or i.MX 93 hardware platforms.

6.1.3.1 Test environment

On i.MX 8M Plus EVK / i.MX 93 EVK platform, the interface name of ENET_QOS port which supports TSN is eth1. On i.MX 8DXL EVK / i.MX 93 9x9 LPDDR4 QSB, the interface name of ENET_QOS port which supports TSN is eth0.

Connect ENET_QOS port to the TestCenter to test TSN features. The commands in this section use the i.MX 8M Plus EVK platform as example:

Use the following command to check the TSN Ethernet device name:

```
# ls /sys/devices/platform/soc@0/30800000.bus/30bf0000.ethernet/net/ eth1
```

The Figure 60 shows the TSN test environment setup.
Note: TestCenter is a device used to capture streams from eth1 of i.MX8MP board. For this example, Spirent TestCenter is used to capture preemptable frames in Qbu test case.

6.1.3.2 Clock synchronization

To test 1588 synchronization on dwcMAC interfaces, use the following procedure:

1. Connect eth1 interfaces on two boards in a back-to-back manner. The Linux booting log is as follows:

   ```
   …
   pps pps0: new PPS source ptp0
   …
   ```

2. Configure the IP address using the command below:

   ```
   ifconfig eth1 192.168.3.1
   ```

3. Check PTP clock and time stamping capability:

   ```
   # ethtool -T eth1
   Time stamping parameters for eth1:
   Capabilities:
   hardware-transmit  (SOF_TIMESTAMPING_TX_HARDWARE)
   software-transmit  (SOF_TIMESTAMPING_TX_SOFTWARE)
   hardware-receive   (SOF_TIMESTAMPING_RX_HARDWARE)
   software-receive   (SOF_TIMESTAMPING_RX_SOFTWARE)
   software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
   hardware-raw-clock  (SOF_TIMESTAMPING_RAW_HARDWARE)
   PTP Hardware Clock: 1
   Hardware Transmit Timestamp Modes:
   off  (HWTSTAMP_TX_OFF)
   on   (HWTSTAMP_TX_ON)
   Hardware Receive Filter Modes:
   none  (HWTSTAMP_FILTER_NONE)
   all   (HWTSTAMP_FILTER_ALL)
   ptpv1-l4-event   (HWTSTAMP_FILTER_PTP_V1_L4_EVENT)
   ptpv1-l4-sync    (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
   ```
4. Run `ptp4l` on two boards:

```
ptp4l -i eth1 -p /dev/ptp1 -m -2
```

5. After running, one board is automatically selected as the master, and the slave board displays synchronization messages.

6. For 802.1AS testing, use the configuration file `gPTP.cfg` in `linuxptp` source. Run the below command on the boards, instead:

```
ptp4l -i eth1 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m
```

Or use GenAVB/TSN Stack with the following command: `avb.sh start`. Note that the configuration file `/etc/genavb/fgptp.cfg` is automatically used.

**Note:** i.MX 8M Plus current `dmac` driver (`eth1`) initializes few hardware functions while opening net device, including PTP initialization. Before that, the operations such as `ethtool` queries, and PTP operations might not work. So, the workaround is to do operations on the `eth1` and `PTP` of `dmac` only after "ifconfig eth1 up".

**Note:** If Qbu preemption is enabled on remote device and the PTP packets are sent as preemption frames, run clock synchronization using the `ptp4l` command along with the parameter `--hwts_filter=full`. For example:

```
ptp4l -i eth1 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m --hwts_filter=full
```

### 6.1.3.3 Qbv

**Note:** The Qbu frame preemption capability is enabled automatically if the link partner also supports frame preemption. This is achieved by LLDP protocol by which the link partner announces its support for the preemption capability via an Additional Ethernet Capabilities TLV in an LLDPDU addressed to the Nearest Bridge group address (see IEEE Std 802.1Q). So firstly user needs to check whether frame preemption is active by command `ethtool --show-frame-preemption eth1`. If so, user needs to disable frame preemption using command `ethtool --set-frame-preemption eth1 disabled` before configuring Qbv using `s (SetGateStates)` command in `sched-entry`.

1. Enable the `ptp` device, and get the current `ptp` time.

```
ptp4l -i eth1 -p /dev/ptp1 -m
#Get current time(seconds)
devmem2 0x30bf0b08
0x5E01F9B2
```

2. Get the basetime to be 2 minutes later.

```
#Basetime = (currentime + 120) * 1000000000 = 1577187882000000000
```

3. Set time schedule, open queue 1 in 100 µs and open queue 2 in 100 µs.

```
tc qdisc replace dev eth1 parent root handle 100 taprio \
```
4. Send two streams into queue 1 and queue 2.

/home/root/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 1 -s 1000 -n 0 -m 90:e2:ba:ff:ff:ff

5. Capture the streams on TestCenter, 100 µs queue 1 frames (length=1004) and 100 µs queue 2 frames (length=1504) will be got. Or if the Ethernet port is connected to another board, the frames can be captured on that board by using Linux tcpdump command as shown below:

tcpdump -i eth0 -e -n -t -xx -c 10000 -w tsn.pcap

Then Wireshark can be used to analyze the pcap file on host PC.

Note:
- More than one entry needs to be set on each tc taprio command.
- Use "devmem2 0x30bf0c58" to get Qbv status and check if qbv status is active. refer to MTL_EST_Status register.

6.1.3.4 Qbu

1. Using ethtool to enable Qbu on eth1, set queue 2 to be preemptable.

   ethtool --set-frame-preemption eth1 preemptible-queues-mask 0x04 min-frag-size 60

   Note: Once Qbu is enabled, queue 0 is always preemptable queue. To support preemption, MAC should have at least 1 queue designated as express queue.

   Note: On a back-to-back setup using two i.MX8M Plus EVK boards connected via eth1, Qbu should be enabled on eth1 of both boards.

2. Send two streams into queue 1 and queue 2.

   /home/root/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 1 -s 150 -n 0 -m 90:e2:ba:ff:ff:ff

3. Capture the mPacket on Spirent TestCenter. Users can observe that Q2 frames are preempted into fragments.

   Note: Spirent TestCenter can capture the preamble of mPacket. Refer to Section 99.3, "MAC Merge Packet (mPacket)" of IEEE standard for Ethernet 802.3-2018 for the mPacket format.

   a. Below is an example mPacket that contains an express packet, which has SMD value of 0xD5.
Frame 2230: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits)

User encapsulation not handled: DLT=147, check your Preferences->Protocols->DLT_USER

[Expert Info (Warning/Undecoded): User encapsulation not handled: DLT=147, check

Data (162 bytes)

Data: 555555555555555d50e2baaffff00e00c020b0108004500000808b500002011cb740000...

[Length: 162]

Figure 61. Sample mPacket that contains an express packet

b. Below is an example mPacket containing an initial fragment of a preemptable packet, which has SMD-S1 value of 0x4C.
Figure 62. Sample mPacket containing an initial fragment of a preemptable packet

c. Below is an example mPacket containing a continuation fragment of a preemptable packet, which has SMD-C1 value of 0x52, as well as frag_count value of 0xE6.
4. User can also check the below counter for the number of fragments transmitted.

```
ethtool -S eth1 | grep "mmc_tx_fpe_fragment_cntr"
```

5. Qbu combined with Qbv test

Once a queue is set to be a preemptable queue and the gate open/close is invalid in Qbv gate control list, the queue is considered as always "Open". Use Hold/Release to control all preemptable queues. When the GCL entry is set from Hold to Release, preemptable queues begin transmitting. When GCL entry is set from Release to Hold, preemptable queues are held.

```
tc qdisc replace dev eth1 parent root handle 100 taprio \
num_tc 5 map 0 1 2 3 4 queues 1@0 1@1 1@2 1@3 1@4 base-time 
1577187820000000000 \
sched-entry H  2 100000 \
sched-entry R  4 100000 flags 2
```

6.1.3.4.1 Preemption Verification

The preemption capability is enabled only if the link partner announces its support for the preemption capability via an Additional Ethernet Capabilities TLV in an LLDPDU addressed to the Nearest Bridge group address (see IEEE Std 802.1Q). The preemption capability is disabled if the MAC Merge sublayer receives indication of link failure.

Connect `eth1` of two boards back to back. Then, run below command to enable the hardware verification:

```
ethtool --set-frame-preemption eth1 fp on preemptible-queues-mask 0x04 min-frag-size 60
```
After that, make the link down and then up to enable the verify mPacket and response mPacket to be exchanged between two boards. Run below command to show the preemption status.

```
ethtool --show-frame-preemption eth1
```

Capture the LLDP frames on `eth1` port to check the Additional Ethernet Capabilities TLV.

### 6.1.3.5 Qav

1. Set a queue map handle.

```
tc qdisc add dev eth1 root handle 1: mqprio num_tc 5 map 0 1 2 3 4 queues 1@0 1@1 1@2 1@3 1@4 hw 0
```

2. Set bandwidth of queue 3 to be 20 Mbps.

```
tc qdisc replace dev eth1 parent 1:4 cbs locredit -1470 hicredit 30 sendslope -980000 idleslope 20000 offload 1
```

3. Send a stream into queue 3:

```
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eth1 -q 3 -s 500 -n 3000
```

4. Get the result, bandwidth is 19 Mbps.

```
WARN : Missing destination MAC address
WARN : Missing destination IP address
Running... ctrl^C to stop
Done
Result device: eth1
Params: count 3000  min_pkt_size: 500  max_pkt_size: 500
    frags: 0  delay: 0  clone_skb: 0  ifname: eth1
    flows: 0  flowlen: 0
    queue_map_min: 3  queue_map_max: 3
    dst_min: 198.18.0.42  dst_max:
    src_min:   src_max:
    udp_src_min: 9  udp_src_max: 109  udp_dst_min: 9  udp_dst_max: 9
    src_mac_count: 0  dst_mac_count: 0
    Flags: UDPSRC_RND  NO_TIMESTAMP  QUEUE_MAP_RND
Current:
    pkts-sofar: 3000  errors: 0
    started: 5631940023us  stopped: 5632560030us  idle: 79984us
    seq_num: 3001  cur_dst_mac_offset: 0  cur_src_mac_offset: 0
    cur_saddr: 0.0.0.0  cur_daddr: 198.18.0.42
    cur_udp_dst: 9  cur_udp_src: 41
    cur_queue_map: 3
    flows: 0
Result: OK: 620007(c540023+d79984) usec, 3000 (500byte,0frags)
4838pps 19Mb/sec (19352000bps) errors: 0
```

5. Set bandwidth of queue 4 to be 40 Mbps.

```
tc qdisc replace dev eth1 parent 1:5 cbs locredit -1440 hicredit 60 sendslope -960000 idleslope 40000 offload 1
```
6. Send a stream into queue 4 and get the result.

```
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eth1 -q 4 -s 500 -n 3000
```

WARN: Missing destination MAC address
WARN: Missing destination IP address
Running... ctrl^C to stop
Done
Result device: eth1
Params: count 3000 min_pkt_size: 500 max_pkt_size: 500
frags: 0 delay: 0 clone_skb: 0 ifname: eth1
flows: 0 flowlen: 0
queue_map_min: 4 queue_map_max: 4
dst_min: 198.18.0.42 dst_max:
src_min: src_max:
udp_src_min: 9 udp_src_max: 109 udp_dst_min: 9 udp_dst_max: 9
src_mac_count: 0 dst_mac_count: 0
Flags: UDP_SRC_RND NO_TIMESTAMP QUEUE_MAP_RND
Current:
pkts-sofar: 3000 errors: 0
started: 6113136017us stopped: 6113443758us idle: 38457us
seq_num: 3001 cur_dst_mac_offset: 0 cur_src_mac_offset: 0
cur_saddr: 0.0.0.0 cur_daddr: 198.18.0.42
cur_udp_dst: 9 cur_udp_src: 17
cur_queue_map: 4
flows: 0
Result: OK: 307741 (c269283+d38457) usec, 3000 (500byte, 0frags)
9748pps 38Mb/sec (38992000bps) errors: 0

7. Send two streams into queue 3 and queue 4 using the command below:

```
/home/root/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 3 -q 4 -s 1500 -n 0
```

8. Capture the streams on TestCenter, the frames sort by one Q3 frame and two Q4 frames.

### 6.1.4 TSN on LS1028A

The tsntool is an application configuration tool to configure the TSN capability on LS1028ARDB. The files `/usr/bin/tsntool` and `/usr/lib/libtsn.so` are located in the rootfs. Run `tsntool` to start the setting shell.

#### 6.1.4.1 TSN configuration on ENETC

The tsntool is an application configuration tool to configure the TSN capability. Users can find the files `/usr/bin/tsntool` and `/usr/lib/libtsn.so` in the rootfs. Run `tsntool` to start the setting shell. The following sections describe the TSN configuration examples on the ENETC Ethernet driver interfaces.

Before testing the ENETC TSN test cases, you must enable `mqprio` by using the command below:

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw 1
```

#### 6.1.4.1.1 Clock synchronization

To test 1588 synchronization on ENETC interfaces, use the following procedure:

1. Connect ENETC interfaces on two boards in a back-to-back manner. (For example, eno0 to eno0.)
The Linux booting log is as follows:

```
pps pps0: new PPS source ptp0
...```

2. Check PTP clock and timestamping capability:

```
# ethtool -T eno0
Time stamping parameters for eno0:
Capabilities:
    hardware-transmit (SOF_TIMESTAMPING_TX_HARDWARE)
    hardware-receive (SOF_TIMESTAMPING_RX_HARDWARE)
    hardware-raw-clock (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
    off (HWTSTAMP_TX_OFF)
    on (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
    none (HWTSTAMP_FILTER_NONE)
    all (HWTSTAMP_FILTER_ALL)
```

3. Configure the IP address and run `ptp4l` on two boards:

```
# ifconfig eno0 <ip_addr>
# ptp4l -i eno0 -p /dev/ptp0 -m
```

4. After running, one board would be automatically selected as the master, and the slave board would print synchronization messages.

5. For 802.1AS testing, just use the configuration file `gPTP.cfg` in linuxptp source. Run the below command on the boards, instead:

```
# ptp4l -i eno0 -p /dev/ptp0 -f /etc/ptp4l_cfg/gPTP.cfg -m
```

6.1.4.1.2 Qbv

This test includes the Basic Gates Closing test, Basetime test, and the Qbv performance test. These are described in the following sections.

6.1.4.1.2.1 Basic gates closing

The commands below describe the steps for closing the basic gates:

```
cat > qbv0.txt << EOF
t0 00000000b 20000
EOF
```

```
#Explanation:
# 'NUMBER'        :   t0
# 'GATE_VALUE'    :   00000000b
# 'TIME_LONG'     :   20000 ns
```

```
tsntool
tsntool> verbose
tsntool> qbvset --device eno0 --entryfile ./qbv0.txt
ethtool -S eno0
ping 192.168.0.2 -c 1  #Should not pass any frame since gates are all off.
```
6.1.4.1.2.2 Basetime test

Base on case 1 qbv1.txt gate list.

```bash
# create 1s gate
cat > qbv1.txt << EOF
t0 11111111b      10000
t1 00000000b      99990000
EOF

# ENETC Qbv basetime can be set any past time or future time.
# For the past time, hardware calculate by:
# effective-base-time = base-time + N x cycle-time
# where N is the smallest integer number of cycles such that effective-base-time
# >= now.
# If you want a future time, you can get current time by:
tsstool> ptptool -g
# Below example shows basetime start at 260.666 s (start of 1 January 1970):
tsstool> qbvset --device eno0 --entryfile qbv1.txt --basetime 260.666
tsstool> qbvget --device eno0 # User can check config_change time
tsstool> regtool 0 0x11a10 # Check pending status, 0x1 means time gate is working
# Waiting to change state, ping remote computer
ping 192.168.0.2 -A -s 1000
# The reply time will be about 100 ms
```

Since 10000 ns is the maximum limit for package size 1250 B.

```bash
ping 192.168.0.2 -c 1 -s 1300 # frame should not pass
```

6.1.4.1.2.3 Qbv performance test

Use the setup described in the Figure 64 for testing ENETC port0 (MAC0).
Note: TestCenter is a device used to capture streams from enetc0 of LS1028ARDB board. Users can use another board to capture streams by using tcpdump command and then use Wireshark to analyze it.

```bash
cat > qbv5.txt << EOF
t0 11111111b 1000000 t1 00000000b 1000000
EOF
qbvset --device eno0 --entryfile qbv5.txt
/home/root/samples/pktgen/pktgen_twoqueue.sh -i eno0 -q 3 -n 0
# The stream would get about half line rate
```

### 6.1.4.1.2.4 Using taprio Qdisc Setup Qbv

LS1028ARDB support the taprio qdisc to setup Qbv either. Below is an example setup.

```bash
# Qbv test do not require the mqprio setting.
# If mqprio is enabled, try to disable it by below command:
tc qdisc del dev eno0 root handle 1: mqprio
# Enable the Qbv for ENETC eno0 port
# Below command set eno0 with gate 0x01, means queue 0 open, the other queues
gate close.
tc qdisc replace dev eno0 parent root handle 100 taprio num_tc 8 map 0 1 2 3 4 5 6 7 queues 100 101 102 103 104 105 106 107 base-time 0 sched-entry S 01 300000 flags 0x2
# Ping through eno0 port should be ok
# Then close the gate queue 0. Open gate queue 1. The other queues gate close.
tc qdisc replace dev eno0 parent root handle 100 taprio num_tc 8 map 0 1 2 3 4 5 6 7 queues 100 101 102 103 104 105 106 107 base-time 0 sched-entry S 02 300000 flags 0x2
# Ping through eno0 port should be dropped
# Disable the Qbv for ENETC eno0 port as below
```
6.1.4.1.3 Qbu

- If user has two LS1028ARDB boards, then link the two eno0 ports back to back. In this case, the test does not need the switch to be set up. Users can omit the steps 2, 3, and 4 and just perform steps 1, 5, and 6.

- If user has only one board, user can set the frame path from eno0 to switch by linking enetc ports MAC0 - SWP0. The setup enables the switch SWP0 port-merging capability. Then enetc eno0 can show the preemption capability. Use the setup as shown in the for the Qbu test.

```
tc qdisc del dev eno0 parent root handle 100 taprio
```

Figure 65. Qbu test

Before linking the cable between ENETC port0 to SWP0, set up the switch up (refer the Switch configuration) and set IP for ENETC port0. To make sure the ENETC port0 is linked to SWP0, use the steps below:

1. Do not forget to enable the priority for each traffic class:

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw 1
```
2. Make sure link speed is 1 Gbps by using the command:

```bash
ethtool eno0
```

3. If it is not 1 Gbps, set it to 1 Gbps by using the command:

```bash
ethtool -s swp0 speed 1000 duplex full autoneg on
```

4. Set the switch to enable merge (or user can link to another merge capability port in another board):

```bash
devmem2 0x1fc100048 w 0x111 #DEV_GMII:MM_CONFIG:ENABLE_CONFIG
```

5. ENETC port setting set and frame preemption test:

```bash
ip link set eno0 address 90:e2:ba:ff:ff:ff
tsn tool qbuset --device eno0 --preemptable 0xfe
/home/root/samples/pktgen/pktgen_twoqueue.sh -i eno0 -q 0 -s 100 -n 20000 -m 90:e2:ba:ff:ff:ff
```

pktgen would flood frames on TC0 and TC1.

6. Check the TX merge counter, if it has a non-zero value, it indicates that the Qbu is working.

```bash
tsn tool regtool 0 0x11f18
```

**Note:** 0x11f18 counting the merge frame count:

```
0x11f18 Port MAC Merge Fragment Count TX Register (MAC_MERGE_MMFCCTXR)
```

LS1028ARDB also supports ethtool setup for preemption as in the example below:

```bash
ethtool --set-frame-preemption eno0 preemptible-queues-mask 0xfe
```

This implies that we can get same result by using TC0 to pass express MAC and TC1~TC7 to pass preemptable MAC.

The ENETC also supports preemption verify. Use two boards to test preemption verification on eno0. Refer to Section 6.1.3.4.1.

6.1.4.1 Qci

Use the following as the background setting:

- Set eno0 MAC address

```bash
ip link set eno0 address 10:00:80:00:00:00
```

Opposite port MAC address 99:aa:bb:cc:dd:ee as frame provider as example.

- Use the figure below as the hardware setup.
6.1.4.1.4.1 Test SFI No Streamhandle

Qci PSFP can work for the streams without stream identify module, which are the streams without MAC address and vid filter. Such kind of filter setting always sets a larger index number stream for filter entry. The frames that are not filtered then flow into this stream filter entry.

The below example tests no streamhandle in a stream filter, set on stream filter entry index 2 with a gate stream entry id 2. Then none stream identifies frames would flow into the stream filter entry index 2 then pass the gate entry index 2, as shown in the following example:

```
tsntool> qcisfiset --device eno0 --index 2 --gateid 2
```

- Streams no streamhandle should pass this filter.

```
tsntool> qcisfiget --device eno0 --index 2
```
• Send a frame from the opposite device port (ping for example).

```bash
tsntool> qcisfiget --device eno0 --index 2
```

• Set Stream Gate entry 2

```bash
tsntool> qcisgiset --device eno0 --index 2 --initgate 1
```

• Send a frame from the opposite device port.

```bash
tsntool> qcisfiget --device eno0 --index 2
```

• Set Stream Gate entry 2, gate closes permanently.

```bash
tsntool> qcisgiset --device eno0 --index 2 --initgate 0
```

• Send a frame from the opposite device port.

```bash
tsntool> qcisfiget --device eno0 --index 2
```

#The result should look like below:

<table>
<thead>
<tr>
<th>match</th>
<th>pass</th>
<th>gate_drop</th>
<th>sdu_pass</th>
<th>sdu_drop</th>
<th>red</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

6.1.4.1.4.2 Testing null stream identify entry

Null stream identify in stream identify module means trying to filter using destination MAC address and vlan id.

Following steps show the stream identify entry index 1 set with filtering destination mac address as 10:00:80:00:00:00 and vlan id ignored (with or without vland id). Then stream filter is set on the entry index 1 with stream gate index entry id 1.

1. Set main stream by closing gate.
2. Set Stream identify Null stream identify entry 1.

```bash
tsntool> cbstreamidset --device eno0 --index 1 --nullstreamid --nullldmac 0x000000800010 --nulltagged 3 --nullvid 10 --streamhandle 100
```


```bash
tsntool> cbstreamidget --device eno0 --index 1
```

4. Set Stream filer entry 1 with stream gate entry id 1.

```bash
tsntool> qcisfiset --device eno0 --streamhandle 100 --index 1 --gateid 1
```

5. Set Stream Gate entry 1, keep gate state close (all frames dropped, return directly if ask user for editing gate list).

```bash
tsntool> qcisgiset --device eno0 --index 1 --initgate 0
```

6. Send one frame from the opposite device port should pass to the close gate entry id 1.

```bash
tsntool> qcisfiget --device eno0 --index 1
```

7. The result should look like the output below:

<table>
<thead>
<tr>
<th>match</th>
<th>pass</th>
<th>gate_drop</th>
<th>sdu_pass</th>
<th>sdu_drop</th>
<th>red</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

6.1.4.1.4.3 Testing source stream identify entry

Source stream identify means stream identify the frames by the source mac address and vlan id.

Use the following steps for this test:

1. Keep Stream Filter entry 1 and Stream gate entry 1.
2. Add stream2 in opposite device port: SMAC is 66:55:44:33:22:11 DMAC:20:00:80:00:00:00 (Not with destination mac address 10:00:80:00:00:00 which stream identify entry index 1 is filtering that dmac address)
3. Set Stream identify Source stream identify entry 3

```
tsntool> cbstreamidset --device eno0 --index 3 --sourcemacvid --sourcemac 0x112233445566 --sourcetagged 3 --sourcevid 20 --streamhandle 100
```

4. Send frame from opposite device port. The frame passes to stream filter index 1.

```
tsntool> qcisfiget --device eno0 --index 1
```

6.1.4.1.4.4 SGI stream gate list

Use the command below for this test:

```
cat > sgi1.txt << EOF
t0 0b -1 100000000 0
t1 1b -1 100000000 0
EOF
tsntool> qcisfiset --device eno0 --index 2 --gateid 2
tsntool> qcisgiset --device eno0 --index 2 --initgate 1 --gatelistfile sgi1.txt #flooding frame size 64bytes from opposite device port.(iperf or netperf as example)
tsntool> qcisfiget --device eno0 --index 2
```

Check the frames dropped and passed, they should be the same since stream gate list is setting 100ms open and 100ms close periodically.

6.1.4.1.4.5 FMI test

Only send green color frames (normally it is the TCI bit value in 802.1Q tag). Flooding the stream against the eno0 port speed to 10000 kbsp/s:

```
tsntool> qcisfiset --device eno0 --index 2 --gateid 2
ntsntool> qcfmiset --device eno0 --index 2 --cm --cf --cbs 1500 --cir 5000 --ebs 1500 --eir 5000
```

The 'cm' parameter set color mode enable means frames to separate green frames and yellow frames judged by the TCI bit in frame. Or else, any frames are green frames.

The 'cf' parameter sets the coupling flag enable. When CF is set to 0, the frames that are declared yellow are bound by EIR. When CF is set to 1, the frames that are declared Yellow are bound by CIR + EIR, depending on volume of the offered frames that are declared Green.

After the above commands are setup, since green frames are not larger than EIR + CIR 10 Mbit/s. So the green frame would not be dropped.
The below setting shows the dropped frames:

```sh
tsntool> qcifmiset --device eno0 --index 2 --cm --cf --cbs 1500 --cir 5000 --ebs 1500 --eir 2000
```

This case makes the green frames pass 5 Mbit/s in CIR, then it pass to the EIR space. However, EIR is 2 Mbit/s, so total EIR + CIR 7 Mbit/s still do not qualify the total 10 Mbit/s bandwidth. So green frame would be dropped part.

To get information of color frame counters showing at application layer, use the code as in the below example:

```sh
tsntool> qcifmiget --device eno0 --index 2
```

<table>
<thead>
<tr>
<th>bytecount</th>
<th>drop</th>
<th>dr0_green</th>
<th>dr1_green</th>
<th>dr2_yellow</th>
<th>remark_yellow</th>
<th>dr3_red</th>
<th>remark_red</th>
</tr>
</thead>
<tbody>
<tr>
<td>1c89</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>cir</th>
<th>cbs</th>
<th>eir</th>
<th>ebs</th>
<th>couple flag</th>
<th>color mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>c34c</td>
<td>5dc</td>
<td>4c4b3c</td>
<td>5dc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.1.4.1.5 Qav

6.1.4.1.5.1 Using tsntool

The following figure illustrates the hardware setup diagram for the Qav test.
Figure 67. Qav test setup

Note: TestCenter is a device to capture streams from enetc0 of LS1028ARDB board. Users can also use another board to capture streams by using tcpdump command, and use Wireshark network protocol analyzer to analyze results.

0. Ensure to enable the priority for each traffic class:

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw 1
```

1. Run the following commands:

```
tsntool cbsset --device eno0 --tc 7 --percentage 60
```

2. Check each queue bandwidth (pktgen requires enabling NET_PKTGEN in kernel)

```
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eno0 -q 7 -s 500 -n 30000
```
Wait a few seconds later to check the result. It should get about 60% percentage line rate.

```
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eno0 -q 6 -s 500 -n 30000
```

Wait a few seconds later to check the result. It should get about 20% percentage line rate.

### 6.1.4.1.5.2 Using CBS Qdisc to setup Qav

LS1028a supports the CBS qdisc to setup Credit-based Shaper. Below commands set CBS with 100 Mbit/s for queue 7 and 300 Mbit/s for queue 6.

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw 1
tc qdisc replace dev eno0 parent 1:8 cbs locredit -1350 hicredit 150 sendslope -900000 idleslope 100000 offload 1
tc qdisc replace dev eno0 parent 1:7 cbs locredit -1050 hicredit 950 sendslope -700000 idleslope 300000 offload 1
```

```
# Try to flood stream here (require kernel enable NET_PKTGEN)
/home/root/samples/pktgen/pktgen_sample01_simple.sh -I eno0 -q 7 -s 500 -n 20000
/home/root/samples/pktgen/pktgen_sample01_simple.sh -I eno0 -q 6 -s 500 -n 20000
tc qdisc del dev eno0 parent 1:7 cbs
tc qdisc del dev eno0 parent 1:8 cbs
```

### 6.1.4.2 TSN configuration on Felix switch

The following sections describe examples for the basic configuration of TSN switch.
6.1.4.2.1 Switch configuration

Use the following commands to configure bridge on LS1028ARDB:

```
ls /sys/bus/pci/devices/0000:00:00.5/net/
```

Get switch device interfaces for swp0, swp1, swp2 and swp3 as shown below:

```
ip link set eno2 up
ip link add name switch type bridge vlan_filtering 1
ip link set switch up
ip link set swp0 master switch && ip link set swp0 up
ip link set swp1 master switch && ip link set swp1 up
ip link set swp2 master switch && ip link set swp2 up
ip link set swp3 master switch && ip link set swp3 up
```

6.1.4.2.2 Linuxptp test

To run PTP clock synchronization cases on TSN switch in:

- 4.3.5 Quick Start for IEEE 1588
- 4.3.6 Quick Start for IEEE 802.1AS

There are additional configurations of PTP packets trapping besides basic L2 switch mode configuration of “4.1.4.2.1 Switch configuration”. An available IP should be configured on bridge, and don’t configure IP on swpX interfaces.

```
$ ./switch-ptp-trap.sh
```
$ ifconfig switch <ip_address>

switch-ptp-trap.sh

```bash
# swp0, trap ptp
tc qdisc add dev swp0 clsact
tc filter add dev swp0 ingress chain 0 pref 49152 flower skip_sw action goto chain 10000
tc filter add dev swp0 ingress chain 10000 pref 49152 flower skip_sw action goto chain 11000
tc filter add dev swp0 ingress chain 11000 pref 49152 flower skip_sw action goto chain 12000
tc filter add dev swp0 ingress chain 12000 pref 49152 flower skip_sw action goto chain 20000
tc filter add dev swp0 ingress chain 20000 pref 49152 flower skip_sw action goto chain 21000
tc filter add dev swp0 ingress chain 21000 pref 49152 flower skip_sw action goto chain 30000
tc filter add dev swp0 ingress chain 30000 pref 49152 flower skip_sw action goto chain 20000
  protocol 0x88f7 flower skip_sw action trap action goto chain 21000
  tc filter add dev swp0 ingress chain 20000 protocol ip flower skip_sw dst_ip 224.0.1.129 action trap action goto chain 21000
  tc filter add dev swp0 ingress chain 20000 protocol ip flower skip_sw dst_ip 224.0.0.107 action trap action goto chain 21000

# swp1, trap ptp
tc qdisc add dev swp1 clsact
tc filter add dev swp1 ingress chain 0 pref 49152 flower skip_sw action goto chain 10000
tc filter add dev swp1 ingress chain 10000 pref 49152 flower skip_sw action goto chain 11000
tc filter add dev swp1 ingress chain 11000 pref 49152 flower skip_sw action goto chain 12000
tc filter add dev swp1 ingress chain 12000 pref 49152 flower skip_sw action goto chain 20000
tc filter add dev swp1 ingress chain 20000 pref 49152 flower skip_sw action goto chain 21000
tc filter add dev swp1 ingress chain 21000 pref 49152 flower skip_sw action goto chain 30000
tc filter add dev swp1 ingress chain 30000 pref 49152 flower skip_sw action goto chain 20000
  protocol 0x88f7 flower skip_sw action trap action goto chain 21000
  tc filter add dev swp1 ingress chain 20000 protocol ip flower skip_sw dst_ip 224.0.1.129 action trap action goto chain 21000
  tc filter add dev swp1 ingress chain 20000 protocol ip flower skip_sw dst_ip 224.0.0.107 action trap action goto chain 21000

# swp2, trap ptp
tc qdisc add dev swp2 clsact
tc filter add dev swp2 ingress chain 0 pref 49152 flower skip_sw action goto chain 10000
tc filter add dev swp2 ingress chain 10000 pref 49152 flower skip_sw action goto chain 11000
tc filter add dev swp2 ingress chain 11000 pref 49152 flower skip_sw action goto chain 12000
tc filter add dev swp2 ingress chain 12000 pref 49152 flower skip_sw action goto chain 20000
tc filter add dev swp2 ingress chain 20000 pref 49152 flower skip_sw action goto chain 21000
```
tc filter add dev swp2 ingress chain 21000 pref 49152 flower skip_sw action goto
chain 30000
tc filter add dev swp2 ingress chain 20000 protocol 0x88f7 flower skip_sw action
trap action goto chain 21000
tc filter add dev swp2 ingress chain 20000 protocol ip flower skip_sw dst_ip
224.0.1.129 action trap action goto chain 21000
tc filter add dev swp2 ingress chain 20000 protocol ip flower skip_sw dst_ip
224.0.0.107 action trap action goto chain 21000

# swp3, trap ptp
tc qdisc add dev swp3 clsact
tc filter add dev swp3 ingress chain 0 pref 49152 flower skip_sw action goto
chain 10000
tc filter add dev swp3 ingress chain 10000 pref 49152 flower skip_sw action goto
chain 11000
tc filter add dev swp3 ingress chain 11000 pref 49152 flower skip_sw action goto
chain 12000
tc filter add dev swp3 ingress chain 12000 pref 49152 flower skip_sw action goto
chain 20000
tc filter add dev swp3 ingress chain 20000 pref 49152 flower skip_sw action goto
chain 21000
tc filter add dev swp3 ingress chain 21000 pref 49152 flower skip_sw action goto
chain 30000
tc filter add dev swp3 ingress chain 21000 protocol 0x88f7 flower skip_sw action
trap action goto chain 21000
tc filter add dev swp3 ingress chain 21000 protocol ip flower skip_sw dst_ip
224.0.1.129 action trap action goto chain 21000
tc filter add dev swp3 ingress chain 21000 protocol ip flower skip_sw dst_ip
224.0.0.107 action trap action goto chain 21000

# ebtables, route ptp, not bridge
ebtables --table broute --append BROUTING --protocol 0x88F7 --jump DROP
ebtables --table broute --append BROUTING --protocol 0x0800 --ip-protocol udp --
ip-destination-port 320 --jump DROP
ebtables --table broute --append BROUTING --protocol 0x0800 --ip-protocol udp --
ip-destination-port 319 --jump DROP

6.1.4.2.3 Qbv test setup for LS1028ARDB

The following figure describes the setup for Qbv test on LS1028ARDB.
Reserve buffer for each queue on ingress and egress port to avoid resource depletion when Qbv gate is closed.

```bash
ingressport=0
egressport=1
for tc in {0..7}; do {
    devlink sb tc bind set pci/0000:00:00.5/$ingressport sb 0 tc $tc type ingress pool 0 th 3000
    devlink sb tc bind set pci/0000:00:00.5/$ingressport sb 1 tc $tc type ingress pool 0 th 10
    devlink sb tc bind set pci/0000:00:00.5/$egressport sb 0 tc $tc type egress pool 1 th 3000
    devlink sb tc bind set pci/0000:00:00.5/$egressport sb 1 tc $tc type egress pool 1 th 10
}
done
```
6.1.4.2.3.1 Tsntool usage

Closing basic gates

Use the set of commands below for basic gate closing.

```
echo "t0 00000000b 20000" > qbv0.txt
#Explanation:
# 'NUMBER'      :  t0
# 'GATE_VALUE'  :  00000000b
# 'TIME_LONG'   :  20000 ns
./tsntool
tsntool> verbose
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt
#Send one broadcast frame to swp0 from TestCenter.
ethtool -S swp1
#Should not get any frame from swp1 on TestCenter.
echo "t0 11111111b 20000" > qbv0.txt
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt
#Send one broadcast frame to swp0 on TestCenter.
ethtool -S swp1
#Should get one frame from swp1 on TestCenter.
```

Basetime test

For the basetime test, first get the current time in seconds:

```
#Get current time:
tsntool> ptptool -g -d /dev/ptp1
#add some seconds, for example user gets 200.666 time clock, then set 260.666 as result
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt --basetime 260.666
#Send one broadcast frame to swp0 on the TestCenter.
#Frame could not pass swp1 until time offset.
```

QBv performance test

Use the following commands for the QBv performance test:

```
cat > qbv5.txt << EOF
t0 11111111b 1000000
11 00000000b 1000000
EOF
qbvset --device swp1 --entryfile qbv5.txt
```

Send 1G rate stream to swp0 on TestCenter.
The stream would get about half line rate from swp1.

**Note:** Each entry time must be larger than guard band, the guard band is set by "--maxsdu", if it's not set, use default 1518Bytes, the least entry time is (1518*8)/1G≈12us.
6.1.4.2.3.2 Tc-taprio usage

LS1028ARDB supports the tarprio qdisc to setup Qbv either. Below is an example setup.

1. Enable the Qbv for swp1 port, set queue 1 gate open, set circle time to be 300 µs.

```
tc qdisc replace dev swp1 parent root handle 100 taprio num_tc 8 map 0 1 2 3 4 5 6 7
  queues 1 0 1 0 2 1 0 3 1 0 4 1 0 5 1 0 6 1 0 7 base-time 0 sched-entry S 0 2
300000 flags 0x2
```

**Note:** Since the hardware can only use PCP, DSCP or other methods to classify QoS, it cannot map QoS to different hardware queues. mqprio is not implemented in the felix driver, so "map 0 1 2 3 4 5 6 7" in the tc-taprio command is invalid.

**Note:** Tc-taprio uses default port max SDU(1518B) as guard band value. Each entry time must be larger than guard band(1518*8/1G≈12us).

2. Send one frame with PCP=1 in vlan tag to swp0 from TestCenter, so as to capture the frame from swp1.
3. Send one frame with PCP=2 in vlan tag to swp0 from TestCenter, gate is closed and the frame from swp1 cannot be captured.
4. Disable the Qbv for swp1 port as below:

```
tc qdisc del dev swp1 parent root handle 100 taprio
```

6.1.4.2.4 Qbu

The figure below illustrates the setup for performing the Qbu test using the TSN switch.
6.1.4.2.4.1 Tsntool usage

1. Disable the Cut-through mode before enabling preemption on switch ports.
   
   ```
   # tsntool> ctset --device swp3 --queue_stat 0x0
   ```

2. Set queue 1 to be preemptable. There are two ways to set preemptable queues, users can choose tsntool or ethtool to set it.

   ```
   #tsntool command to set preemptable queues:
   tsntool> qbuset --device swp3 --preemptable 0x02
   ```

3. Send two streams from TestCenter, set packet size to be 1500 Byte and bandwidth to be 1G. Now, check the number of additional mPackets transmitted by PMAC using the command below:

   ```
   ethtool -S swp3 | grep tx_merge.fragments
   ```

4. Follow the steps below to perform Qbu combined with Qbv test.
   Set queue 0 gate open 20 µs, queue 1 gate open to 20 µs.

   ```
   cat > qbv0.txt << EOF
t0 00000001b 200000
t1 00000010b 200000
EOF
tqbuset --device swp3 --entryfile qbv0.txt
   ```

   Send two streams from TestCenter. Observe that packets in queue 1 are preempted when gate 1 is closed.
6.1.4.2.4.2 Ethtool usage

1. Set queue 1 to be preemptable. There are two ways to set preemptable queues, users can choose tsntool or ethtool to set it.

   ```
   # ethtool command to set preemptable queues:
   ethtool --set-frame-preemption swp3 preemptible-queues-mask 0x02 min-frag-size 124
   ```

   **Explanation:**
   - `preemptible-queues-mask`: An 8-bit vector that specifies preemptable queues within the 8 priorities (with bit-0 for priority-0 and bit-7 for priority-7).
   - `min-frag-size`: specifies the least frame bytes that have been transmitted in the fragment. The minimum non-final fragment size is 64, 128, 192, or 256 octets (include 4 Bytes fragment header).

2. Send two streams from TestCenter. Set packet size to be 1500 Bytes and bandwidth to be 1 G. Now, check the number of additional mPackets transmitted by PMAC:

   ```
   ethtool -S swp3 | grep tx_merge_fragments
   ```

3. Qbu combined with Qbv test.
   Set queue 0 gate open 20 µs, queue 1 gate open 20 µs.

   ```
   tc qdisc replace dev swp3 parent root handle 100 taprio num_tc 8 map 0 1 2 3 4 5 6 7 \ 
   queues 100 101 102 103 104 105 106 107 base-time 0 \ 
   sched-entry S 01 200000 \ 
   sched-entry S 02 200000 flags 0x2
   ```

   Send two streams from TestCenter. Note that packets in queue 1 are preempted when gate 1 closed.

4. The felix switch port also supports preemption verify. Use two boards to test preemption verification on swp0-3. Refer to **4.1.3.4.1 Preemption verify**.

6.1.4.2.5 Qci

The figure below illustrates the Qci test case setup.
6.1.4.2.5.1 Tsntool usage

Stream identification

Use the following commands for stream identification:

1. Set a stream to swp0 on TestCenter. Edit the stream, set the destination MAC as: 00:01:83:fe:12:01, Vlan ID : 1

2. Add the MAC to MAC table on LS1028a. (This step is not needed if the mac is already learned on port)

   bridge fdb add 00:01:83:fe:12:01 dev swp1 vlan 1 master static

3. Use the destination MAC as: 00:01:83:fe:12:01, vlan ID : 1 to set the stream identification on LS1028A.

   tsntool> cbstreamidset --device swp1 --nullstreamid --index 1 --nullidmac 0x000183fe1201 --nullvid 1 --streamhandle 1
Explanation:

- **device**: set the device port which is the stream forwarded to. If the \{destmac, VID\} is already learned by switch, switch will not care device port.
- **nulltagged**: switch only support nulltagged=1 mode, so there is no need to set it.
- **nullvid**: Use "bridge vlan show" to see the ingress VID of switch port.

```
tsntool> qcisfiset --device swp0 --index 1 --streamhandle 1 --gateid 1 --priority 0 --flowmeterid 68
```

Explanation:

- **device**: can be any one of switch ports.
- **index**: value is the same as streamhandle of cbstreamidset.
- **streamhandle**: value is the same as streamhandle of cbstreamidset.
- **flowmeterid**: PSFP Policer id, ranges from 63 to 383.

4. Send one frame, then check the frames.

```
ethtool -S swp1
ethtool -S swp2
```

Only swp1 can get the frame.

5. Use the following command to check and debug the stream identification status.

```
qcisfiget --device swp0 --index 1
```

**Note:** The parameter **streamhandle** is the same as **index** in stream filter set, we use **streamhandle** as SFID to identify the stream, and use **index** to set stream filter table entry.

Stream gate control

1. Use the following commands for stream gate control:

```
echo "t0 1b 3 50000 200" > sgi.txt
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile sgi.txt --basetime 0x0
```

Explanation:

- **device**: can be any one of switch ports.
- **index**: gateid
- **basetime**: It is the same as Qbv set.

2. Send one frame on TestCenter.

```
ethtool -S swp1
```

Note that the frame could pass, and green_prio_3 has increased.

3. Now run the following commands:

```
echo "t0 0b 3 50000 200" > sgi.txt
ntsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile sgi.txt --basetime 0x0
```

4. Next, send one frame on TestCenter.

```
ethtool -S swp1
```

Note that the frame could not pass.
SFI maxSDU test

Disable the cut-through mode on swp0 and swp1:

```
tsntool> ctset --device swp0 --queue_stat 0
tsntool> ctset --device swp1 --queue_stat 0
```

Use the following command to run this test:

```
tsntool> qcisfiset --device swp0 --index 1 --gateid 1 --priority 0 --flowmeterid 68 --maxsdu 200
```

Now, send one frame (frame size > 200) on TestCenter.

```
ethtool -S swp1
```

Users can observe that the frame could not pass.

FMI test

Use the following set of commands for the FMI test.

1. Reserve buffer for each queue on ingress port to receive yellow frames(dp=1) in switch.

   ```
ingressport=0
   for tc in {0..7}; do {
       devlink sb tc bind set pci/0000:00:00.5/$ingressport sb 0 tc $tc type
       ingress pool 0 th 3000
       devlink sb tc bind set pci/0000:00:00.5/$ingressport sb 1 tc $tc type
       ingress pool 0 th 10
   }
done
   ```

2. Run the command:

   ```
tsntool> qcifmiset --device swp0 --index 68 --cir 100000 --cbs 4000 --ebs 4000 --eir 100000
   ```

   **Note:**
   - The 'device' in above command can be any one of the switch ports.
   - The index of qcifmiset must be the same as flowmeterid of qcisfiset.

3. Now, send one stream (rate = 100M) on TestCenter.

   ```
ethtool -S swp0
   ```

   Note that all frames pass and get all green frames.

4. Now, send one stream (rate = 200M) on TestCenter.

   ```
ethtool -S swp0
   ```

   Observe that all frames pass and get green and yellow frames.

5. Send one stream (rate = 300M) on TestCenter.

   ```
ethtool -S swp0
   ```

   Note that not all frames could pass and get green, yellow, and red frames.

6. Send one yellow stream (rate = 100M) on TestCenter.

   ```
ethtool -S swp0
   ```
All frames pass and get all yellow frames.

7. Send one yellow stream (rate = 200M) on TestCenter.
   
   ```bash
   ethtool -S swp0
   ```

   Note that not all frames could pass and get yellow and red frames.

8. Test cf mode.
   
   ```bash
   tsntool> qcifmiset --device swp0 --index 68 --cir 100000 --cbs 4000 --ebs 4000 --eir 100000 --cf
   ```

9. Send one yellow stream (rate = 200M) on TestCenter.
   
   ```bash
   ethtool -S swp0
   ```

   All frames pass and get all yellow frames (use CIR as well as EIR).

10. Send one yellow stream (rate = 300M) on TestCenter.
    
    ```bash
    ethtool -S swp0
    ```

   Note that not all frames could pass and get yellow and red frames.

Port-based SFI set

LS1028A switch can work on port-based PSFP set. This implies that when a null-identified stream is received on an ingress port, switch will use the port, default SFI.

Below example tests no streamhandle in qcisfiset to set a port, default SFI.

1. Use SFID 2 to set swp0 port as default SFI.
   
   ```bash
   tsntool> qcisfiset --device swp0 --index 2 --gateid 1 --flowmeterid 68
   ```

   After the port default SFI set, any stream sent from swp0 port will do the gate 1 and flowmeter 68 policy.

2. Set stream gate control.
   
   ```bash
   echo "t0 1b 4 50000 200" > sgi.txt
   tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --
gatelistfile sgi.txt
   ```

3. Send any stream to swp0.
   
   ```bash
   ethtool -S swp0
   ```

   Note that the frame could pass, and green_prio_4 has increased.

6.1.4.2.5.2 Tc-flower usage

The figure below illustrates the TC-flower-based Qci test case setup.
1. Add the MAC address 'CA:9C:00:BC:6D:68' in MAC table by using bridge fdb command if it is not learned.

   ```
   bridge fdb add add dev swp3 CA:9C:00:BC:6D:68 vlan 1 master static
   ```

2. Register chains on ingress port swp0. Refer to Section 6.4.2.

   ```
   tc qdisc add dev swp0 clsact
   tc filter add dev swp0 ingress chain 0 pref 49152 flower skip_sw action goto chain 10000
   tc filter add dev swp0 ingress chain 10000 pref 49152 flower skip_sw action goto chain 11000
   tc filter add dev swp0 ingress chain 11000 pref 49152 flower skip_sw action goto chain 12000
   tc filter add dev swp0 ingress chain 12000 pref 49152 flower skip_sw action goto chain 20000
   tc filter add dev swp0 ingress chain 20000 pref 49152 flower skip_sw action goto chain 21000
   tc filter add dev swp0 ingress chain 21000 pref 49152 flower skip_sw action goto chain 30000
   ```

3. Set Qci on ingress port swp0.
a) Use the following commands to set Qci gate.

```bash
tc filter add dev swp0 ingress chain 30000 protocol 802.1Q flower skip_sw
   dst_mac CA:9C:00:BC:6D:68 vlan_id 1 action gate index 1 base-time 0 sched-entry
   CLOSE 6000 -1 -1
```

b). Use the following commands to set Qci flow meter.

```bash
tc filter add dev swp0 ingress chain 30000 protocol 802.1Q flower skip_sw
   dst_mac CA:9C:00:BC:6D:68 vlan_id 1 action police index 1 rate 10Mbit burst
   10000 conform-exceed drop/ok
```

c). Use the following commands to set Qci SFI priority.

```bash
tc filter add dev swp0 ingress chain 30000 protocol 802.1Q flower skip_sw
   dst_mac CA:9C:00:BC:6D:68 vlan_id 1 vlan_prio 1 action gate index 1 base-time 0
   sched-entry CLOSE 6000 -1 -1
```

d). Use the following commands to set both gate and flow meter.

```bash
tc filter add dev swp0 ingress chain 30000 protocol 802.1Q flower skip_sw
   dst_mac CA:9C:00:BC:6D:68 vlan_id 1 action gate index 1 base-time 0 sched-entry
   OPEN 6000 2 -1 action police index 1 rate 10Mbit burst 10000 conform-exceed
drop/ok
```

3. Send a stream from TestCenter, set the stream destination mac as CA:9C:00:BC:6D:68, set vid=1 and vlan_prio=1 in the vlan tag.

4. Using "tcpdump -i eno0 -w eno0.pcap" to receive the stream on eno0, check if packets are received.

5. Use the following commands to delete a stream rule.

```bash
tc -s filter show dev swp0 ingress chain 30000
tc filter del dev swp0 ingress chain 30000 pref 49152
```

**Note:**

- Each stream can only be added only once. If a user wants to update it, delete the rule and add a new one.
- MAC and VID of stream must have been learned in switch MAC table if the stream is required to be added.
- Qci gate cycle time is expected to be more than 5 μs.
- Qci flow meter can only set cir and cbs now, and the policers are shared with ACL VCAPs.

### 6.1.4.2.6 Qav

The below figure illustrates the Qav test case setup.
6.1.4.2.6.1 Tsntool usage

1. Set the percentage of two traffic classes:

```
  tsntool> ctset --device swp0 --queue_stat 0x0
  tsntool> ctset --device swp1 --queue_stat 0x0
  tsntool> ctset --device swp2 --queue_stat 0x0
  tsntool> cbsset --device swp2 --tc 1 --percentage 20
  tsntool> cbsset --device swp2 --tc 2 --percentage 40
```

2. Send two streams from TestCenter, then check the frames count.

```
ethtool -S swp2
```

*Note that the frame count of queue1 is half of queue2.*

*Note: Stream rate must larger than bandwidth limited of queue.*

3. Capture frames on swp2 on TestCenter.

```
  # The Get Frame sequence is: (PCP=1), (PCP=2), (PCP=2), (PCP=1), (PCP=2),
  # (PCP=2), ...
```

6.1.4.2.6.2 Tc-cbs usage

LS1028A supports the CBS qdisc to setup Credit-based Shaper. The below commands set CBS with 20 Mbit/s for queue 1 and 40 Mbit/s for queue 2.
1. Set the cbs of two traffic classes:

```
tc qdisc add dev swp2 root handle 1: mgprio num_tc 8 map 0 1 2 3 4 5 6 7 \  queues 100 101 102 103 104 105 106 107 hw 0\tc qdisc replace dev swp2 parent 1:2 cbs locredit -1470 hicredit 30 \  sendslope -980000 idleslope 20000 offload 1\tc qdisc replace dev swp2 parent 1:3 cbs locredit -1440 hicredit 60 \  sendslope -960000 idleslope 40000 offload 1
```

2. Send one stream with PCP=1 from TestCenter, we can get the stream bandwidth is 20 Mbit/s from swp2.

3. Send two streams from TestCenter, then check the frames count.

```
ethtool -S swp2
```

Note: The frame count of queue1 is half of queue2.

4. Delete the cbs rules.

```
tc qdisc del dev swp2 parent 1:2 cbs
```

6.1.4.2.7 802.1CB

The Figure 74 describes the test setup for the seamless redundancy test case.

6.1.4.2.7.1 Sequence Generator test

Use the following set of commands for the 'Sequence Generator' test.

1. Configure switch ports to be forward mode.

On board A:

```
ifconfig eno2 up
ip link add name switch type bridge vlan_filtering 1
ip link set switch up
ip link set swp0 master switch & & ip link set swp0 up
```
ip link set swp1 up
ip link set swp2 master switch && ip link set swp2 up
ip link set swp3 master switch && ip link set swp3 up
bridge vlan add dev swp0 vid 1 pvid
bridge vlan add dev swp2 vid 1 pvid
bridge vlan add dev swp3 vid 1 pvid

2. On board A, run the commands:

bridge fdb add 7E:A8:8C:9B:41:DD dev swp2 vlan 1 master static
tsntool> cbstreamidset --device swp0 --index 1 --nullstreamid --nulldmac
0x7EA88C9B41DD --nullvid 1 --streamhandle 1
tsntool> cbgen --device swp3 --index 1 --iport_mask 0x08 --split_mask 0x07 --seq_len 16 --seq_num 2048

In the command above,
• device: can be any one of switch ports.
• index: value is the same as streamhandle of cbstreamidset.

3. Send a stream from TestCenter to swp3 of board A, set destination mac as 7E:A8:8C:9B:41:DD.

4. Capture frames on swp2 on TestCenter.
   We can get frames from swp2 on TestCenter, each frame adds the sequence number: 23450801, 23450802, 23450803...

5. Capture frames from swp2 of board B on TestCenter, we can get the same frames.

6.1.4.2.7.2 Sequence Recover test

Use the following steps for the Sequence Recover test:

1. On board B, run the following commands:

bridge fdb add 7E:A8:8C:9B:41:DD dev swp2 vlan 1 master static
tsntool> cbstreamidset --device swp2 --index 1 --nullstreamid --nulldmac
0x7EA88C9B41DD --nullvid 1 --streamhandle 1
tsntool> cbridge --device swp0 --index 1 --seq_len 16 --his_len 31 --rtag_pop_en

In the cbridge command mentioned above:
• device: can be any one of switch ports.
• index: value is the same as streamhandle of cbstreamidset.

2. Send a frame from TestCenter to swp3 of board A, set the destination MAC address to be 7E:A8:8C:9B:41:DD.

3. Capture frames from swp2 of board B on the TestCenter, we can get only one frame without sequence tag.
6.1.4.2.8 TSN stream identification

TSN module uses QoS class to identify and control streams. There are three ways to identify the stream to different QoS class. These are explained in the following sections.

6.1.4.2.8.1 Stream identification based on PCP value of Vlan tag

The default QoS class is based on PCP of Vlan tag for a frame. If there is no Vlan tag for a frame, the default QoS class is 0.

Set the PCP value on TestCenter.

![Figure 75. Using PCP value of Vlan tag](image)

6.1.4.2.8.2 Based on DSCP of ToS tag

Use the below steps to identify stream based on DSCP value of ToS tag.
1. Map the DSCP value to a specific QoS class using the command below:

```
tsntool> dscpset --device swp0 --index 1 --cos 1 --dpl 0
```

**Explanation:**
- **index**: DSCP value of stream, 0-63.
- **cos**: QoS class which is mapped to.
- **dpl**: Drop level which is mapped to.

2. Set the DSCP value on TestCenter. DSCP value is the upper six bits of ToS in IP header, set the DSCP value on TestCenter as shown in Figure 76.
6.1.4.2.8.3 Based on qci stream identification

The following steps describe how to use qci to identify the stream and set it to a QoS class.

1. Identify a stream.

```
  tsntool> cbstreamidset --device swp1 --nullstreamid --nulldmac 0x000183fe1201
  --nullvid 1 --streamhandle 1
```

Figure 76. Setting DSCP value on TestCenter
tsntool> qcisfiset --device swp0 --index 1 --gateid 1 --flowmeterid 68

2. Set to Qos class 3 by using stream gate control.

echo "t0 1b 3 50000 200" > sgi.txt
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile sgi.txt

Note: The Qci-based identity stream can only be used on both the ingress and egress are bridge ports. The flow injected or extracted through the CPU port cannot be configured for Qci.

6.2 GenAVB/TSN stack

This section describes the steps for configuring GenAVB/TSN stack.

6.2.1 Introduction

The GenAVB/TSN Stack provides advanced implementation for Audio Video Bridging (AVB) and Time-Sensitive Networking (TSN) functionalities on NXP SoCs and hardware platforms, for both Endpoints and Bridges.

This section provides information on how to set up and evaluate the GenAVB/TSN Stack. In that context, it provides information on supported SoCs and boards, compile time software package configuration, and runtime configuration settings.

The GenAVB/TSN stack supports the following roles:

- **TSN Endpoint**
  TSN Endpoint functionality requires TSN hardware support, available on i.MX 93, i.MX 8M Plus, and i.MX 8DXL SoCs.

- **AVB/TSN Bridge**
  AVB/TSN Bridge functionality requires AVB/TSN hardware support, available in LS1028A SoC.

- **AVB Endpoint**
  AVB Endpoint functionality is provided in i.MX 93, i.MX 8M Plus, i.MX 8DXL, i.MX 8M Mini, and i.MX 6ULL SoCs (using hardware support if available).

6.2.1.1 gPTP Stack

The gPTP stack implements IEEE 802.1AS-2020 standard, and supports both time-aware Endpoint and Bridge systems. The stack runs fully in userspace, using Linux socket APIs for packet transmit, receive, and timestamping. Linux clock APIs are used for clock adjustment. Configuration files are used to configure the stack at initialization time and extensive logging is available at runtime.

6.2.1.2 SRP stack

The SRP stack implements MRP, MVRP, and MSRP defined in IEEE 802.1Q-2018, sections 10, 11, and 35, and supports both Endpoint and Bridge systems. The stack runs fully in userspace, using Linux socket APIs for packet transmit and receive. Linux tc and bridge netlink APIs are used to update Multicast FDB entries and FQTSS Credit Based Shaper (CBS) configuration. Configuration files are used to configure the stack at initialization time and extensive logging is available at runtime.

6.2.1.3 AVTP Stack

The AVTP stack implements IEEE 1722-2016 standard, supporting both AVB Talker/Listener end stations and multiple Audio/Video stream formats. The stack runs in userspace, in combination with a Linux AVB kernel.
module, providing low latency network packet processing and AVTP packet encapsulation/decapsulation. The stack provides an API for external media applications through a run time library. The API allows external applications to act as sources of AVTP Talker streams/sinks of AVTP Listener streams.

6.2.1.4 AVDECC/Milan Stack

The AVDECC stack implements IEEE 1722.1-2013 standard, and supports Talker, Listener and Controller entities. The stack also implements Milan “Discovery, connection and control specification for talkers and listeners Revision 1.1a” standard, which can be enabled at initialization time. AVDECC entity definitions are loaded from the filesystem and can be created based on a C header file definition. The stack provides an API for external media applications through a run time library.

6.2.1.5 MAAP Stack

The MAAP stack implements IEEE 1722-2016, Annex B. The stack provides an API for external media applications through a run time library, but it mainly serves the AVDECC/Milan stack.

6.2.1.6 Supported configurations

GenAVB/TSN stack currently supports the following boards and the associated roles:

- LS1028ARDB: gPTP Time-aware Bridge and SRP Bridge
- i.MX 93 EVK: gPTP Time-aware Endpoint station, TSN Endpoint, and AVB Endpoint stack/applications.
- i.MX 93 9x9 LPDDR4 QSB: gPTP Time-aware Endpoint station and TSN Endpoint.
- i.MX 8M Plus LPDDR4 EVK: gPTP Time-aware Endpoint station, TSN Endpoint, and AVB Endpoint stack/applications.
- i.MX 8M Mini LPDDR4 EVK: gPTP Time-aware Endpoint station and AVB Endpoint stack/applications.
- i.MX 6ULL 14x14 EVK: gPTP Time-aware Endpoint station and AVB Endpoint stack/applications. Along with AVB bridge stack when connecting SJA1105Q-EVB Hardware.

The TSN stack supports and is enabled in the following Yocto Real-time Edge machines:

- imx93evk
- imx93-9x9-lpddr4-qsb
- imx8mp-lpddr4-evk
- ls1028ardb
- imx8dxlb0-lpddr4-evk

The AVB Endpoint stack supports and is enabled in the following Yocto Real-time Edge machines:

- imx6ull14x14evk
- imx8mm-lpddr4-evk
- imx8mp-lpddr4-evk
- imx8dxlb0-lpddr4-evk

Follow Real-time Edge Software Yocto Project to get the code and build images for these platforms.

6.2.1.7 AVB endpoint example applications

The stack provides extensive example applications for media playback/capture and server. Please refer to “GenAVB/TSN Stack Evaluation User Guide” for detailed information. Refer Section 1.4.
6.2.1.8 TSN endpoint example application

The TSN example application provides example code and re-usable middleware exercising the GenAVB/TSN API. It can be used to exercise and verify the real time behavior of the local system as well as TSN properties of the network between endpoints. Figure 77 shows the TSN application cycle.

The TSN example application implements a control loop similar to industrial use cases requiring cyclic isochronous exchanges over the network.

The TSN endpoints run their application synchronized to a common time grid in the same gPTP domain so that they can send and receive network traffic in a cyclic isochronous pattern (the application cycle time is equal and synchronous to the network cycle time as shown in Figure 77). Currently the cycle is configured with a period of 2 ms, and periods as low as 1 ms have been confirmed to work as well. When the application is scheduled, frames from other endpoints are ready to be read and at the end of the application time frames are sent to other endpoints.
As shown in Figure 78, the controller and the IO devices are scheduled with a half cycle offset in order to reduce the processing latency.

The time sensitive traffic is layer 2 multicast with VLAN header and proprietary EtherType. Its priority is defined using the PCP field of the VLAN header.

In addition, the TSN application provides detailed logs and time sensitive traffic timing statistics (based on hardware timestamping of packets), which allow characterization of an entire real time distributed system.

Finally, an OPCUA server is implemented and offer the possibility to browse and retrieve the TSN application statistics exposed as OPCUA objects. The OPCUA server runs over TCP and allows access to any OPCUA client.

6.2.2 GenAVB/TSN stack start/stop

GenAVB/TSN stack can be manually started/stopped at runtime by using the commands listed below.

1. To start the TSN stack (if not already started) and start/stop the demo applications:
   
   ```
   # avb.sh <start|stop>
   ```

2. To just start/stop the TSN stack (gPTP and SRP) use:
   
   ```
   # fgptp.sh <start|stop>
   ```

3. To restart/stop all GenAVB/TSN processes, TSN stack, and demo applications:
   
   ```
   avb.sh restart_all/stop_all
   ```

4. Real-time Edge also provides a systemd service to run genavb-tsn stack as a system service.
   
   ```
   # systemctl enable genavb-tsn
   # systemctl start genavb-tsn
   ```

5. The below commands can be used to stop or disable this service.
   
   ```
   # systemctl stop genavb-tsn
   # systemctl disable genavb-tsn
   ```
6.2.3 Use cases description

6.2.3.1 AVB endpoint

AVB endpoint use cases and example applications are described in the GenAVB/TSN Stack AVB Endpoint User Guide located in Real Time Edge Documentation.

6.2.3.2 gPTP Bridge

LS1028ARDB or i.MX8DXL EVK paired with a SJA1105Q-EVB can be used as a generic time-aware bridge, connected to other time-aware end stations or bridges.

By default, they don't forward packets if no bridge interface is configured under Linux. Enabling bridge interface is dependent on the board used. For example, the configuration of bridge interface is described in section Section 6.1.4.2.1.

Note: The i.MX8DXL EVK uses eth0 instead of eno2.

Once gPTP stack is started, logs can be displayed with the following command:

```
# tail -f /var/log/fgptp-br
```

In this log file, one can observe which ports are connected, which ports are currently communicating a synchronized time and what is the role of the port in the time-aware system.

If a port of the bridge is connected to another port capable of communicating a synchronized time, the following log should appear for each enabled gPTP domain:

```
gptp_stats_dump : Port(1) domain(0,0): Role: Master Link: Up asCapable: Yes neighborGptpCapable: Yes DelayMechanism: P2P
...  
gptp_stats_dump : Port(1) domain(1,20): Role: Master Link: Up asCapable: Yes neighborGptpCapable: Yes DelayMechanism: COMMON_P2P
```

Role status can also take the value Slave depending on parameters described in section Section 6.2.4.2.2.

If a port is not connected, Link status takes the value Down.

If a port is not capable of communicating a synchronized time, AS_Capable status takes the value No.

If a port is using the Common Mean Link Delay Service (CMLDS) the DelayMechanism takes the value COMMON_P2P, else the value P2P.

For further details about gPTP logs, refer to section Section 6.2.5.2.

6.2.3.3 gPTP Endpoint

Once gPTP stack is started, logs can be displayed using the following command:

```
# tail -f /var/log/fgptp
```

In this log file, one can observe the role of the port in the time-aware system.

If the port of the endpoint is connected to another port capable of communicating a synchronized time, the following log should appear for each gPTP domain:

```
gptp_stats_dump : Port(0) domain(0,0): Role: Slave Link: Up AS_Capable: Yes neighborGptpCapable: Yes DelayMechanism: P2P
...  
```
Role status can also take the value Master depending on Grandmaster Parameters described in section Section 6.2.4.2.2.

If a port is not connected, Link status takes the value Down.

If a port is not capable of communicating a synchronized time, AS_Capable status takes the value No.

If a port is using the Common Mean Link Delay Service (CMLDS) the DelayMechanism takes the value COMMON_P2P, else the value P2P.

For further details about gPTP logs, refer to section Section 6.2.5.1.

6.2.3.4 gPTP multiple domains

This use case illustrates two gPTP domains co-existing independently on a TSN network, over different 802.1AS-2020 Time-aware systems.

The first domain uses the PTP timescale whereas the second domain uses the ARB (arbitrary) timescale.

6.2.3.4.1 Requirements

The reference setup for gPTP multiple domains is made of:

- Two gPTP endpoints EP1-DUT and EP2-DUT:
  - (i.MX 8M Plus LPDDR4 EVK or i.MX 93 EVK connected through TSN interface eth1: dwnacOr
  - i.MX 8DXL LPDDR4 EVK or i.MX 93 9x9 LPDDR4 QSB connected through TSN interface eth0):
- One gPTP bridge (LS1028ARDB): BR-DUT
6.2.3.4.2 gPTP Stack Configuration

The gPTP stack can enable or disable each domain independently through a configuration file.

The default configuration file (for example: `/etc/genavb/fgptp.cfg`) is for general gPTP parameters as well as domain 0 parameters. To enable other domains, new files must be created with `-N` appended to the filename (for example: `/etc/genavb/fgptp.cfg-1` for domain 1).

For gPTP multiple domains, all devices configuration should be changed to support two domains. The first domain (domain 0) must be assigned domain number 0. The second domain (domain 1) is assigned domain number 20.

BR-DUT is defined as the GrandMaster for the first domain (domain 0). EP1-DUT is defined as the GrandMaster for the second domain (domain 1).

On EP1-DUT, edit the file `/etc/genavb/fgptp.cfg-1` and change `domain_number` and `priority1` parameters as follows:

```
domain_number = 20
priority1 = 245
```

On EP2-DUT, edit the file `/etc/genavb/fgptp.cfg-1` and change `domain_number` parameter as follows:

```
domain_number = 20
```
On BR-DUT, edit the file `/etc/genavb/fgptp-br.cfg-1` and change `domain_number` parameter as follows:

```
domain_number = 20
```

**Note:**

- **On Domain 0,** BR-DUT is the GrandMaster with the highest priority (lowest value) among all devices in the domain (BR-DUT priority=246, EP1-DUT and EP2-DUT priority=248).
- **On Domain 1,** EP1-DUT is the GrandMaster with the highest priority (lowest value) among all devices in the domain (BR-DUT priority=246, EP1-DUT priority=245 and EP2-DUT priority=248).
- By default,
  - All ports on Domain 0 are configured to use the per instance peer delay mechanism (DelayMechanism=P2P).
  - All ports on Domain 1 are configured to use the CMLDS (DelayMechanism=COMMON_P2P).

### 6.2.3.4.3 Evaluation instructions

**Test Procedure**

1. Start gPTP stack manually on all DUTs by issuing the command below:
   ```
   # tsn.sh start
   ```
2. Wait for 30 s.
3. Check gPTP stack logs on BR-DUT (`/var/log/fgptp-br`), EP1-DUT and EP2-DUT (`/var/log/fgptp`) 

**Verification:**

Check the following:

- **After Step 3,** the log on EP1-DUT reports Port 0 as synchronized on domain 0 only:
  ```
  Port(0) domain(0, 0) SYNCHRONIZED - synchronization time (ms): 250
  ```
- **After Step 3,** the log on EP2-DUT reports Port 0 as synchronized on all domains:
  ```
  Port(0) domain(0, 0) SYNCHRONIZED - synchronization time (ms): 250
  Port(0) domain(1, 20) SYNCHRONIZED - synchronization time (ms): 250
  ```
- **After Step 3,** the log on BR-DUT reports Port 0 as synchronized on domain 1 only:
  ```
  Port(0) domain(1, 20) SYNCHRONIZED - synchronization time (ms): 250
  ```
- The "Initial adjustment" message should be reported only once per synchronized domains (domain 0 for EP1-DUT and EP2-DUT, domain 1 for EP2-DUT and BR-DUT):
  ```
  domain(0,0) Initial Adjustment, offset: 125486471315484 ns, freq_adj: 32764
  domain(1,20) Initial Adjustment, offset: 125455671332661 ns, freq_adj: 16384
  ```

Once synchronization is achieved, all the reported clock offset average values should be stable within -50 to +50 ns range (domain 0 for EP1-DUT and EP2-DUT, domain 1 for EP2-DUT and BR-DUT):

- **domain(0,0) Offset between GM and local clock (ns):** min -45 avg 0 max 35
- **domain(1,20) Offset between GM and local clock (ns):** min -66 avg 0 max 15
6.2.3.5 AVB Bridge

This use case illustrates an AVB Bridge (mixing gPTP and SRP stack) with other AVB Endpoints

6.2.3.5.1 Requirements

6.2.3.5.1.1 Using LS1028ARDB as AVB bridge

- Two AVB endpoints
- One AVB bridge (LS1028ARDB)

6.2.3.5.1.2 Using i.MX8DXL EVK with SJA1105Q EVB as AVB bridge

- Two AVB endpoints including:
  - an i.MX8DXL with an IMX-RMII-PHY TJA1100
  - an i.MX8MPlus with an Ethernet to BoardR-Reach converter
- One AVB bridge (i.MX8DXL with a SJA1105Q-EVB)
Figure 81. AVB Bridge setup

Setup the device tree:

- On the i.MX8DXL Bridge with a SJA1105Q-EVB. Use the following device tree in U-boot:
  ```
  setenv fdt_file imx8dxl-evk-enet0-sja1105.dtb
  ```

- On the i.MX8DXL Endpoint with a TJA1100. Use the following device tree in U-boot:
  ```
  setenv fdt_file imx8dxl-evk-enet0-tja1100-avb.dtb
  ```

Configure MASTER/SLAVE role of the PHYs:

- On the Bridge with SJA1105Q-EVB, check the default phy role for the connected ports:
  ```
  ethtool swpX
  ```
  **Note:** Default role is Master (ethtool will report forced-master under master-slave cfg)

- On the endpoint with TJA1100 PHY Daughter card, set the PHY role accordingly (master if SJA1105 port reports slave and vice versa):
  ```
  ethtool -s eth0 master-slave forced-slave
  ```

- On the automotive media converter, set the role using the same method

### 6.2.3.5.2 AVB network configuration

This topic describes AVB configuration.
6.2.3.5.2.1 Priority to traffic class mapping

The priority to traffic class mapping used for the bridge comes directly from the recommended mapping for two SR classes in IEEE Std 802.1Q-2018 Table 34-1:

<table>
<thead>
<tr>
<th>Priority</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic Class</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 66. Priority to traffic class mapping

The Bridge should be configured to forward VLAN tagged packets based on their PCP values according to this mapping, and should configure credit-based shapers on the two highest traffic classes (traffic class 6 and traffic class 7) for SR class A (priority 3) and SR class B (priority 2) traffic.

Refer to Section 6.2.3.5.3 for the bridge PCP mapping configuration.

6.2.3.5.2.2 FQTSS Credit Based Shapers configuration

The SRP bridge stack relies on preconfigured qdiscs with specific handles to configure the hardware's credit-based shapers, on the two hardware queues with the two highest traffic classes, for every port. Thus, an mqprio qdisc with 8 traffic classes should be configured with the above priority to traffic class mapping and credit-based shapers qdiscs with the following handles: 0x9006 for CBS on traffic class 6 and 0x9007 for CBS on traffic class 7.

Refer to Section 6.2.3.5.3 for the bridge qdisc configuration.

6.2.3.5.2.3 Linux Best Effort Traffic classification

Linux classifies egress packets, for assignment to traffic classes, based on skb priorities. To avoid assigning egress best effort traffic to traffic classes with configured credit-based shapers, the skb priorities should be rewritten so no packets with skb priorities 2 and 3 are present on egress. Furthermore, the bridge code is using the skb priority as the traffic class for packets injected from the CPU port, making packets with skb priorities 6 and 7 end up in the hardware’s traffic classes 6 and 7 on the external ports which in turn harms traffic shaping. Again, forcing a remapping of these skb priorities avoids this scenario.

Refer to Section 6.2.3.5.3 for the skb priorities remapping configuration.

6.2.3.5.2.4 Bridge VLAN awareness

A proper AVB bridge functioning requires that the switch forward AVB streams (with multicast destination MAC addresses and specific VLAN ID) only to ports configured in the Forwarding DataBase (FDB). For that, we should enable VLAN filtering on bridge level, add the desired VLAN ID to all ports and disable the default multicast flooding configuration (at least for the two highest priority queues) on all the external ports.

Refer to Section 6.2.3.5.3 for the bridge vlan configuration.

6.2.3.5.3 Setup preparation

This has two steps described in the following sections.

6.2.3.5.3.1 GenAVB/TSN stack configuration

This configuration needs to be done once and is saved across reboots.
Edit the GenAVB/TSN configuration file using the following command at the Linux prompt:

```
# vi /etc/genavb/config
```

And set the GENAVB_TSN_CONFIG correctly as "Bridge AVB".

**Note:** It's the default configuration on LS1028A while the Bridge with SJA1105Q needs to be changed accordingly.

For a proper gPTP operation with AVB endpoints, the gPTP stack needs to compensate for PHY delay in PTP timestamps:

- **AVB Bridge on LS1028A**
  
  In the `/etc/genavb/fgptp-br.cfg`, apply the settings (rxDelayCompensation and txDelayCompensation) described in Table 78 on all bridge ports.

  **Attention:** The PHY Delay Compensation Values in Table 78 are calibrated for 1 Gbps links. The i.MX AVB endpoints are configured to run by default with 100 Mbps links. These compensation values should be enough to keep pDelay values under 800 ns (propagation time threshold), and therefore the port would still be declared as Capable. If, with these values, the calculated propagation delay is still above 800 ns (or too close to it), adapt them accordingly (decrease rxDelayCompensation and/or increase txDelayCompensation).

  Future releases shall have proper compensation values for each supported link speed.

- **AVB Bridge on SJA1105Q**

  On AVB bridge setup with SJA1105Q-EVB, the calculated propagation delay is exceeding default propagation delay threshold (800ns) and the link would not be asCapable. To bypass that, increase the neighborPropDelayThreshold on all devices, including the bridge:

  ```
  vi /etc/genvb/fgptp-br.cfg
  ```

  Set the neighborPropDelayThreshold to 8000ns

  The same should be done on endpoints connected to the bridge: `/etc/genavb/fgptp.cfg`

6.2.3.5.3.2 Bridge configuration

This configuration should be done after each boot. The user can either enter these commands manually or execute a ready to use script provided by GenAVB/TSN stack.

1. If you're using an i.MX8DXL Bridge, edit the GenAVB/TSN configuration file using the following command at the Linux prompt:

   ```
   # vi /etc/genavb/config
   ```

   And set the configuration GENAVB_TSN_CONFIG to use the config_avb_bridge file:

   ```
   GENAVB_TSN_CONFIG=3
   ```

2. Execute the automated configuration script and start the AVB bridge stack:

   ```
   # avb-bridge.sh
   # avb.sh start
   ```

3. Alternatively, configure the bridge manually using the following commands:

   a. Setup bridge forwarding:

      ```
      # ip link add name br0 type bridge
      # ip link set br0 up
      # ip link set master br0 swp0 up
      ```
# ip link set master br0 swp1 up
# ip link set master br0 swp2 up
# ip link set master br0 swp3 up

b. Establish the PCP to QoS mapping for every port on the bridge using a custom tool (supported only on LS1028A):

```bash
# pcp_to_qos_map=(0="1" 1="0" 2="6" 3="7" 4="2" 5="3" 6="4" 7="5");
for port in @avb_ports@; do
    for ((pcp=0; pcp < 8; ++pcp)); do
        tsntool pcpmap -d $port -p $pcp -e 0 -c ${pcp_to_qos_map[$pcp]} -l 0;
        tsntool pcpmap -d $port -p $pcp -e 1 -c ${pcp_to_qos_map[$pcp]} -l 1;
    done;
done
```

c. Configure the qdiscs and shapers, with the correct handles, and optionally offload the pcp to qos mapping for every external port:

```bash
# pcp_to_qos_map=(0="1" 1="0" 2="6" 3="7" 4="2" 5="3" 6="4" 7="5");
for port in @avb_ports@;
        tc qdisc add dev $port root handle 100: mqprio num_tc 8 map ${pcp_to_qos_map[@]} queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 hw <HW Offload> ;
        tc qdisc replace dev $port handle 0x9007 parent 100:8 cbs locredit -2147483646 hicredit 2147483647 sendslope -1000000 idle_slop 0 offload 0 ;
        tc qdisc replace dev $port handle 0x9006 parent 100:7 cbs locredit -2147483646 hicredit 2147483647 sendslope -1000000 idle_slop 0 offload 0 ;
        done
```

Note:

<HW offload>: set to 1 if the driver supports MQPrio PCP mapping offload (SJA1105Q), otherwise 0 (LS1028A).

The two most important CBS parameters for every port device are:
- the parent, which should match the traffic class 6 and 7,
- the handle, which should be 0x9006 and 0x9007.

The other parameters are initialization values and are overridden by the stack at runtime stream configuration:
- offload is set to 1 to offload the operation to hardware,
- idleslope and sendslope are set depending on stream,
- port bit rates and the credit values are kept at their minimum and maximum values as they do not directly affect the hardware shaping operation.

d. Setup skb priorities remapping for every external port:

```bash
# avb_ports="swp0 swp1 swp2 swp3";
for port in @avb_ports@; do
    tc qdisc add dev $port clsact;
    tc filter add dev $port egress basic match 'meta(priority eq 2)' or 'meta(priority eq 3)' action skbedit priority 0;
done
```

4. Enable Vlan filtering, set the correct Vlan IDs and disable multicast flooding, for every external port:

```bash
# ip link set br0 type bridge vlan_filtering 1;
# avb_ports="swp0 swp1 swp2 swp3";
for port in @avb_ports@; do
    bridge vlan add dev $port vid 2 master;
    bridge link set dev $port mcast_flood off;
done
```
5. Start the AVB and gPTP stacks:

   # avb.sh start

   • Since multicast traffic flooding is now disabled, adding MDB entries for AVDECC (ACMP/ADP) and MAAP protocols multicast addresses is needed. The following commands should be executed for every port facing an AVB endpoint.

   # bridge mdb add dev br0 port <port> grp 91:e0:f0:01:00:00 permanent
   # bridge mdb add dev br0 port <port> grp 91:e0:f0:00:ff:00 permanent

6.2.3.5.4 Evaluation instructions

1. Reset all endpoints and the bridge.
2. Using the procedures described above, configure the bridge and start the stack on all connected devices (bridge and endpoints)
3. After a few seconds, AVB endpoints should be synchronized through gPTP
4. Connect an SR class A (or SR class B) stream from EP-DUT2 as talker to EP-DUT1 as listener: the stream should be forwarded correctly to the listener endpoint

6.2.3.5.4.1 gPTP operation

If the gPTP protocol is running correctly on all devices, the following line should appear in the bridge gptp log file for every port connected to a gPTP capable device:

<table>
<thead>
<tr>
<th>gptp_stats_dump: Port(0) domain(0, 0): Role: Master   Link: Up   asCapable: Yes neighborGptpCapable: Yes DelayMechanism: P2P</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>gptp_stats_dump: Port(1) domain(0, 0): Role: Master   Link: Up   asCapable: Yes neighborGptpCapable: Yes DelayMechanism: COMMON_P2P</td>
</tr>
</tbody>
</table>

Refer to Section 6.2.3.2, for more details on gPTP Bridge operation.

6.2.3.5.4.2 SRP Operation

A detailed view on the SRP protocol communications (such as Domain declaration, SRP port boundary, Talker/Listener declarations and registration) can be followed by displaying the SRP specific logs from the TSN bridge stack log file /var/log/tsn-br:

   # tail -f /var/log/tsn-br | grep srp

On stream connection, the FQTSS and FDB operation should be visible in the TSN bridge stack log file:

• Stack log shows the FQTSS configuration for the port facing the AVB listener:

   fqtss_set_oper_idle_slope : logical_port(2) port (swp0, ifindex 5) tc(7)
   cbs_qdisc_handle(9007:0): set idle_slope 7872000

• Stack log shows the FDB configuration for the port facing the AVB listener:

   bridge_rtnetlink : add MDB: bridge (br0, ifindex 9) logical_port(2) port (swp0, ifindex 5) mac_addr(91:e0:f0:00:fe:11) vlan_id(2)

Also, the same configuration can be checked using the Linux standard tools (tc and bridge)
• TC tool shows the FQTSS configuration for the port facing the AVB listener:

```bash
# tc qdisc show dev swp0
qdisc mqprio 100: root tc 8 map 1 0 6 7 2 3 4 5 0 0 0 0 0 0 0
 queues:(0:0) (1:1) (2:2) (3:3) (4:4) (5:5) (6:6) (7:7)
qdisc pfifo 0: parent 9006: limit 1000p
qdisc pfifo 0: parent 9007: limit 1000p
qdisc pfifo_fast 0: parent 100:6 bands 3 priomap 1 2 2 1 2 0 0 1 1 1 1 1 1 1
 queues:(0:0) (1:1) (2:2) (3:3) (4:4) (5:5) (6:6) (7:7)
qdisc pfifo_fast 0: parent 100:4 bands 3 priomap 1 2 2 1 2 0 0 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:3 bands 3 priomap 1 2 2 1 2 0 0 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:2 bands 3 priomap 1 2 2 1 2 0 0 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:1 bands 3 priomap 1 2 2 1 2 0 0 1 1 1 1 1 1 1
qdisc cbs 9006: parent 100:7 hicredit 2147483647 locredit -2147483646 sendslope
 -1000000 idleslope 0 offload 0
qdisc cbs 9007: parent 100:8 hicredit 2147483647 locredit -2147483648 sendslope
 -992128 idleslope 7872 offload 1
```

• Bridge tool shows the FDB configuration for the port facing the AVB listener:

```bash
# bridge mdb show
dev br0 port swp0 grp 91:e0:f0:00:fe:11 permanent offload vid 2
```

6.2.3.6 TSN endpoint sample application

6.2.3.6.1 Requirements

• Two TSN endpoints
  – i.MX 8MPlus LPDDR4 EVK or i.MX 93 EVK connected through TSN interface eth1: dwmac
  – i.MX 8DXL LPDDR4 EVK or i.MX 93 9x9 LPDDR4 QSB connected through TSN interface eth0 or optionally an i.MX RT1170 EVK
• One TSN bridge (LS1028ARDB)

*Note: The second IO Device is optional.*
6.2.3.6.2 Configuring GenAVB/TSN stack and example applications

For some platforms, the GenAVB/TSN stack supports both modes: Endpoint TSN and Endpoint AVB.

By default, these platforms are configured as Endpoint TSN. The GENAVB_TSN_CONFIG parameter should be set to the right configuration using the file /etc/genavb/config:

```
# avb.sh stop_all
# vi /etc/genavb/config
```

Platforms that support both Endpoint AVB and Endpoint TSN (for example i.MX 8MP, i.MX 8DXL and i.MX 93), should have:

```
GENAVB_TSN_CONFIG=1
```

For Endpoint TSN mode, the change from one profile to another is made by modifying the /etc/genavb/config_tsn file. This file specifies the application configuration file. APPS_CFG_FILE (apps*.cfg) points to a file containing a demo configuration (application to use, options...). It is parsed by the startup script avb.sh.

TSN configuration profile is made of the application configuration profile. The file /etc/genavb/config_tsn already lists the supported cfg files. Set the PROFILE variable to choose the desired configuration profile.

6.2.3.6.3 TSN network configuration

This topic describes TSN configuration.
6.2.3.6.3.1 Streams

The stream details can be used for analysis and also for computing scheduled traffic timings.

Table 67. TSN streams definition

<table>
<thead>
<tr>
<th>Stream No</th>
<th>Source</th>
<th>Destination</th>
<th>Unicast / Multicast</th>
<th>Destination MAC Address</th>
<th>Vlan ID</th>
<th>Vlan PCP</th>
<th>Frame Length[1] (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream1</td>
<td>Controller</td>
<td>IO device(s)</td>
<td>Multicast</td>
<td>91:e0:f0:00:fe:70</td>
<td>2</td>
<td>5</td>
<td>84</td>
</tr>
<tr>
<td>Stream2</td>
<td>IO device 1</td>
<td>Controller</td>
<td>Multicast</td>
<td>91:e0:f0:00:fe:71</td>
<td>2</td>
<td>5</td>
<td>84</td>
</tr>
<tr>
<td>Stream3</td>
<td>IO device 2</td>
<td>Controller</td>
<td>Multicast</td>
<td>91:e0:f0:00:fe:80</td>
<td>2</td>
<td>5</td>
<td>84</td>
</tr>
</tbody>
</table>

[1] The frame length includes inter frame gap, preamble, start of frame and CRC (can be used as is for timing calculations)

6.2.3.6.3.2 Scheduled traffic

For deterministic packet transmission the use of scheduled traffic is required both on endpoints and bridges.

The default scheduling configuration for the TSN endpoint example application, as shown in Figure 78, leads to the following traffic schedules.

Endpoints

Endpoints are running a schedule with a 2000us period. The base offset of the schedule is aligned to gPTP time modulo 1 second.

Controller transmit gate (for Stream1) opens at 500us offset (relative to the period start).

IO device transmit gate (for Stream2/3) opens at 1000us + 500us offset (relative to the period start).

The gate open interval is around 4us (enough to accommodate the stream frame length plus some margin).

The 500us offset is related to the worst case application latency to send its frame to its peer(s). This value provides a good margin for a Linux PREEMPT-RT system but can be lowered on a well-tuned system.

Bridges

The schedule for all Bridges and all Bridge ports that transmit one of the streams above, must have a 2000 μs period and a base offset aligned to gPTP time modulo 1 second.

One possible schedule is to open transmit gate (for the ports and queues transmitting Stream 1) at offset 500 μs and use a gate open interval that accommodates the worst propagation delay.

It is also possible to use a fixed gate open interval but increase the transmit time offset at each hop along the stream path.

For ports and queues transmitting Stream 2 and 3, open the transmit gate at offset 1000 + 500 μs.

6.2.3.6.4 Setup preparation

One of the TSN endpoint needs to be configured as “controller” and the other one as “IO device”. Both endpoints are connected to the TSN bridge.

Note:

1. On i.MX 8MPlus LPDDR4 EVK and i.MX 93 EVK: the TSN interface used is eth1.
2. On i.MX 8DXL LPDDR4 EVK and i.MX 93 9x9 LPDDR4 QSB: the TSN interface used is eth0 rather than eth1.

6.2.3.6.4.1 Preparing the controller

To be done once:

1. Edit the GenAVB configuration file using the following command at the Linux prompt:
   
   ```
   # vi /etc/genavb/config_tsn
   ```

2. Set the configuration profile to PROFILE 1:
   
   ```
   PROFILE=1
   ```

3. Exit and save.

The below steps should be done at each boot:

4. The system configuration required for the tsn-app can be performed (after setting the correct PROFILE) by using the following command (replace `ethX` with the right TSN network interface):
   
   ```
   * eth1 on i.MX 8M Plus LPDDR4 EVK and i.MX 93 EVK
   * eth0 on i.MX 8DXL LPDDR4 EVK and i.MX 93 9x9 LPDDR4 QSB
   
   # tsn-app-setup.sh ethX
   ```

   **Note:** This script sets many different settings to improve real time system behavior and to setup proper network configuration
   
   • VLAN configuration: the script sets vlan id 2 on the TSN interface as VLAN hardware filtering is enabled by default in kernel.
   • Low latency settings on network interface: the script disable coalescing and flow control on TSN interface.
   • Qdiscs and filters: the script sets taprio qdisc with proper parameters for TX and flower qdisc for RX classification.
   • Interrupts, network tasks, CPU affinities, and priorities: the scripts enable threaded NAPI in kernel and isolate tasks processing TSN traffic on a separate CPU core.

5. Start the TSN demo application using the following command:
   
   ```
   avb.sh start
   ```

6.2.3.6.4.2 Preparing IO device(s)

To be done once:

1. Edit the GenAVB configuration file using the following command at the Linux prompt:
   
   ```
   # vi /etc/genavb/config_tsn
   ```

2. Set the configuration profile to PROFILE 2:
   
   ```
   PROFILE=2
   ```

3. Exit and save.

The below steps should be done at each boot:

4. Edit the system configuration needed for the tsn-app after setting the correct PROFILE, using the command shown below (replace `ethX` with the right TSN network interface):
   
   ```
   * eth1 on i.MX 8M Plus LPDDR4 EVK and i.MX 93 EVK
   ```
5. **Note:** This script sets many different settings to improve real time system behavior and to setup proper network configuration:

- **VLAN configuration:** the script sets vlan id 2 on the TSN interface as VLAN hardware filtering is enabled by default in the kernel.
- **Low latency settings on network interface:** the script disables coalescing and flow control on TSN interface.
- **Qdiscs and filters:** the script sets `taprio qdisc with proper parameters for TX and flower qdisc for RX classification`.
- **Interrupts, network tasks, CPU affinities, and priorities:** the scripts enable threaded NAPI in kernel and isolate tasks processing TSN traffic on a separate CPU core.

6. Start the TSN demo application using the following command:

```bash
tsn-app-setup.sh ethX
```

### 6.2.3.6.4.3 Preparing the Bridge

Refer to section [Section 6.1.2](#) and [Section 6.1.4.2.3.2](#) to configure scheduled traffic on the LS1028ARDB board.

The schedule described in section [Section "Bridges"](#) should be followed.

The below steps should be done at each boot:

1. Setup bridge forwarding:

```bash
# ip link add name br0 type bridge
# ip link set br0 up
# ip link set master br0 swp0 up
# ip link set master br0 swp1 up
# ip link set master br0 swp2 up
# ip link set master br0 swp3 up
```

2. Disable Pause frames:

```bash
# ethtool -A swp0 autoneg off rx off tx off
# ethtool -A swp1 autoneg off rx off tx off
# ethtool -A swp2 autoneg off rx off tx off
# ethtool -A swp3 autoneg off rx off tx off
```

3. Start the gPTP stack:

```bash
# tsn.sh start
```

4. Setup scheduled traffic (see above)

```bash
# tc qdisc del dev swp0 root
# tc qdisc del dev swp1 root
# tc qdisc del dev swp2 root
# tc qdisc replace dev swp0 root taprio \num_tc 8 \map 0 1 2 3 4 5 6 7 \queues 100 101 102 103 104 105 106 107 \base-time 1500000 \sched-entry S 0x20 20000 \sched-entry S 0xdf 1980000 \flags 0x2
# tc qdisc replace dev swp1 root taprio
```
6.2.3.6.4 Preparing the OPC UA client

In order to visualize the data exposed by the TSN endpoint application OPC UA server it is required to use an OPC UA client on a PC connected to the bridge.

1. Install an OPC UA client on a PC:
   b. opcua-commander: CLI alternative based nodejs node-opcua stack. Can be found here: [https://github.com/node-opcua/opcua-commander](https://github.com/node-opcua/opcua-commander)

2. Connect the PC to the bridge. If not already done, setup IP addresses on the endpoint running the TSN example application and also on the PC. Then, make sure you can successfully ping the endpoint using the PC.

6.2.3.6.5 Evaluation instructions

1. Reset all endpoints.
2. Using the procedures described above, start the gPTP stack on the bridge and the tsn-app application on the endpoints with the proper enabled scheduled traffic as configured above.
3. After a few seconds, TSN endpoints should be synchronized through gPTP and exchanging packets at the rate of 500 packets per second (pps). In order to observe this behavior, logs should be checked.

6.2.3.6.5.1 gPTP operation

If the gPTP protocol is running correctly on an endpoint or on the bridge, the following line should appear in the gptp log file (refer to Section 6.2.3.3 for more details):

```
gptp_stats_dump: Port(0) domain(0,0) : Role: Slave Link : Up AS_Capable: Yes DelayMechanism: P2P
```

If the device is grand master, the role field should be “Master” otherwise it should be “Slave”. The line appears periodically, but the role should not change over time, except for significant events (such as a cable disconnection).
6.2.3.6.5.2 Baseline tsn-app operation

If the TSN endpoint sample application is running correctly and receiving valid packets, the following points may be verified in the tsn_app log file (refer to Section 6.2.5.4 for more details).

The following line should appear at regular intervals:

```
socket_stats_print : link up
```

The "valid frames" counter should increment by 2500 (500 pps for 5 seconds) between two appearances of the following log:

```
socket_stats_print : valid frames : XXXXX
```

The various error counters should not increment (it is normal to have non-zero values, because of the startup period when gPTP and/or the remote tsn-app endpoint may not be running and stable):

- "sched early", "sched late", "sched missed", "sched timeout", "sched discont", "clock err"
- "err id", "err ts", "err underflow"
- "frames err" (for both RX and TX directions)

Note:
The checks above apply to all tsn-app endpoints, whether they be the controller or one of the IO devices.

6.2.3.6.5.3 Scheduled traffic evaluation with no concurrent traffic

The observations below assume an otherwise idle system receiving and sending traffic only through the tsn-app application, with a 802.1Qbv schedule in place on all devices (tsn-app endpoints, bridge).

Scheduling error statistics ("sched err") should respect the following:

- min around 8 µs
- avg around 11 µs
- max around 25 µs

```
stats(0xaaaab06ed74b0) sched err min 8817 mean 11120 max 22077 rms^2 125202075 stddev^2 1544829 absmin 7417 absmax 1882057
```

Processing time statistics ("processing time") should respect the following:

- min around 23 µs
- avg around 29 µs
- max around 70 µs

```
stats(0xaaaab06ed7910) processing time min 23400 mean 29185 max 59100 rms^2 85770540 stddev^2 5943315 absmin 19560 absmax 4143240
```

Traffic latency statistics should respect the following:

- min around 503 µs
- avg around 503 µs
- max around 503 µs
6.2.3.6.5.4 Scheduled traffic evaluation with TX best-effort traffic

1. Connect a PC to the 4th port of the LS1028ARDB switch (swp3).
2. Run iperf3 in server mode on the PC (replace ethX by the PC interface connected to the LS1028):

   ```
   # ifconfig ethX 192.168.1.10 up
   # iperf3 -s &
   # iperf3 -s -p 5202 &
   # iperf3 -s -p 5203 &
   # iperf3 -s -p 5204 &
   ```

3. Run iperf3 in client mode on the controller (replace ethX by the controller interface connected to the LS1028):

   ```
   # ifconfig ethX 192.168.1.80
   # taskset b iperf3 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
   # taskset b iperf3 -p 5202 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
   # taskset b iperf3 -p 5203 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
   # taskset b iperf3 -p 5204 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
   ```

4. Observe stats in the tsn-app log files (a 2nd terminal may have to be opened through SSH). The values should match the table below (in µs):

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean</th>
<th>max</th>
<th>stddev^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched err (controller)</td>
<td>21</td>
<td>29</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>Processing time (controller)</td>
<td>47</td>
<td>80</td>
<td>260</td>
<td></td>
</tr>
<tr>
<td>Traffic latency (controller and IO device)</td>
<td>503</td>
<td>503</td>
<td>503</td>
<td>&lt;3000</td>
</tr>
</tbody>
</table>

6.2.3.6.5.5 Scheduled traffic evaluation with RX best-effort traffic

**Note:**

By default, the tsn-app traffic is processed in the same queue as best-effort untagged traffic. To more easily validate tsn-app with best-effort traffic, we should add a VLAN tag with PCP=0 to best-effort packets so they are dispatched into a different queue on receive.

1. Connect a PC to the 4th port of the LS1028ARDB switch (swp3).
2. Run iperf3 in server mode on the controller (replace ethX by the controller interface connected to the LS1028):

   ```
   # ip link add link ethX name ethX.5 type vlan id 5
   # ifconfig ethX.5 192.168.5.80 up
   # taskset b iperf3 -s &
   # taskset b iperf3 -s -p 5202 &
   # taskset b iperf3 -s -p 5203 &
   # taskset b iperf3 -s -p 5204 &
   ```
3. Run iperf3 in client mode on the PC (replace ethX by the PC interface connected to the LS1028):

```bash
# ip link add link ethX name ethX.5 type vlan id 5
# iperf3 -c 192.168.5.80 -u -b 0 -i 2 -t 100 &
# iperf3 -p 5202 -c 192.168.5.80 -u -b 0 -i 2 -t 100 &
# iperf3 -p 5203 -c 192.168.5.80 -u -b 0 -i 2 -t 100 &
# iperf3 -p 5204 -c 192.168.5.80 -u -b 0 -i 2 -t 100 &
```

4. Observe stats in the tsn-app log file (a 2nd terminal may have to be opened through SSH). The values should match the table below (in µs):

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean</th>
<th>max</th>
<th>stddev^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched err (controller)</td>
<td>9</td>
<td>13</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Processing time (controller)</td>
<td>25</td>
<td>33</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>Traffic latency (controller and IO device)</td>
<td>503</td>
<td>503</td>
<td>503</td>
<td>&lt;130000</td>
</tr>
</tbody>
</table>

6.2.3.6.5.6 Modifying the scheduling period of the TSN sample application

The default tsn-app period of 2 ms can be changed through a command-line option. The change has to be made on all endpoints (controller and devices). The 802.1 Qbv schedule must also be updated to reflect the new period. The example below shows how to modify the period from the default 2 ms down to 1 ms (this value has been confirmed to work on the latest builds).

On the controller:

1. Stop the application if it was already running:

```bash
# avb.sh stop
```

2. Edit the application configuration file:

```bash
# vi /etc/genavb/apps-tsn-network-controller.cfg
```

or for an IO device:

```bash
# vi /etc/genavb/apps-tsn-network-iodevice.cfg
```

3. Use the "-p" option to change the period. The below example sets the period to 1 ms (1000000 ns):

```bash
CFG_EXTERNAL_MEDIA_APP_OPT="-m network_only -r controller -p 1000000"
```

4. Update the traffic schedule using 'tc' command.

In the sample command below, replace ethX with the right TSN network interface:

- eth1 on i.MX 8M Plus LPDDR4 EVK and i.MX 93 EVK
- eth0 on i.MX 8DXL LPDDR4 EVK and i.MX 93 9x9 LPDDR4 QSB

```bash
# tc qdisc del dev ethX root
#tc qdisc replace dev ethX root taprio \n num_tc 5 \n map' 0 1 1 2 2 3 4 0 0 0 0 0 0 0 0 \n queues 100 101 102 103 104 \n base-time 250000 \n sched-entry S 0x4 4000 \n sched-entry S 0x1b 996000 \n```
flags 0x2

5. Restart the tsn-app application:

```bash
# avb.sh start
```

On the IO device(s):

1. Stop the application if it was already running:

```bash
# avb.sh stop
```

2. Edit the application configuration file:

```bash
# vi /etc/genavb/apps-tsn-network-iodevice.cfg
```

3. Use the "-p" option to change the period. The below example sets the period to 1 ms (1000000 ns):

```
CFG_EXTERNAL_MEDIA_APP_OPT="-m network_only -r iodevice_N -p 1000000"
```

4. Update the traffic schedule using tc.
   In the sample command below, replace ethX with the right TSN network interface:
   • eth1 on i.MX 8M Plus LPDDR4 EVK and i.MX 93 EVK
   • eth0 on i.MX 8DXL LPDDR4 EVK and i.MX 93 9x9 LPDDR4 QSB

```bash
# tc qdisc del dev ethX root
# tc qdisc replace dev ethX root taprio
# num_tc 5 \n  map 0 1 1 2 2 3 4 0 0 0 0 0 0 0 0 \n  queues 100 101 102 103 104 \n  base-time 750000 \n  sched-entry S 0x4 4000 \n  sched-entry S 0x1b 996000 \n  flags 0x2
```

5. Restart the tsn-app application:

```bash
# avb.sh start
```

On the bridge, update the Qbv schedule on all ports:

```bash
# tc qdisc del dev swp0 root
# tc qdisc del dev swp1 root
# tc qdisc del dev swp2 root
# tc qdisc replace dev swp0 root taprio
  num_tc 8 \n  map 0 1 2 3 4 5 6 7 \n  queues 100 101 102 103 104 105 106 107 \n  base-time 750000 \n  sched-entry S 0x20 20000 \n  sched-entry S 0xdf 980000 \n  flags 0x2
# tc qdisc replace dev swp1 root taprio
  num_tc 8 \n  map 0 1 2 3 4 5 6 7 \n  queues 100 101 102 103 104 105 106 107 \n  base-time 250000 \n  sched-entry S 0x20 20000 \n  sched-entry S 0xdf 980000 \n  flags 0x2
# tc qdisc replace dev swp2 root taprio
  num_tc 8 \n```
map 0 1 2 3 4 5 6 7 \\
queues 100 101 102 103 104 105 106 107 \\
basis-time 250000 \\
sched-entry S 0x20 20000 \\
sched-entry S 0xdf 980000 \\
flags 0x2

After that, the evaluation can follow the various use cases described previously with the default configuration: baseline operation, scheduled traffic evaluation with or without best-effort traffic.

**Note:**

An arbitrary low period might run into the scheduling limits of the systems, and result in errors in the tsn-app logs, as the systems may no longer be able to keep up with the requested pace.

### 6.2.3.6.5.7 Enabling AF_XDP sockets in TSN sample application

A new feature makes it possible to use AF_XDP sockets with the Linux tsn-app application, to take advantage of the lower latency offered by the AF_XDP path. The steps below describe how to reconfigure an i.MX8M Plus LPDDR4 EVK, i.MX 8DXL LPDDR4 EVK, i.MX 93 EVK or i.MX 93 9x9 LPDDR4 QSB board to use AF_XDP sockets.

1. Stop the application and TSN stack if they were already running:
   ```shell
   # avb.sh stop_all
   ```
2. Edit the application configuration file:
   ```shell
   # vi /etc/genavb/apps-tsn-network-controller.cfg
   ```
3. To enable AF_XDP mode, replace the line:
   ```shell
   CFG_EXTERNAL_MEDIA_APP_OPT="-m network_only -r controller"
   ```
   With:
   ```shell
   CFG_EXTERNAL_MEDIA_APP_OPT="-m network_only -r controller -x"
   ```
4. Attach the XDP program to the TSN interface. This step can be done at any time, even if the TSN sample application is still running with its default configuration, as long as it is done before restarting it in AF_XDP mode.
   ```shell
   # ip l set dev ethX xdp obj /lib/firmware/genavb/genavb-xdp.bin
   ```
5. Restart the tsn-app application in AF_XDP mode:
   ```shell
   # avb.sh start
   ```

After that, the evaluation can follow the various use cases described previously with the default configuration: baseline operation, scheduled traffic evaluation with or without best-effort traffic. Statistics should be similar to or better than the default configuration, except for traffic latencies: because AF_XDP currently cannot provide packet timestamps, traffic latencies display bogus values that should be ignored. The tables below summarize typical values (in µs), on a setup using a 1 ms period.

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched err (controller)</td>
<td>6</td>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>Processing time (controller)</td>
<td>10</td>
<td>13</td>
<td>19</td>
</tr>
</tbody>
</table>
Table 68. Timing statistics without any concurrent traffic…continued

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time (controller)</td>
<td>16</td>
<td>20</td>
<td>33</td>
</tr>
</tbody>
</table>

Table 69. Timing statistics with TX best-effort traffic

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched err (controller)</td>
<td>18</td>
<td>25</td>
<td>51</td>
</tr>
<tr>
<td>Processing time (controller)</td>
<td>21</td>
<td>26</td>
<td>52</td>
</tr>
<tr>
<td>Total time (controller)</td>
<td>42</td>
<td>51</td>
<td>108</td>
</tr>
</tbody>
</table>

Table 70. Timing statistics with RX best-effort traffic

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched err (controller)</td>
<td>7</td>
<td>9</td>
<td>34</td>
</tr>
<tr>
<td>Processing time (controller)</td>
<td>8</td>
<td>11</td>
<td>25</td>
</tr>
<tr>
<td>Total time (controller)</td>
<td>15</td>
<td>21</td>
<td>55</td>
</tr>
</tbody>
</table>

6.2.3.6.5.8 OPC UA server evaluation

The OPC UA server address is in this format: opc.tcp://<endpoint IP address>:4840/

Once connected, the server objects can be browsed and accessed. The same statistics described in the TSN example application logs are available as OPC UA objects. The OPC UA server traffic is classified as best effort and doesn't affect the time sensitive traffic.

See below screenshot using FreeOPCUA GUI client:
6.2.4 Configuration files

6.2.4.1 System

The system configuration file, located in `/etc/genavb/system.cfg`, lists system network interface names and PTP hardware clock device names. The default values are used if the configuration file or the option key are missing. The values in the installed file may also required to be updated to match the system configuration.

Table 71. Logical ports

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint interface</td>
<td>endpoint</td>
<td>eth0</td>
<td>Endpoint network interface name. Only valid for endpoint package, otherwise should be set to “off”</td>
</tr>
<tr>
<td>Bridge 0 interfaces</td>
<td>bridge_0</td>
<td>SJA1105P_p0, SJA1105P_p1, SJA1105P_p2, SJA1105P_p3, SJA1105P_p4*</td>
<td>Bridge 0 network interface names (comma separated). Only valid for bridge package, otherwise should be set to “off”</td>
</tr>
</tbody>
</table>
Table 72. Clock
This section lists clock device names.
Clocks names are either a PHC device name or a generic software clock (sw_clock). Local clock points to a PHC device, target clocks point to either:
• The same PHC device as local clock (gPTP time is reflected in the local clock)
• A generic software clock (in which case gPTP time is not reflected in the local clock).

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint gPTP domain 0 target clock</td>
<td>endpoint_gptp_0</td>
<td>/dev/ptp0</td>
<td>Endpoint clock for gPTP domain 0 target clock. Only valid for endpoint package.</td>
</tr>
<tr>
<td>Endpoint gPTP domain 1 target clock</td>
<td>endpoint_gptp_1</td>
<td>sw_clock</td>
<td>Endpoint clock for gPTP domain 1 target clock. Only valid for endpoint package.</td>
</tr>
<tr>
<td>Endpoint local clock</td>
<td>endpoint_local</td>
<td>/dev/ptp0</td>
<td>Endpoint clock for the local clock. Only valid for endpoint package.</td>
</tr>
<tr>
<td>Bridge gPTP domain 0 target clock</td>
<td>bridge_gptp_0</td>
<td>sw_clock</td>
<td>Bridge clock for gPTP domain 0 target clock. Only valid for bridge package.</td>
</tr>
<tr>
<td>Bridge gPTP domain 1 target clock</td>
<td>bridge_gptp_1</td>
<td>sw_clock</td>
<td>Bridge clock for gPTP domain 1 target clock. Only valid for bridge package.</td>
</tr>
<tr>
<td>Bridge local clock</td>
<td>bridge_local</td>
<td>/dev/ptp1</td>
<td>Bridge clock for the local clock. Only valid for bridge package.</td>
</tr>
</tbody>
</table>

6.2.4.2 gPTP
The gPTP general parameters as well as default domain (domain 0) parameters are defined in the following configuration files depending on the package used:
• Endpoint package: /etc/genavb/fgptp.cfg
• Bridge package: /etc/genavb/fgptp-br.cfg

To enable other domains, new configuration files must be created with the associated domain instance appended to the configuration file name e.g.:
• Endpoint package, domain 1: /etc/genavb/fgptp.cfg-1
• Bridge package, domain 1: /etc/genavb/fgptp-br.cfg-1

Attention:
By default the GenAVB/TSN gPTP stack is packaged with the general parameters configuration file (fgptp.cfg or fgptp-br.cfg) and a reference configuration for domain 1 (fgptp.cfg-1 or fgptp-br.cfg-1)

6.2.4.2.1 General
Profile
The gPTP stack can operate in two different modes known as 'standard' or 'automotive' profiles.
When the 'standard' profile is selected, the gPTP stack operates following the specifications described in IEEE 802.1AS. When the 'automotive' profile is selected, the gPTP stack operates following the specifications described in the AVnu AutoCDSFunctionalSpec_1.4 which is a subset of the IEEE 802.1AS standard optimized
for automotive applications. IEEE 802.1AS-2020 features are not available in 'automotive' profile (e.g. Multiple domains).

The automotive environment is unique in that it is a closed system. Every network device is known prior to startup and devices do not enter or leave the network, except in the case of failures. Because of the closed nature of the automotive network, it is possible to simplify and improve gPTP startup performance. Specifically, functions like election of a grand master and calculations of wire delays are tasks that can be optimized for a closed system.

**Reverse sync feature control**

The Reverse Sync feature (Avnu specification) should be used for test/evaluation purpose only. Usually, to measure the accuracy of the clock synchronization, the traditional approach is to use a 1 Pulse Per Second (1PPS) physical output. While this is a good approach, there may be cases where using a 1PPS output is not feasible. More flexible and fully relying on software implementation the Reverse Sync feature serves the same objective using the standard gPTP Sync/Follow-Up messages to relay the timing information, from the Slave back to the GM.

**Neighbor propagation delay threshold**

The parameter neighborPropDelayThresh defines the propagation time threshold, above which a port is not considered capable of participating in the IEEE 802.1AS protocol (see IEEE 802.1AS-2020 - 11.2.2 Determination of asCapable and asCapableAcrossDomains). If a computed neighborPropDelay exceeds neighborPropDelayThresh, then asCapable is set to FALSE for the port. This setting does not apply to Automotive profile where a link is always considered to be capable or running IEEE 802.1AS.

**IEEE 802.1AS-2011 Compatibility**

The parameter force_2011 defines if the gPTP Stack operates following the IEEE 802.1AS-2011 standard, i.e. disabling the IEEE 802.1AS-2020 specifics features such as Multiple Domain support. The use of this option may, in some cases, improve compatibility with gPTP equipment not supporting IEEE 802.1AS-2020 standard.

### Table 73. General parameters

**General configuration parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile</td>
<td>profile</td>
<td>&quot;standard&quot;</td>
<td>&quot;standard&quot; or &quot;automotive&quot;</td>
<td>Set fgptp main profile. &quot;standard&quot; - IEEE 802.1AS specs, &quot;automotive&quot; - AVnu automotive profile</td>
</tr>
<tr>
<td>Grandmaster ID</td>
<td>gm_id</td>
<td>&quot;0x0001f2ffe0025fe&quot;</td>
<td>64bits EUI format</td>
<td>Set static grandmaster ID in host order (used by automotive profile, ignored in case of standard profile)</td>
</tr>
<tr>
<td>Domains</td>
<td>domain_number</td>
<td>0 for default domain -1: for domains different from 0</td>
<td>-1 to 127</td>
<td>Disable (-1) or assign a gPTP domain number to a domain instance.</td>
</tr>
<tr>
<td>802.1AS-2011 mode</td>
<td>force_2011</td>
<td>no</td>
<td>&quot;no&quot; or &quot;yes&quot;</td>
<td>Set to &quot;yes&quot; to force 802.1AS-2011 standard, &quot;no&quot; to enable 802.1AS-2020 full support.</td>
</tr>
<tr>
<td>Log output level</td>
<td>log_level</td>
<td>info</td>
<td>crit, err, init, info, or dbg</td>
<td>Set this configuration to dbg to enable debug mode</td>
</tr>
<tr>
<td>Reverse sync feature control</td>
<td>reverse_sync</td>
<td>0</td>
<td>0 or 1</td>
<td>Set to 1 to enable reverse sync feature.</td>
</tr>
<tr>
<td>Reverse sync feature interval</td>
<td>reverse_sync_interval</td>
<td>112</td>
<td>32 to 10000</td>
<td>Reverse sync transmit interval in ms units</td>
</tr>
</tbody>
</table>
### General configuration parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neighbor propagation delay threshold</td>
<td>neighborPropDelayThresh</td>
<td>800</td>
<td>32 to 10</td>
<td>Neighbor propagation delay threshold expressed in ns</td>
</tr>
<tr>
<td>Statistics output interval</td>
<td>statsInterval</td>
<td>10</td>
<td>0 to 255</td>
<td>Statistics output interval expressed in seconds. Use 0 to disable statistics</td>
</tr>
</tbody>
</table>

[1] For domain instances other than 0, only domain_number is configurable in this section.

### Grandmaster parameters

This section defines the native Grand Master capabilities of a time-aware system (see IEEE 802.1AS-2020 - 8.6.2 PTP Instance attributes). Grand Master capabilities parameters are defined in the main configuration file for gPTP domain 0 (e.g. fgptp.cfg) and in the additional per domain configuration files for other domains (e.g. fgptp.cfg-1).

gmCapable defines if the time-aware system is capable of being a grandmaster. By default gmCapable is set to 1 as in standard profile operation the Grand Master is elected dynamically by the BMCA. In case of automotive profile gmCapable must be set on each AED node to match the required network topology (that is, within a given gPTP domain only one node must have its gmCapable property set to 1).

priority1, priority2, clockClass, clockAccuracy and offsetScaledLogVariance are parameters used by the Best Master Clock algorithm to determine which of the Grand Master capable node within the gPTP domain has the highest priority/quality. Note that the lowest value for these parameters matches the highest priority/quality.

### Automotive parameters

The static pdelay feature is used only if the gPTP stack operates in automotive profile configuration.
At init time the gPTP stack's configuration file is parsed and based on neighborPropDelay_mode the specified initial_neighborPropDelay is applied to all ports and used for synchronization until a pdelay response from the peer is received. This is done only if no previously stored pdelay is available from the nvrarn database specified by nvrarn_file. As soon as a pdelay response from the peer is received the 'real' pdelay value is computed, and used for current synchronization. An indication may then be sent via callback up to the OS-dependent layer. Upon new indication the Host may update its nvrarn database and the stored value will be used at next restart for the corresponding port instead of the initial_neighborPropDelay. The granularity at which pdelay change indications are sent to the Host is defined by the neighborPropDelay_sensitivity parameter.

In the gPTP configuration file the neighborPropDelay_mode parameter is set to 'static' by default, meaning that a predefined propagation delay is used as described above while pdelay requests are still sent to the network. The 'silent' mode behaves the same way as the 'static' mode except that pdelay requests are never sent at all to the network.

Optionally the neighborPropDelay_mode parameter can be set to standard forcing the stack to operate propagation delay measurements as specified in the 802.1AS specifications even if the automotive profile is selected.

(see AutoCDSFunctionalSpec-1_4 - 6.2.2 Persistent gPTP Values)

### Table 75. Automotive parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Value &amp; Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pdelay mode</td>
<td>neighborPropDelay_mode</td>
<td>static</td>
<td>'static', 'silent' or 'standard'</td>
<td>Defines pdelay mechanism used</td>
</tr>
<tr>
<td>Static pdelay value</td>
<td>initial_neighborPropDelay</td>
<td>250</td>
<td>0 to 10000</td>
<td>Predefined pdelay value applied to all ports. Expressed in ns.</td>
</tr>
<tr>
<td>Static pdelay sensitivity</td>
<td>neighborPropDelay_sensitivity</td>
<td>10</td>
<td>0 to 1000</td>
<td>Amount of ns between two pdelay measurements required to trigger a change indication. Expressed in ns.</td>
</tr>
<tr>
<td>Nvram file name</td>
<td>nvrarn_file</td>
<td>/etc/genavb/fgptp.nvram</td>
<td></td>
<td>Path and nvrarn file name.</td>
</tr>
</tbody>
</table>

### 6.2.4.2.4 Timing

Pdelay requests and Sync messages sending intervals have a direct impact on the system synchronization performance. To reduce synchronization time while optimizing overall system load, two levels of intervals are defined. The first level called 'Initial', defines the messages intervals used until pdelay values have stabilized and synchronization is achieved. The second level called 'Operational', defines the messages intervals used once the system is synchronized.

initialLogPdelayReqInterval and operLogPdelayReqInterval define the intervals between the sending of successive Pdelay_Req messages. initialLogSyncInterval and operLogSyncInterval define the intervals between the sending of successive Sync messages. initialLogAnnounceInterval defines the interval between the sending of successive Announce messages.

(see AutoCDSFunctionalSpec-1_4 - 6.2.1 Static gPTP Values, IEC-60802 section 5, 802.1AS-2020 sections 10.7 and 11.5)
Table 76. Timing parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Value and Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial pdelay request interval value</td>
<td>initialLogPdelayReqInterval</td>
<td>0</td>
<td>0 to 3</td>
<td>Set pdelay request initial interval between the sending of successive Pdelay_Req messages. Expressed in log2 unit (default 0 -&gt; 1s).</td>
</tr>
<tr>
<td>Initial sync interval value</td>
<td>initialLogSyncInterval</td>
<td>-3</td>
<td>-5 to 0</td>
<td>Set sync transmit initial interval between the sending of successive Sync messages. Expressed in log2 unit (default -3 -&gt; 125ms).</td>
</tr>
<tr>
<td>Initial announce interval value</td>
<td>initialLogAnnounceInterval</td>
<td>0</td>
<td>0 to 3</td>
<td>Set initial announce transmit interval between the sending of successive Announce messages. Expressed in log2 unit (default 0 -&gt; 1s).</td>
</tr>
<tr>
<td>Operational pdelay request interval value</td>
<td>operLogPdelayReqInterval</td>
<td>0</td>
<td>0 to 3</td>
<td>Set pdelay request transmit interval used during normal operation state. Expressed in log2 unit (default 0 -&gt; 1s).</td>
</tr>
<tr>
<td>Operational sync interval value</td>
<td>operLogSyncInterval</td>
<td>-3</td>
<td>-5 to 0</td>
<td>Set sync transmit interval used during normal operation state. Expressed in log2 unit (default -3 -&gt; 125ms).</td>
</tr>
</tbody>
</table>

6.2.4.2.5 PORTn

This section describes the settings per port where \( n \) represents the port index starting at \( n=1 \).

Table 77. Port related parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Value &amp; Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port role</td>
<td>portRole</td>
<td>disabled</td>
<td>'slave', 'master', 'disabled'</td>
<td>Static port role (ref. 802.1AS-2011, section 14.6.3, Table 10-1), applies to &quot;automotive&quot; profile only.</td>
</tr>
<tr>
<td>Ptp port enabled</td>
<td>ptpPortEnabled</td>
<td>1</td>
<td>0 or 1</td>
<td>Set to 1 if both time-synchronization and best master selection functions of the port should be used (ref. 802.1AS-2011, sections 14.6.4 and 10.2.4.12).</td>
</tr>
<tr>
<td>RX timestamp compensation</td>
<td>rxDelayCompensation</td>
<td>0</td>
<td>min=-100000 max=100000 (in ns units)</td>
<td>Compensation delay subtracted from receive timestamps.</td>
</tr>
<tr>
<td>TX timestamp compensation</td>
<td>txDelayCompensation</td>
<td>0</td>
<td>min=-100000 max=100000 (in ns units)</td>
<td>Compensation delay added to transmit timestamps.</td>
</tr>
<tr>
<td>Delay Mechanism</td>
<td>delayMechanism</td>
<td>P2P</td>
<td>'P2P' or 'COMMON_P2P'</td>
<td>Must be set to COMMON_P2P for all domains others than Domain 0. For Domain 0 the value can be either P2P or COMMON_P2P.</td>
</tr>
</tbody>
</table>
The following table lists the recommended Rx and Tx compensation values to be applied to the supported NXP boards for optimized gPTP synchronization.

### Table 78. PHY Delay Compensation Values

<table>
<thead>
<tr>
<th>Board Type</th>
<th>rxDelayCompensation</th>
<th>txDelayCompensation</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS1028ARDB</td>
<td>-274</td>
<td>349</td>
</tr>
<tr>
<td>i.MX 8M Plus EVK</td>
<td>-569</td>
<td>184</td>
</tr>
</tbody>
</table>

### 6.2.4.3 SRP

The SRP parameters are defined in the following configuration files, depending on the package used:

- **Endpoint**: `/etc/genavb/srp.cfg`
- **Bridge**: `/etc/genavb/srp-br.cfg`

The default values are used if the configuration file or the option key are missing. The values in the installed file may also require an update to match the system configuration.

### Table 79. SRP General

This section lists general SRP stack component parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log output level</td>
<td>log_level</td>
<td>info</td>
<td>crit, err, init, info, or dbg</td>
<td>Log level for the SRP stack component.</td>
</tr>
</tbody>
</table>

### Table 80. MSRP

This section lists MSRP parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Default value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>enabled</td>
<td>1</td>
<td>0-disabled, 1-enabled</td>
<td>Enable/disable MSRP at runtime.</td>
</tr>
</tbody>
</table>

### 6.2.5 Log files

Several log files are available at runtime to monitor the different stack components.

#### 6.2.5.1 gPTP Endpoint

Logs are stored in `/var/log/fgptp`.

- **Linux command**:
  
  ```
  # tail -f /var/log/fgptp
  ```

- If the stack is configured in automotive mode, then the log contains:
  
  ```
  Running fgptp in automotive profile on interface eth0
  ```

- Port Role, Port AS-capability and link Status are reported each time there is a change in the link state (link is 802.1AS capable or not) or upon Grand Master (GM) change. This information is also displayed regularly along with current synchronization and pdelay statistics for each of the enabled gPTP domain:

  ```
  Port(0) domain(0,0): role changed from DISABLED to SLAVE
  ...
  Port(0) domain(0,0): Slave - Link: Up - AS_Capable: Yes
  ```
• Selected Grand Master (GM) capabilities are reported upon new GM selection. Root Identity represents the clock ID of the currently selected GM. Priority1, Priority2, Class and Accuracy describe the clock quality of the selected GM. Finally, the Source Port Identity of the peer master port (e.g. the bridge port the local slave port is connected to). This information is displayed for each of the enabled gPTP domain:

```
domain(0,0) Grand master: root identity 00049fffe039e35
domain(0,0) Grand master: priority1 245 priority2
domain(0,0) Grand master: class 248 accuracy 248
domain(0,0) Grand master: variance 17258
domain(0,0) Grand master: source port identity 0001f2fffe0025fe, port number 2
```

• Synchronization State is reported upon GM selection (SYNCHRONIZED) or when no GM is detected (NOT SYNCHRONIZED). Synchronization Time expressed in ms represents the time it took for the local clock to reach synchronization threshold starting from the first SYNC message received. This information is displayed for each of the enabled domain.

```
Port(0) domain(0) SYNCHRONIZED – synchronization time (ms): 250
```

• Pdelay (propagation delay) and local clock adjustments are printed out every 5 seconds. PDelay is expressed in ns units and represents the one-way delay from the endpoint and its peer master. Correction is expressed in parts per billion and represents the frequency adjustment performed to the local clock. Offset is expressed in ns represents the resulting difference between the locally adjusted clock and the reference gPTP GrandMaster’s clock. (Min/Max/Avg and Variance are computed for both Correction and Offset statistics). PDelay is displayed only for Domain 0. Correction and Offset are displayed for each of the enabled domain.

```
Port 0 domain(0,0): Propagation delay (ns): 37.60 min 34 avg 36 max 45 variance 17
Port 0 domain(0,0): Correction applied to local clock (ppb): min -5603 avg 5572 max 5538 variance 148
Port 0 domain(0,0): Offset between GM and local clock (ns) min -12 avg 4 max 22 variance 111
... Port 0 domain(1,20): Correction applied to local clock (ppb): min 32074 avg 32314 max 32574 variance 17695
Port 0 domain(1,20): Offset between GM and local clock (ns) min -61 avg 3 max 70 variance 1149
```

• The following per port per domain statistics (32 bits counters) are printed out every 15 seconds on slave and master entities:

```
Table 81. Port statistics displayed on slave and master entities

Receive counters
<table>
<thead>
<tr>
<th>PortStatRxPkts</th>
<th>Number of gPTP packets received (ether type 0x88F7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortStatRxSyncCount</td>
<td>Number of SYNC packets received</td>
</tr>
<tr>
<td>PortStatRxSyncReceiptTimeouts</td>
<td>Number of SYNC packets receive timeout</td>
</tr>
<tr>
<td>PortStatRxFollowUpCount</td>
<td>Number of FOLLOW-UP packets received</td>
</tr>
<tr>
<td>PortStatRxAnnounce</td>
<td>Number of ANNOUNCE packets received</td>
</tr>
<tr>
<td>PortStatAnnounceReceiptTimeouts</td>
<td>Number of ANNOUNCE packets timeout</td>
</tr>
<tr>
<td>PortStatAnnounceReceiptDropped</td>
<td>Number of ANNOUNCE packets dropped by the entity</td>
</tr>
<tr>
<td>PortStatRxSignaling</td>
<td>Number of SIGNALING packets received</td>
</tr>
<tr>
<td>PortStatRxPdelayRequest</td>
<td>Number of PDELAY REQUEST packets received</td>
</tr>
<tr>
<td>PortStatRxPdelayResponse</td>
<td>Number of PDELAY RESPONSE packets received</td>
</tr>
</tbody>
</table>
```
Table 81. Port statistics displayed on slave and master entities...continued

<table>
<thead>
<tr>
<th>Counter Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortStatPdelayAllowedLostResponsesExceeded</td>
<td>Number of excess of allowed lost responses to PDELAY requests</td>
</tr>
<tr>
<td>PortStatRxPdelayResponseFollowUp</td>
<td>Number of PDELAY FOLLOW-UP packets received</td>
</tr>
<tr>
<td>PortStatRxErrEtype</td>
<td>Number of ether type errors (not 0x88F7)</td>
</tr>
<tr>
<td>PortStatRxErrPortId</td>
<td>Number or port ID errors</td>
</tr>
<tr>
<td>Transmit counters</td>
<td></td>
</tr>
<tr>
<td>PortStatTxPkts</td>
<td>Number of gPTP packets transmitted</td>
</tr>
<tr>
<td>PortStatTxSyncCount</td>
<td>Number of SYNC packets transmitted</td>
</tr>
<tr>
<td>PortStatTxFollowUpCount</td>
<td>Number of FOLLOW-UP packets transmitted</td>
</tr>
<tr>
<td>PortStatTxAnnounce</td>
<td>Number of ANNOUNCE packets transmitted</td>
</tr>
<tr>
<td>PortStatTxSignaling</td>
<td>Number of SIGNALING packets transmitted</td>
</tr>
<tr>
<td>PortStatTxPdelayReques</td>
<td>Number of PDELAY REQUEST packets transmitted</td>
</tr>
<tr>
<td>PortStatTxPdelayResponse</td>
<td>Number of PDELAY RESPONSE packets transmitted</td>
</tr>
<tr>
<td>PortStatTxPdelayResponseFollowUp</td>
<td>Number of PDELAY FOLLOW-UP packets transmitted</td>
</tr>
<tr>
<td>PortStatTxErr</td>
<td>Number of transmit errors</td>
</tr>
<tr>
<td>PortStatTxErrAlloc</td>
<td>Number of transmit packets allocation errors</td>
</tr>
<tr>
<td>Miscellaneous counters</td>
<td></td>
</tr>
<tr>
<td>PortStatAdjustOnSync</td>
<td>Number of adjustments performed upon SYNC received</td>
</tr>
<tr>
<td>PortStatMdPdelayReqSmReset</td>
<td>Number of reset of the PDELAY REQUEST state machine</td>
</tr>
<tr>
<td>PortStatMdSyncRcvSmReset</td>
<td>Number of reset of the SYNC RECEIVE state machine</td>
</tr>
<tr>
<td>PortStatHwTsRequest</td>
<td>Number of egress timestamp requests</td>
</tr>
<tr>
<td>PortStatHwTsHandler</td>
<td>Number of egress timestamp notification</td>
</tr>
<tr>
<td>PortStatNumSynchronizationLoss</td>
<td>Number of synchronization loss on the slave endpoint</td>
</tr>
<tr>
<td>PortStatNumNotAsCapable</td>
<td>Number of transitions from AS_Capable=TRUE to AS_Capable=FALSE</td>
</tr>
</tbody>
</table>

6.2.5.2 gPTP Bridge

Logs are stored in /var/log/fgptp-br.

- Linux command:
  ```
  # tail -f /var/log/fgptp-br
  ```

- The bridge stack statistics are similar to the endpoint stack ones except that they are reported for each of the external ports of the switch (Port 0 to 3) and also for the internal port connected to the endpoint stack (Port 4) in case of Hybrid setup.

- Pdelay (propagation delay) is printed only for Domain 0. Link status, AS capability and Port Role are printed out for each port and each gPTP domain.

Port 0 domain(0,0): Role: Disabled Link: Up AS_Capable: No neighborGtpCapable: No DelayMechanism: P2P
Port 1 domain(0,0): Role: Disabled Link: Up AS_Capable: No neighborGtpCapable: No DelayMechanism: P2P
Port 2 domain(0,0): Role: Disabled Link: Up AS_Capable: Yes neighborGptpCapable: Yes DelayMechanism: P2P
Port 2 domain(0,0): Propagation delay (ns): 433.98 min 425 avg 438 max 457 variance 87
Port 3 domain(0,0): Role: Disabled Link: Up AS_Capable: No neighborGptpCapable: Yes DelayMechanism: P2P
Port 4 domain(0,0): Role Master Link: Up AS_Capable: Yes neighborGptpCapable: Yes DelayMechanism: P2P
Port 4 domain(0,0): Propagation delay (ns): 433.98 min 425 avg 438 max 457 variance 87

Port 0 domain(1,20): Role: Disabled Link: Up AS_Capable: No neighborGptpCapable: No DelayMechanism: COMMON_P2P
Port 1 domain(1,20): Role: Disabled Link: Up AS_Capable: No neighborGptpCapable: No DelayMechanism: COMMON_P2P
Port 3 domain(1,20): Role: Disabled Link: Up AS_Capable: No neighborGptpCapable: No DelayMechanism: COMMON_P2P

6.2.5.3 SRP Bridge

Logs are stored in /var/log/tsn-br.

- Linux command:

  ```
  # tail -f /var/log/tsn-br | grep srp
  ```

- SRP protocol information is reported per port

  INFO srp  msrp_vector_add_event : port(0) domain(5, 2, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_add_event : port(0) domain(6, 3, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_add_event : port(1) domain(5, 2, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_add_event : port(1) domain(6, 3, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_add_event : port(2) domain(5, 2, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_add_event : port(2) domain(6, 3, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_add_event : port(4) domain(5, 2, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_add_event : port(4) domain(6, 3, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_handler : port(3) domain(5, 2, 2)
  MRP_ATTR_EVT_MT
  INFO srp  msrp_vector_handler : port(3) domain(6, 3, 2)
  MRP_ATTR_EVT_MT
  INFO srp  msrp_vector_handler : port(3) domain(5, 2, 2)
  MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_handler : port(3) domain(6, 3, 2)
  MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_handler : port(3) domain(5, 2, 2)
  MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_handler : port(3) domain(6, 3, 2)
  MRP_ATTR_EVT_JOINMT
  INFO srp  msrp_vector_add_event : port(3) domain(5, 2, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOININ
  INFO srp  msrp_vector_add_event : port(3) domain(6, 3, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOININ
  INFO srp  msrp_vector_add_event : port(3) domain(5, 2, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOININ
  INFO srp  msrp_vector_add_event : port(3) domain(6, 3, 2)
  MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOININ

### 6.2.5.4 TSN Endpoint example application

Logs are stored in `/var/log/tsn_app`.

The TSN application has various counters and statistics which help to validate:

- application scheduling and processing timing statistics
- network traffic correctness and latency statistics

Most of the information in the logs are either:

- **Counters**: single integer values counting specific events (frames received, transmitted, errors, etc)
- **Statistics**: composite data over a series of measurements: min (minimum during the last period), mean (average of measurements during the last period), max (maximum during the last period), rms (root mean square of measurements during the last period), stddev (square of standard deviation during the last period), absmin (absolute minimum since the application start), absmax (absolute maximum since the application start)
- **Histograms**: number and size of slots of the histogram one the 1st line, array of counters for each slot on the 2nd line.

#### 6.2.5.4.1 Main TSN task

The main TSN task logs are described below:

- **Scheduling counters** (*sched* should increment of 500 per second):

<table>
<thead>
<tr>
<th>INFO</th>
<th>Time</th>
<th>Task</th>
<th>Sched</th>
<th>Sched Early</th>
<th>Sched Late</th>
<th>Sched Missed</th>
<th>Sched Timeout</th>
<th>Clock Discont</th>
<th>Clock Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>1604531064</td>
<td>1604531064</td>
<td>tsn_task_stats_print</td>
<td>tsn_task (0x37d8d630)</td>
<td>1700000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
6.2.5.4.2 Network socket

Below is an example of the network socket logs:

- Low-level network socket. Only frames relevant to the network socket (Layer 2) are counted here:

```plaintext
INFO 1604531059 net_socket_stats_print net rx socket(0x37d8d660) 0
INFO 1604531059 net_socket_stats_print frames : 1697802
INFO 1604531059 net_socket_stats_print frames err : 0
INFO 1604531059 net_socket_stats_print net tx socket(0x37d8d6e0) 0
INFO 1604531059 net_socket_stats_print frames : 1697500
INFO 1604531059 net_socket_stats_print frames err : 0
```

6.2.5.4.3 Application socket

- Application header is checked at this level. Also, the timestamps from the remote peers are verified as well which guarantees that only expected and in sequence data is processed.

```plaintext
INFO 1604531069 socket_stats_print cyclic rx socket(0x419560) net_sock(0x37d8d660) peer id: 1
```
6.3 IEEE 1588/802.1AS

IEEE 1588 is the IEEE standard for a precision clock synchronization protocol for networked measurement and control systems.


NXP’s Layerscape platform provides hardware assist for 1588 compliant time stamping with the 1588 timer module to support applications of IEEE 1588/802.1AS.

6.3.1 Introduction

NXP’s i.MX and Layerscape platforms provide hardware assist for 1588 compliant time stamping with the 1588 timer module. The software components required to run IEEE 1588/802.1AS protocol utilizing the hardware feature are listed below:

1. Linux PTP Hardware Clock (PHC) driver
2. Linux Ethernet controller driver with hardware timestamping support
3. A software stack application for IEEE 1588/802.1AS

Note: In this document, IEEE 1588 mentioned is IEEE 1588-2008, and IEEE 802.1AS mentioned is IEEE 802.1AS-2011.

6.3.2 IEEE 1588 device types

There are five basic types of PTP devices in IEEE 1588.

• Ordinary clock

A clock that has a single Precision Time Protocol (PTP) port in a domain and maintains the timescale used in the domain. It may serve as a source of time (if used as a master clock) or may synchronize with another clock (if used as a slave clock).

• Boundary clock
A clock that has multiple Precision Time Protocol (PTP) ports in a domain and maintains the timescale used in the domain. It may serve as a source of time (be a master clock) or may synchronize to another clock (be a slave clock).

- **End-to-end transparent clock**
A transparent clock that supports the use of the end-to-end delay measurement mechanism between slave clocks and the master clock.

- **Peer-to-peer transparent clock**
A transparent clock that provides Precision Time Protocol (PTP) event transit time information. It also provides corrections for the propagation delay of the link connected to the port receiving the PTP event message. In the presence of peer-to-peer transparent clocks, delay measurements between slave clocks and the master clock are performed using the peer-to-peer delay measurement mechanism.

- **Management node**
A device that configures and monitors clocks.

**Note:** *Transparent clock is a device that measures the time taken for a PTP event message to transit the device. It provides this information to clocks receiving the PTP event message.*

### 6.3.3 IEEE 802.1AS time-aware systems

In gPTP, there are only two types of time-aware systems: end stations and Bridges, while IEEE 1588 has ordinary clocks, boundary clocks, end-to-end transparent clocks, and P2P transparent clocks. A time-aware end station corresponds to an IEEE 1588 ordinary clock, and a time-aware Bridge is a type of IEEE 1588 boundary clock where its operation is very tightly defined, so much so that a time-aware Bridge with Ethernet ports can be shown to be mathematically equivalent to a P2P transparent clock in terms of how synchronization is performed.

1. **Time-aware end station**
An end station that is capable of acting as the source of synchronized time on the network, or destination of synchronized time using the IEEE 802.1AS protocol, or both.

2. **Time-aware bridge**
A Bridge that is capable of communicating synchronized time received on one port to other ports, using the IEEE 802.1AS protocol.

### 6.3.4 Software stacks

#### 6.3.4.1 linuxptp stack

**Features of open source linuxptp**

- Supports hardware and software time stamping via the Linux SO_TIMESTAMPING socket option.
- Supports the Linux PTP Hardware Clock (PHC) subsystem by using the clock_gettime family of calls, including the clock_adjtimex system call.
- Implements Boundary Clock (BC), Ordinary Clock (OC) and Transparent Clock (TC).
- Transport over UDP/IPv4, UDP/IPv6, and raw Ethernet (Layer 2).
- Supports IEEE 802.1AS-2011 in the role of end station.
- Modular design allowing painless addition of new transports and clock servos.
• Implements unicast operation.
• Supports a number of profiles, including:
  – The automotive profile.
  – The default 1588 profile.
  – The enterprise profile.
  – Supports the NetSync Monitor protocol.
• Implements Peer to peer one-step.
• Supports bonded, IPoIB, and vlan interfaces.

Note: the features listed are from linuxptp website. It does not mean all these features work on release boards. The hardware 1588 capability, driver support and ptp4l version needs to be considered. Refer to following user manual of this chapter for what had been verified.

Features added by Real-time Edge
• Supports IEEE 802.1AS-2011 in the role of time-aware bridge.
• Support dynamic direction in ts2phc to cooperate with ptp4l.

6.3.4.2 NXP GenAVB/TSN gPTP stack
Following are the features of the NXP GenAVB/TSN gPTP stack:
• Implements gPTP IEEE 802.1AS-2020, for both time-aware Endpoint and Bridge systems
• Implements gPTP BMCA
• Supports GrandMaster, Master, and Slave capabilities
• Supports multiple gPTP domains
• Supports Avnu Alliance Automotive profile
• Supports configuration profiles for the stack
• Supports hardware time stamping via the Linux SO_TIMESTAMPING socket option
• Supports the Linux PTP Hardware Clock (PHC) subsystem by using the clock_gettime family of calls, including the clock_adjtimex system call.

6.3.5 Quick Start for IEEE 1588

6.3.5.1 Ordinary clock verification
Connect two network interfaces in back-to-back manner for two boards. Make sure there is no MAC address conflict on the boards, the IP addresses are set properly and ping the test network. Run linuxptp on each board. For example, eth0 is used on each board.

$ ptp4l -i eth0 -m

On running the above command time synchronization will start, and the slave linuxptp selected automatically will synchronize to master with synchronization messages displayed, such as time offset, path delay and so on. For example,

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptp4l[878.504]: master offset</td>
<td>-10 s2 freq -2508 path delay 1826</td>
</tr>
<tr>
<td>ptp4l[878.629]: master offset</td>
<td>-5 s2 freq -2502 path delay 1826</td>
</tr>
<tr>
<td>ptp4l[878.754]: master offset</td>
<td>0 s2 freq -2495 path delay 1826</td>
</tr>
<tr>
<td>ptp4l[878.879]: master offset</td>
<td>9 s2 freq -2482 path delay 1826</td>
</tr>
<tr>
<td>ptp4l[879.004]: master offset</td>
<td>-9 s2 freq -2507 path delay 1826</td>
</tr>
<tr>
<td>ptp4l[879.129]: master offset</td>
<td>-24 s2 freq -2530 path delay 1826</td>
</tr>
</tbody>
</table>
Some other options of ptp4l

<table>
<thead>
<tr>
<th>Delay Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-E</td>
<td>E2E, delay request-response (default)</td>
</tr>
<tr>
<td>-P</td>
<td>P2P, peer delay mechanism</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Transport</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>IEEE 802.3</td>
</tr>
<tr>
<td>-4</td>
<td>UDP IPV4 (default)</td>
</tr>
<tr>
<td>-6</td>
<td>UDP IPV6</td>
</tr>
</tbody>
</table>

Note: must keep same delay mechanism and network transport protocol used on two boards.

Configure master mode

In default, the master clock is selected by BMC (Best Master Clock) algorithm. To appoint a specific clock as master, a lower "priority1" attribute value than the other clock can be set. Lower value takes precedence. For example, in current case, specify one clock as master with below option. (The other clock is using default priority1 value 128.)

```
--priority1=127
```

One-step timestamping

Currently one-step timestamping is supported only on DPAA2. To use one-step timestamping, add below option for ptp4l running.

```
--twoStepFlag=0
```

6.3.5.2 Boundary clock verification

At least three boards are needed. Below is an example for three boards network connection. Make sure there is no MAC address conflict on the boards, the IP addresses are set properly and ping the test network.
Run `linuxptp` on Board1 (boundary clock).

```
$ ptp4l -i eth0 -i eth1 -m
```

Run `linuxptp` on Board2/Board3 (ordinary clock).

```
$ ptp4l -i eth0 -m
```

On running the above command, time synchronization starts, and the slaves `linuxptp` selected automatically synchronizes to the unique master with synchronization messages displayed such as time offset, path delay and so on. For example,

```
ptp4l[878.504]: master offset        -10 s2 freq   -2508 path delay      1826
ptp4l[878.629]: master offset         -5 s2 freq   -2502 path delay      1826
ptp4l[878.754]: master offset          0 s2 freq   -2495 path delay      1826
ptp4l[878.879]: master offset          9 s2 freq   -2482 path delay      1826
ptp4l[879.004]: master offset         -9 s2 freq   -2507 path delay      1826
ptp4l[879.129]: master offset        -24 s2 freq   -2530 path delay      1826
ptp4l[879.255]: master offset        -7 s2 freq   -2508 path delay      1826
ptp4l[879.380]: master offset        -2 s2 freq   -2502 path delay      1826
ptp4l[879.505]: master offset        -17 s2 freq   -2524 path delay      1827
ptp4l[879.630]: master offset         6 s2 freq   -2493 path delay      1827
ptp4l[879.755]: master offset         6 s2 freq   -2492 path delay      1827
ptp4l[879.880]: master offset          0 s2 freq   -2500 path delay      1827
```

Some other options of `ptp4l`

<table>
<thead>
<tr>
<th>Delay Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-E</td>
<td>E2E, delay request-response (default)</td>
</tr>
<tr>
<td>-P</td>
<td>P2P, peer delay mechanism</td>
</tr>
<tr>
<td>Network Transport</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>IEEE 802.3</td>
</tr>
<tr>
<td>-4</td>
<td>UDP IPV4 (default)</td>
</tr>
</tbody>
</table>
Note: You must keep same delay mechanism and use same network transport protocol on these boards.

Configure master mode

In default, the master clock is selected by BMC (Best Master Clock) algorithm. To appoint a specific clock as master, a lower "priority1" attribute value than the other clock can be set. Lower value takes precedence. For example, in current case, specify one clock as master with below option. (The other clocks is using default priority1 value 128.)

```
--priority1=127
```

One-step timestamping

Currently one-step timestamping is supported only on DPAA2. To use one-step timestamping, add below option for ptp4l running.

```
--twoStepFlag=0
```

### 6.3.5.3 Transparent clock verification

At least three boards are needed. Below is an example for three boards network connection. Make sure there is no MAC address conflict on the boards, the IP addresses are set properly, and ping the test network.

![Sample network connection of three boards](image)

Run `linuxptp` on Board1 (transparent clock). If you want Board1 works as E2E TC, use `E2E-TC.cfg`. If you want Board1 works as P2P TC, use `P2P-TC.cfg`.

```
$ ptp4l -i eth0 -i eth1 -f /etc/linuxptp/E2E-TC.cfg -m
```

Run `linuxptp` on Board2/Board3 (ordinary clock).
On running the above commands, time synchronization starts between ordinary clocks, and the slave linuxptp selected automatically synchronizes to the master with synchronization messages displayed such as time offset, path delay and so on.

6.3.6 Quick Start for IEEE 802.1AS

The following sections describe the steps for implementing IEEE 802.1AS on NXP boards. The following steps make use of linuxptp stack but similar commands can be executed with NXP GenAVB/TSN gPTP stack on supported boards, as described here.

6.3.6.1 Time-aware end station verification

Connect two network interfaces in back-to-back way for two boards. Make sure no MAC address conflict on the boards, IP address set properly and ping test work.

Remove below option in /etc/linuxptp/gPTP.cfg to use default larger value, because estimate path delay including PHY delay may exceed 800ns since hardware is using MAC timestamping.

```plaintext
neighborPropDelayThresh 800
```

Run linuxptp on each board. For example, eth0 is used on each board.

```plaintext
$ ptp4l -i eth0 -f /etc/linuxptp/gPTP.cfg -m
```

Time synchronization will start, and the slave linuxptp selected automatically will synchronize to master with synchronization messages printed, like time offset, path delay and so on.

6.3.6.2 Time-aware bridge verification

At least three boards are needed for the time-aware bridge verification. Below is an example of the network connection amongst the three boards. Make sure there is no MAC address conflict on the boards.

![Figure 87. Setup for time-aware bridge verification](image)
Remove the below option in `/etc/linuxptp/gPTP.cfg` file to use the default larger value, because estimated path delay including PHY delay may exceed 800 ns since hardware is using MAC timestamping.

```
neighborPropDelayThresh 800
```

Run `linuxptp` on Board1 (time-aware bridge) using the command below:

```
$ ptp4l -i eth0 -i eth1 -f /etc/linuxptp/gPTP.cfg -m
```

Run `linuxptp` on Board2/Board3 (time-aware end station) using the command:

```
$ ptp4l -i eth0 -f /etc/linuxptp/gPTP.cfg -m
```

Time synchronization will start between the three boards, and the `linuxptp` slaves selected will automatically synchronize to the unique master with synchronization messages displayed (such as time offset, path delay and so on).

### 6.3.7 Long term test

This section describes the long term test results for Linux PTP stack implementation.

### 6.3.8 Known issues and limitations

1. When LS1028A TSN switch in Linux is configured as L2 switch, the interfaces should not be configured with IP addresses. Running `linuxptp` on these interfaces must use Ethernet protocol instead of UDP/IP. The method is to add an option `-2` executing `ptp4l` command. For example,

```
$ ptp4l -i eth0 -2 -m
```

2. i.MX 8M Plus current `dwmac` driver (eth1) initializes some hardware functions during opening net device, including PTP initialization. Before that, the operations on it may not work, like `ethtool` queries, and PTP operations. So, the workaround is, do operations on the `eth1` and PTP of `dwmac` only after `"ifconfig eth1 up"`.

3. If below error is reported during `ptp4l` running, just try to increase `tx_timestamp_timeout`. User space may need to wait longer for TX timestamp.

For example, use option `--tx_timestamp_timeout=20` when running `ptp4l` as shown below:

```
ptp4l[1560.726]: timed out while polling for tx timestamp
ptp4l[1560.726]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a driver bug
```

### 6.4 Networking

#### 6.4.1 Q-in-Q on LS1028A Felix switch

1. **Q-in-Q feature**

   Q-in-Q feature allows service providers to create a Layer 2 Ethernet connection between two user sites. Providers can segregate VLAN traffic of different users on a link or bundle using different user VLANs.
When using Q-in-Q, the service VLAN tag (S-TAG: 0x88A8) prepend the 802.1Q VLAN tags (C-TAG:0x8100) of the user.

2. Q-in-Q application scenario
   In the following scenario, port swp0 of the switch connects with Customer 1's LAN, swp1 connects with the MAN of the ISP.
   The traffic with VLAN tag is shown below:
   **uplink**: Customer LAN (only C-TAG) -> swp0 -> swp1 (add S-TAG) -> ISP MAN (S-TAG + C-TAG)
   **downlink**: ISP MAN (S-TAG + C-TAG) -> swp1 (pop S-TAG) -> swp0 (only C-TAG) -> Customer LAN

3. Q-in-Q configuration example
   a. Enable swp1 Q-in-Q mode
      ```bash
      devlink dev param set pci/0000:00:00.5 name qinq_port_bitmap value 2 cmode runtime
      
      Note:
      • 0000:00:00.5 is the PCIe bus and device number of ocelot switch.
      • The value 2 is bitmap for port 1. If port n is linked to ISP MAN, the related bit ‘n’ should be set to 1.
   b. Create bridge and add ports:
      ```bash
      ip link add dev br0 type bridge vlan_protocol 802.1ad
      ip link set dev swp0 master br0
      ip link set dev swp1 master br0
      ip link set dev br0 type bridge vlan_filtering 1
      ```
   c. Set swp0 pvid and set untagged for egress traffic:
      ```bash
      bridge vlan del dev swp0 vid 1 pvid
      bridge vlan add dev swp0 vid 100 pvid untagged
      bridge vlan add dev swp1 vid 100
      ```
6.4.2 VCAP on LS1028A Felix switch

The VCAP is a content-aware packet processor for wire-speed packet inspection. It uses the 'tc flower' command to set the filter and actions. Following keys and actions are supported on LS1028A:

**keys:** vlan_id vlan_prio dst_mac/src_mac for non IP frames dst_ip/src_ip dst_port/src_port

**actions:** trap drop police vlan modify vlan push(Egress)

Use the following commands to set, get, and delete VCAP rules:

```
tc qdisc add dev swp0 clsact
tc filter add dev swp0 ingress chain [chain-id] protocol [ip/802.1Q] flower skip_sw [keys] action [actions]
tc filter show dev swp0 ingress chain [chain-id]
tc filter del dev swp0 ingress chain [chain-id] pref [pref_id]
tc qdisc add dev swp1 clsact
tc filter add dev swp1 egress protocol 802.1Q flower skip_sw [keys] action vlan push id [value] priority [value]
tc filter show dev swp1 egress
tc filter del dev swp1 egress pref [pref_id]
```

There are two ingress VCAPs and one egress VCAPs. The tc-flower chains are used on LS1028A ingress ports. Each action has a fixed chain. Following is the chain allocation:

<table>
<thead>
<tr>
<th>chain ID</th>
<th>Actions</th>
<th>Hardware module</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>skbedit priority</td>
<td>IS1 lookup 0</td>
<td>Source MAC address, source IP address (32 bits) outer VLAN, IP protocol, source TCP/UDP ports.</td>
</tr>
<tr>
<td>11000</td>
<td>vlan pop; vlan modify</td>
<td>IS1 lookup 1</td>
<td>Inner and outer VLAN, source and destination IP addresses (32 bits), IP protocol, source and destination TCP/UDP ports.</td>
</tr>
<tr>
<td>12000</td>
<td>goto chain [PAG]</td>
<td>IS1 lookup 2</td>
<td>Source MAC address, source IP address (32 bits) outer VLAN, IP protocol, source TCP/UDP ports.</td>
</tr>
<tr>
<td>20000-20255</td>
<td>police; trap</td>
<td>IS2 lookup 0</td>
<td>Source and destination MAC address, source and destination IP addresses (32 bits), IP protocol, source and destination TCP/UDP ports.</td>
</tr>
<tr>
<td>21000-21255</td>
<td>drop; redirect</td>
<td>IS2 lookup 1</td>
<td>Source and destination MAC address, source and destination IP addresses (32 bits), IP protocol,</td>
</tr>
</tbody>
</table>
Table 82. Chain allocation...continued

<table>
<thead>
<tr>
<th>chain ID</th>
<th>Actions</th>
<th>Hardware module</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>30000</td>
<td>gate; police</td>
<td>PSFP</td>
<td>destination MAC address and VLAN ID</td>
</tr>
</tbody>
</table>

Before using chains, users should register each chain and set chain pipeline order for a packet. The hardware ingress order is: IS1->IS2->PSFP.

```bash
tc qdisc add dev swp0 clsact
for each chain, add filter rules as follows:
tc filter add dev swp0 ingress chain 0 pref 49152 flower skip_sw action goto chain 10000
```

After registering the chain, add rules to the corresponding chain. Following are the use cases for testing:

```
tc filter add dev swp0 ingress chain 10000 pref 49152 flower skip_sw action goto chain 11000
```

Figure 89. VCAP test
1. Drop all frames from source IP 192.168.2.1.

   tc filter add dev swp0 ingress chain 21000 protocol ip flower skip_sw src_ip 192.168.2.1 action drop

   Set source IP as 192.168.2.1 and send IP package from TestCenter, package will be dropped on swp0.

2. Limit bandwidth of HTTP streams to 10 Mbps.

   tc filter add dev swp0 ingress chain 20000 protocol ip flower skip_sw ip_proto tcp dst_port 80 action police rate 10Mbit burst 10000 conform-exceed drop/pipe action goto chain 21000

   Send TCP package and set destination port as 80 on TestCenter, set the stream bandwidth to 1Gbit/s, we can get a 10Mbits/s stream rate.

3. Filter frames that have a specific vlan tag (VID=1 and PCP=1). Then, modify the vlan tag (VID=2, PCP=2) and classified to QoS traffic class 2.

   ip link set switch type bridge vlan_filtering 1
   tc filter add dev swp0 ingress chain 11000 protocol 802.1Q flower skip_sw vlan_id 1 vlan_prio 1 action vlan modify id 2 priority 2 action goto chain 12000
   bridge vlan add dev swp0 vid 2
   bridge vlan add dev swp1 vid 2

   Set vid=1 and pcp=1 in vlan tag. Then, send IP package from TestCenter. Thus you can get a package with vid=2, pcp=2 from swp1 on TestCenter.

4. Push a specific vlan tag (vid=3, pcp=3) into frames (classified vid=2, pcp=2 in switch) egress from swp1.

   tc qdisc add dev swp1 clsact
   tc filter add dev swp1 egress protocol 802.1Q flower skip_sw vlan_id 2 vlan_prio 2 action vlan push id 3 priority 3

   Set vid=1 and pcp=1 in vlan tag, then send IP package from TestCenter, the frame will hit rule in usecase 3 and retag the vlan (vid=2, pcp=2). Thus, users can get a frame with vid=3, pcp=3 from swp1 on TestCenter.

5. Push double vlan tag(Q-in-Q) into frames egress to swp1.

   ip link add dev br0 type bridge
   ip link set dev swp0 master br0
   ip link set dev swp1 master br0
   ip link set br0 type bridge vlan_filtering 1
   bridge vlan add dev swp0 vid 222
   bridge vlan add dev swp1 vid 222
   tc qdisc add dev swp1 clsact
   tc filter add dev swp1 egress protocol 802.1Q flower skip_sw vlan_id 222 vlan_prio 2
   action vlan push id 200 priority 1 protocol 802.1AD
   action vlan push id 300 priority 3

   Result: TX(tpid:8100 vid:222 pri:2) -> swp0 -> swp1 -> RX(S-TAG tpid:88A8 vid:200 pri:1, C-TAG tpid:8100 vid:300 pri:3)

6. Pop single or double vlan tag(Q-in-Q) from frames ingress from swp0.

   ip link add dev br0 type bridge
   ip link set dev swp0 master br0
   ip link set dev swp1 master br0
   tc filter add dev swp0 ingress chain 11000 protocol 802.1ad flower
   vlan_id 111 vlan_prio 1 vlan_ethtype 802.1q
   cvlan_id 222 cvlan_prio 2 cvlan_ethtype ipv4
   action vlan pop action goto chain 12000
**Result:** TX(S-TAG tpid:88A8 vid:111 pri:1, C-TAG tpid:8100 vid:222 pri:2) -> swp0
-> swp1 -> RX(TAG tpid:8100 vid:222 pri:2)

tc filter add dev swp0 ingress chain 11000 
protocol 802.1ad flower 
  vlan_id 111 vlan_prio 1 vlan_ethtype 802.1q 
  cvlan_id 223 cvlan_prio 2 cvlan_ethtype ipv4 
action vlan pop 
  action vlan pop action goto chain 12000

**Result:** TX(S-TAG tpid:88A8 vid:111 pri:1, C-TAG tpid:8100 vid:223 pri:2) -> swp0
-> swp1 -> RX(received packets without VLAN tag)
7 Protocols

This section describes the protocols supported.

7.1 EtherCAT master

Real-time Edge supports the usage of EtherCAT (Ethernet for Control Automation Technology) master and integrates the IGH EtherCAT master stack and SOEM. EtherCAT is verified on NXP platforms.

7.1.1 Introduction

EtherCAT is an Ethernet-based fieldbus system, invented by BECKHOFF Automation. The protocol is standardized in IEC 61158 and is suitable for both hard and soft real-time computing requirements in automation technology. EtherCAT was developed with an objective to apply Ethernet for automation applications requiring short data update times, low communication jitter, and reduced hardware costs.

Note: (Data update time is also called cycle time; low communication jitter implies less than 100 µs and for precise synchronization purposes, it is lesser than 1 µs).

- EtherCAT is Fast: 1000 dig. I/O: 30 µs, 100 slaves: 100 µs.
- EtherCAT is Ethernet: Standard Ethernet at I/O level.
- EtherCAT is Flexible: Star, line, drop, with or without switch.
- EtherCAT is Inexpensive: Ethernet is mainstream technology, therefore inexpensive.
- EtherCAT is Easy: everybody knows Ethernet, it is simple to use.

Currently, there are two open source EtherCAT masters, IGH for Cortex-A core and SOEM for Cortex-M core, which Real-time Edge supports. Both IgH and SOEM are solutions that help users get rid of the low-level development directly on EtherCAT protocol, and provide a common API for real-time applications. For more information, see https://rt-labs.com/ethercat/ and http://www.etherlab.org.

As integrated into the Preempt-RT Linux kernel and using native Ethernet drivers (only supported on a few specific NXP platforms), IgH shows significantly superior real-time characteristics. Also, IgH is more powerful and integrated by providing users with auxiliary functions such that it encourages users to focus on specific high-level programs. For example, IgH runs with an automatic master FSM (Finite State Machine), which is a EtherCAT slave manager dynamically adapting the slave configurations to the new topology and responding to requests from application-layer. In addition, IgH provides a command-line tool in user space to display detailed information about master/slave configurations and to list SDO dictionaries and reading/writing addresses.

SOEM (Simple Open EtherCAT master) is a C library that offers methods to send and receive EtherCAT frames. It is very light-weight but still powerful and stable. It can run on multiple operating systems, for example, Linux, Windows, or FreeRTOS. It can even run without an operating system (such as 'baremetal'). Unlike IgH, users have to configure PDOs and SDOs aligned with determined memory addresses in their own applications, which might cause uncertainties during development. On Real-time Edge software, SOEM is only supported on Cortex-M core on i.MX 8M Mini LPDDR4 EVK and i.MX 8M Plus LPDDR4 EVK platforms and runs on FreeRTOS or baremetal.

7.1.2 EtherCAT protocol

Following are the characteristics of the EtherCAT protocol:

- The EtherCAT protocol is optimized for process data and is transported directly within the standard IEEE 802.3 Ethernet frame using Ethertype 0x88a4.
- The data sequence is independent of the physical order of the nodes in the network; addressing can be in any order.
Broadcast, multicast, and communication between slaves is possible, but must be initiated by the master device.

If IP routing is required, the EtherCAT protocol can be inserted into UDP/IP datagrams. This also enables any control with Ethernet protocol stack to address EtherCAT systems.

It does not support shortened frames.

The Figure 90 shows the EtherCAT frame structure.

![EtherCAT frame structure diagram](image)

### 7.1.3 IGH EtherCAT architecture

The components of the master environment are described below:

- **Master module:** This is the kernel module containing one or more EtherCAT master instances, the ‘Device Interface’ and the ‘Application Interface’.

- **Device modules:** These are EtherCAT-capable Ethernet device driver modules that offer their devices to the EtherCAT master via the device interface. These modified network drivers can handle network devices used for EtherCAT operation and ‘normal’ Ethernet devices in parallel. A master can accept a certain device and then, is able to send and receive EtherCAT frames. Ethernet devices declined by the master module are connected to the kernel's network stack, as usual.

- **Application:** A program that uses the EtherCAT master (usually for cyclic exchange of process data with EtherCAT slaves). These programs are not part of the EtherCAT master code, but require to be generated or written by the user. An application can request a master through the application interface. If this succeeds, it has the control over the master: It can provide a bus configuration and exchange process data. Applications can be kernel modules that use the kernel application interface directly. They also include user space programs, which use the application interface via the EtherCAT library or the RTDM library.

The Figure 91 shows the IGH EtherCAT master architecture.
7.1.3.1 IGH EtherCAT device drivers

The EtherCAT protocol is based on the Ethernet standard, so a master relies on standard Ethernet hardware to communicate with the bus. The term device is used as a synonym for Ethernet network interface hardware. There are two kinds of device drivers modules:

1. **Native Ethernet Device Drivers**

Native Ethernet Device Drivers allow the EtherCAT master direct and exclusive access to the Ethernet hardware. This implies that the network device must not be connected to the kernel's stack as usual, which allows a high Real-time performance. In Real-time Edge software, there are three native Ethernet drivers:

- **ec_fec**: the native driver `ec_fec` can be used for the FEC MAC on i.MX 8DXX LPDDR4 EVK, i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, and i.MX 93 EVK. This driver is not verified on other i.MX
platforms in this release. Please note that the original Ethernet fec driver must be recompiled to a module by reconfiguring Linux with command "make menuconfig" when using ec_fec native driver.

- ec_enetc: the native driver "ec_enetc" is used for ENETC MAC on the LS1028ARDB platform.
- ec_dpaa1: the native driver "ec_dpaa1" is used for DPAA1 MAC on the LS1043ARDB and LS1046ARDB.

2. Generic Ethernet Device Driver

The Generic driver uses the lower layers of the Linux network stack to connect to the hardware, independently of the actual hardware driver. So it can be used by all the hardware platforms that Real-time Edge supports. However, the performance by using generic Ethernet Device driver is a bit worse than the native driver, because the Ethernet frame data has to traverse the Linux network stack.

7.1.3.2 IGH EtherCAT setup

Before the IGH EtherCAT daemon starts, the Ethernet device and the Ethernet driver must be specified by setting the "MASTER0_DEVICE" and "DEVICE_MODULES" variables in the "/etc/ethercat.conf" file.

7.1.3.2.1 Specifying the Ethernet device

The Ethernet device is specified by setting MASTER0_DEVICE variable to the MAC address of the Ethernet device to use as the EtherCAT network interface as below:

```
MASTER0_DEVICE="00:04:9f:07:11:a6"
```

For LS1046ARDB or LS1043ARDB platforms, if multiple masters are required, adding a non-empty variable MASTER1_DEVICE creates a second master, and so on.

7.1.3.2.2 Generic Ethernet driver

The generic Ethernet driver is enabled on Real-time Edge images by default for all platforms. It can be specified by setting "DEVICE_MODULES" variable to "generic" as shown below in the "/etc/ethercat.conf" file.

```
DEVICE_MODULES="generic"
```

7.1.3.2.3 Native Ethernet driver for i.MX 8M Mini LPDDR4 EVK

The native Ethernet driver "ec_fec" is enabled on Real-time Edge images by default for i.MX 8M Mini LPDDR4 EVK. And it can be specified by setting "DEVICE_MODULES" variable to "fec" as below on "/etc/ethercat.conf" file.

```
DEVICE_MODULES="fec"
```

But note that the original Ethernet fec driver must be compiled to modules by reconfiguring Linux menuconfig when the native Ethernet driver "ec_fec" is enabled. On Real-time Edge, the original fec Ethernet driver has been configured as a module by default.

7.1.3.2.4 Native Ethernet Driver for i.MX 8M Plus LPDDR4 EVK and i.MX 93 EVK

The native Ethernet driver 'ec_fec' is not enabled on Real-time Edge images by default for i.MX 8M Plus LPDDR4 EVK and i.MX 93 EVK. It can be enabled manually before building the Real-time Edge images by following the steps listed below:

1. Reconfigure the original Ethernet fec driver to modules:
Similar to i.MX 8M Mini LPDDR4 EVK, the original ENET fec driver must be compiled to modules when the native Ethernet driver 'ec_fec' is enabled. The native Ethernet driver 'ec_fec' can be reconfigured by adding the below line on "source/meta-real-time-edge/conf/distro/include/real-time-edge-base.inc" file.

```
# for i.MX 8M Plus LPDDR4 EVK
DELTA_KERNEL_DEFCONFIG:append:mx8mp-nxp-bsp = " linux-fec.config"

# for i.MX 93 EVK
DELTA_KERNEL_DEFCONFIG:append:mx93-nxp-bsp = " linux-fec.config"
```

2. **Enable the native Ethernet driver 'ec_fec' for i.MX 8M Plus LPDDR4 EVK or i.MX 93 EVK:**
   The native Ethernet driver 'ec_fec' can be enabled by adding the below line on "source/meta-real-time-edge/conf/distro/include/igh-ethercat.inc" file for i.MX 8M Plus LPDDR4 EVK.

```
# for i.MX 8M Plus LPDDR4 EVK
IGH_ETHERCAT:imx8mp-lpddr4-evk = " fec "

# for i.MX 93 EVK
IGH_ETHERCAT:imx93evk = " fec "
```

**Note:** In the above code examples, that the interface names of ENET and ENET_QOS are exchanged because modules are loaded later than built-in drivers. It means that the ENET would be renamed "eth1". In the meantime, the ENET_QOS interface would be renamed to "eth0" from "eth1". This change might break some existing scripts that use "eth1" for ENET_QOS.

7.1.3.2.5 Native Ethernet driver for LS1028ARDB

The native Ethernet driver ec_enetc is enabled on Real-time Edge images by default for LS1028ARDB. It can be specified by setting DEVICE_MODULES variable to enetc as shown in the below /etc/ethercat.conf file.

```
DEVICE_MODULES="enetc"
```

7.1.3.2.6 Native Ethernet driver for LS1043ARDB and LS1046ARDB

The native Ethernet driver 'ec_dpaa1' is enabled by default for LS1046ARDB and LS1043ARDB platforms. There are up to 7 Ethernet ports on LS1046ARDB or LS1043ARDB. Hence, the multiple master and redundancy features can be supported on LS1046ARDB and LS1043ARDB platforms.

DPAA is a network co-processor that is more complex than ordinary network cards and is different from other native drivers mentioned above. So, the native driver "ec_dpaa1" must be configured by "ethercat_port" variable in U-Boot to notify the DPAA co-processor how to schedule data frames before EtherCAT stack starting. The format of "ethercat_port" variable is as below:

```
ethercat_port=master<x>._device,master<x>._backup,core_index;
```

The master<x>._device variable specifies the main Ethernet port for master with index 'x', while the master<x>._backup variable specifies the backup Ethernet port if redundancy is required. Every master must be bundled a specified CPU core, and core_index variable is used to specific which CPU core to be bundled for this master.

The relationship between Ethernet port name on LS1046ARDB chassis and the name in the Linux is in the Table 83:
The relationship between Ethernet port name on LS1043ARDB chassis and the name in the Linux is shown in Table 84.

### Table 83. LS1046ARDB chassis

<table>
<thead>
<tr>
<th>Port name on chassis</th>
<th>Port name in Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGMII1</td>
<td>fm1-mac3</td>
</tr>
<tr>
<td>RGMII2</td>
<td>fm1-mac4</td>
</tr>
<tr>
<td>SGMII1</td>
<td>fm1-mac5</td>
</tr>
<tr>
<td>SGMII2</td>
<td>fm1-mac6</td>
</tr>
<tr>
<td>10G Copper</td>
<td>fm1-mac9</td>
</tr>
<tr>
<td>10G SEP+</td>
<td>fm1-mac10</td>
</tr>
</tbody>
</table>

Figure 92. LS1046ARDB Ethernet chassis

There are 3 instances of ec_dpaa1 configuration for LS1046ARDB.

### Table 84. LS1043ARDB chassis

<table>
<thead>
<tr>
<th>Port name on chassis</th>
<th>Port name in Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSGMII.P0</td>
<td>fm1-mac1</td>
</tr>
<tr>
<td>QSGMII.P1</td>
<td>fm1-mac2</td>
</tr>
<tr>
<td>QSGMII.P2</td>
<td>fm1-mac5</td>
</tr>
<tr>
<td>QSGMII.P3</td>
<td>fm1-mac6</td>
</tr>
<tr>
<td>RGMII1</td>
<td>fm1-mac3</td>
</tr>
<tr>
<td>RGMII2</td>
<td>fm1-mac4</td>
</tr>
<tr>
<td>10G Copper</td>
<td>fm1-mac9</td>
</tr>
</tbody>
</table>

Figure 93. LS1043ARDB Ethernet chassis
1. There is only one network topology, and no backup port for redundancy. The Ethernet port of this network is RGMII1, and the master is bundled to CPU core 3. For this case, the `ethercat_port` variable is as below:

```
setenv ethercat_port "fm1-mac3,,3;"
```

2. There are two EtherCAT network topologies, and each network topology has a backup port for redundancy. The main port of the first network is RGMII1, the backup port is RGMII2, and the master of this network is bundled to CPU core 3. The main port of the other network is SRGMII1, the backup port is SRGMII2, and the master of this network is bundled to CPU core 2. For this case, the `ethercat_port` variable is as below:

```
setenv ethercat_port "fm1-mac3,fm1-mac4,3;fm1-mac5,fm1-mac6,2;"
```

3. There are two EtherCAT network topologies, and only the first network topology has a backup port for redundancy. The main port of the first network is RGMII1, the backup port is RGMII2, and the master of this network is bundled to CPU core 3. The main port of the other network is SRGMII1, and the master of this network is bundled to CPU core 2. For this case, the `ethercat_port` variable is as below:

```
setenv ethercat_port "fm1-mac3,fm1-mac4,3;fm1-mac5,,2;"
```

As mentioned above, each master is bundled to a specific CPU core. So it is recommended to bundle the corresponding EtherCAT real-time application to the same core with the master.

7.1.3.2.7 Starting IGH EtherCAT

Use the below command to start the IGH EtherCAT daemon:

```
$ ethercatctl start
```

Also, the below commands are used to stop or restart it.

```
# ethercatctl stop
# ethercatctl restart
```

**Note:** If the generic driver is used, make sure using "ifconfig <ethX> up" command to enable the network Card.

IGH provides a powerful auxiliary command-line tool, named “ethercat”. It can be used to query the master and all slaves information and status. The usage is as below:

```
Usage: ethercat <COMMAND> [OPTIONS] [ARGUMENTS]
Commands (can be abbreviated):
  alias Write alias addresses.
  config Show slave configurations.
  crc CRC error register diagnosis.
  cstruct Generate slave PDO information in C language.
  data Output binary domain process data.
  debug Set the master's debug level.
  domains Show configured domains.
  download Write an SDO entry to a slave.
  eoe Display Ethernet over EtherCAT statistics.
  foe_read Read a file from a slave via FoE.
  foe_write Store a file on a slave via FoE.
  graph Output the bus topology as a graph.
  master Show master and Ethernet device information.
  pdos List Sync managers, PDO assignment and mapping.
  reg_read Output a slave's register contents.
  reg_write Write data to a slave's registers.
  rescan Rescan the bus.
```
Real-time Edge also provides a systemd service to run IGH EtherCAT daemon as a system service.

```
# systemctl enable ethercat
# systemctl start ethercat
```

The below commands are used to stop or disable this service:

```
# systemctl stop ethercat
# systemctl disable ethercat
```

### 7.1.3.3 The 'real-time-edge-servo' stack

The **real-time-edge-servo** is a CiA402 (also referred to as DS402) profile framework based on IgH CoE interface. (For details of an EtherCAT Master stack, see Section 7.1). It abstracts the CiA 402 profile and provides an easily-usuable API for the Application developer.

The real-time-edge-servo project consists of a basic library `libnservo` and several auxiliary tools.

The application developed with `libnservo` is flexible enough to adapt to the changing of CoE network by modifying the `xml` config file, which is loaded when the application starts. The `xml` config file describes the necessary information, which includes EtherCAT network topology, slave and master configurations, and definitions of all the axles.

The stack has been tested on below CoE servo production: DELTA ASDA-B3, HCFA SV-X6EB, SV-X3EB, Just Motion Control 2HSS458-EC, and INOVANCE InoSV680N.

#### 7.1.3.3.1 CoE network

The Figure 94 illustrates a typical CoE network.
There are three CoE servos on this network and we name them slave x as the position they are. Each CoE servo can have more than one axle. The libnservo initiates the CoE network and encapsulates the details of network topology into axle nodes. Therefore, the developer can focus on the each axle operation without taking care of the network topology.

7.1.3.3.2 Libservo architecture

*real-time-edge-servo* runs on top of the *Igh* EtherCAT stack. The *Igh* stack provides CoE communication mechanisms - Mailbox and Process Data. Using these mechanisms, real-time-edge-servo can access the CiA Object Dictionary located on CoE servo. The Figure 95 shows the Libservo architecture.
The control task initiates the master and all slaves on the CoE network and registers all PDOs to Igh stack. It then constructs a data structure to describe each axle. Finally, the control task creates a task to run the user task periodically.

### 7.1.3.3.3 Xml configuration for real-time-edge-servo

This section describes how the xml config file that describes a CoE network should be configured. The basic framework of the XML configuration is shown in the code below:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Config Version="1.2">
  <PeriodTime>#1000000</PeriodTime>
  <MaxSafeStack>#8192</MaxSafeStack>
  <master_status_update_freq>#1</master_status_update_freq>
  <slave_status_update_freq>#1</slave_status_update_freq>
  <axle_status_update_freq>#1</axle_status_update_freq>
  <sync_ref_update_freq>#2</sync_ref_update_freq>
  <sched_priority>#90</sched_priority>
  <sched_policy>#SCHED_FIFO</sched_policy>
  <Masters>
    <Master>
      ...
    </Master>
    <Master>
      ...
    </Master>
    <Master>
```
• All config elements must be within the <Config> element.
• All config elements shown above are mandatory.
• The numerical value started with # indicates that it is a decimal value.
• The numerical value started with #x indicates that it is a hexadecimal value.
• <PeriodTime> element indicates that the period of control task is 10 ms.
• <MaxSafeStack> indicates the stack size, and it is an estimated value. 8K is sufficient to meet the needs of most applications.
• <master_status_update_freq> element indicates the frequency of masters status update. the value #1 means update the masters status every task period.
• <slave_status_update_freq> element indicates the frequency of slaves status update. the value #1 signifies to update the slaves status every task period.
• <axle_status_update_freq> element indicates the frequency of axles status update. the value #1 signifies to update the axles status every task periods.
• <sync_ref_update_freq> element indicates the frequency of reference clock update. the value #2 signifies to update the axles status every two task periods.
• <sched_policy> element specifies which schedule policy is used for a user task.
• <sched_priority> element indicates the priority of the user task.
• <Masters> element can contain more than one Master element. For most cases, there is only one master on a host.
• <Axles> element can contain more than one Axle element, which is an important feature for the developers.

### 7.1.3.3.1 Master element

As shown in the Section 7.1.3.3.1, the Master can have many slaves, so the Master element might consist of few Slave elements.

```xml
<Master>
  <Master_index>#0</Master_index>
  <Reference_clock>#0</Reference_clock>
  <Slave alias="#0" slave_position="#0">
    ....
  </Slave>
  <Slave alias="#1" slave_position="#1">
    ....
  </Slave>
</Master>
```

• <Master_index> element indicates the index of the master. As mentioned above, for many cases, there is only one master, so the value of this element is always #0.
• <Reference_clock> element indicates the slave that is used the reference clock.
• <Slave> element indicates that there is a slave on this master.
Slave element

```
<Slave alias="#0" slave_position="#0">
  <VendorId>#x66668888</VendorId>
  <ProductCode>#x20181302</ProductCode>
  <Name>2HSS458-EC</Name>
  <Emerg_size>#x08</Emerg_size>
  <WatchDog>
    <Divider>#x0</Divider>
    <Intervals>#4000</Intervals>
  </WatchDog>
  <DC>
    <SYNC SubIndex='#0'>
      <Shift>#0</Shift>
    </SYNC>
  </DC>
  <SyncManagers force_pdo_assign="#1">
    <SyncManager SubIndex="#0">
      ...
    </SyncManager>
    <SyncManager SubIndex="#1">
      ...
    </SyncManager>
  </SyncManagers>
  <Sdos>
    <Sdo>
      ...
    </Sdo>
    <Sdo>
      ...
    </Sdo>
  </Sdos>
</Slave>
```

- **alias** attribute means the alias name of this slave.
- **slave_position** attribute means which position of the slave is on this network.
- **Name** element is the name of the slave.
- **Emerg_size** element is always 8 for all CoE device.
- **WatchDog** element is used to set the watch dog of this slave.
- **DC** element is used to set the sync info.
- **SyncManagers** element should contain all syncManager channels.
- **Sdos** element contains the default value we want to initiate by SDO channel.

### SyncManagers Element

For a CoE device, there are generally four syncManager channels.

- SM0: Mailbox output
- SM1: Mailbox input
- SM2: Process data outputs
- SM3: Process data inputs

```
<SyncManager SubIndex="#2">
  <Index>#x1c12</Index>
  <Name>Sync Manager 2</Name>
  <Dir>OUTPUT</Dir>
```

All information provided in this document is subject to legal disclaimers. © 2023 NXP B.V. All rights reserved.
• `<Index>` element is the object address.
• `<Name>` is a name of this syncmanager channel.
• `<Dir>` element is the direction of this syncmanager channel.
• `<Watchdog>` is used to set watchdog of this syncmanager channel.
• `<PdoNum>` element means how many PDO we want to set.
• `<Pdo SubIndex` element contains the object dictionary entry that is required to be mapped to.
  – `<Index>` PDO address.
  – `<Name>` PDO name
  – `<Entry>` the object dictionary we want to mapped.

The Entry element is used to describe an object dictionary that is required to be mapped to.

```xml
<Entry SubIndex="#1">
  <Index>#x6041</Index>
  <SubIndex>#x0</SubIndex>
  <DataType>UINT</DataType>
  <BitLen>#16</BitLen>
  <Name>statusword</Name>
</Entry>
```

**Sdo element**

The Sdo element is used to set the default value of a object dictionary.

```xml
<Sdo>
  <Index>#x6085</Index>
  <Subindex>#x0</Subindex>
  <value>#x1000</value>
  <BitLen>#32</BitLen>
  <DataType>DINT</DataType>
  <Name>Quick_stop_deceleration</Name>
</Sdo>
```

The element shown in figure above means set the Object Dictionary "6085" to 0x1000.

### 7.1.3.3.3.2 Axle element

```xml
<Axle master_index='#0' slave_position='#0' AxleIndex='#0' AxleOffset='#0'>
  <Mode>pp</Mode>
  <Name>x-axle</Name>
  <reg_pdo>
```

---

All information provided in this document is subject to legal disclaimers.

© 2023 NXP B.V. All rights reserved.

User guide Rev. 2.7 — 18 December 2023
• master_index attribute indicates which master this axle belongs to.
• slave_position attribute indicates which slave this axle belongs to.
• AxleOffset attribute indicates which axle this axle is on the slave. As mentioned above, a CoE slave could have more than one axle. If this axle is the second axle on the slave, set AxleOffset="#1".
• <Mode> indicates the mode this axle will work on.
• <Name> is the name of this axle.
• <reg_pdo> is the PDO entry we want to register.

**reg_pdo** element

```xml
<reg_pdo>
  <Index>#x606c</Index>
  <Subindex>#x0</Subindex>
  <Name></Name>
</reg_pdo>
```

### 7.1.3.3.4 Testing a CoE servo system

#### 7.1.3.3.4.1 Hardware preparation

- A CoE servo system:
  A CoE servo system includes a CoE servo and a motor. In this test, 'Delta ASDA-B3-E' or '2HSS458-EC' servo system shown as in the figure below is used.
- A board supported by Real-time Edge:
  For this test, i.MX 8M Mini LPDDR4 EVK is used.
7.1.3.3.4.2 Software preparation

Make sure the below config options are selected when configuring Real-time Edge.

- igh-ethercat
- libxml2
- real-time-edge-servo

7.1.3.3.4.3 CoE network detection

- Igh configuration
  - Configure the MASTER0_DEVICE field of the /etc/ethercat.conf
  - Set MASTER0DEVICE to the MAC address to indicate which port the Igh uses.
  - Configure DEVICE_MODULES="generic" of the /etc/ethercat.conf
• Use the command below to start Igh service.

[root]# ethercatctl start

• Check CoE servo using below command.

[root]# ethercat slaves
0 0:0 PREOP + 2HSS458-EC
Or
[root]# ethercat slaves
0 0:0 PREOP + Delta ASDA-B3-E EtherCAT(CoE) Drive Rev0.00

7.1.3.3.4 Starting the test with 2HSS458-EC servo

Note:
The Position encoder resolution and Velocity encoder resolution of "2HSS458-EC" servo system are both 4000. It means the ratio of encoder increments per motor revolution.

Profile Position mode test

• Start the test service as below.

[root]# nservo_run -f /home/root/nservo_example/hss248_ec_config_pp.xml &

• Check whether the status of the slave has been transferred from "PREOP" to "OP".

[root]# ethercat slaves
0 0:0 OP + 2HSS458-EC

• Check whether the phase of the master has been transferred from "Idle" to "Operation".

[root]# ethercat master | grep Phase
Phase: Operation

• Run below commands to test whether the motor works.

  – Get current mode of axle 0.

    [root]# nservo_client -a 0 -c get_mode
    get_mode of the axle 0 : Profile Position Mode

  – Get current position of axle 0.

    [root]# nservo_client -a 0 -c get_current_position
    get_current_position of the axle 0 : 0

  – Get the profile speed of axle 0.

    [root]# nservo_client -a 0 -c get_profile_velocity
    get_profile_velocity of the axle 0 : 800000

    The value 800000 indicates 200 revolutions per second.

  – Set profile speed of axle 0.

    [root]# nservo_client -a 0 -c set_profile_velocity:20000
    set_profile_velocity of the axle 0 : 20000

    Set the profile speed to 5 revolutions per second.

  – Set target position of axle 0

    [root]# nservo_client -c set_target_position:400000
    set_target_position of the axle 0 : 400000
The value 400000 means that the motor will turn 100 rounds.
(target_position: 400000 - current_position:0) / 4000 = 100

- Get current speed of axle 0
  
  `[root]# nservo_client -a 0 -c get_current_velocity
  current_velocity of the axle 0 : 19999`

- Get target position of axle 0
  
  `[root]# nservo_client -a 0 -c get_target_position
  get_target_position of the axle 0 : 400000`

- Exit
  
  `[root]# nservo_client -c exit`

### Profile Velocity mode test

- Start the test service as below.
  
  `[root]# nservo_run -f /home/root/nservo_example/hss248_ec_config_pv.xml &`

- Check whether the status of the slave has been transferred from "PREOP" to "OP".
  
  `[root]# ethercat slaves
  0 0:0 OP 2HSS458-EC`

- Check whether the phase of the master has been transferred from "Idle" to "Operation".
  
  `[root]# ethercat master | grep Phase
  Phase: Operation`

- Run below commands to test whether the motor works.
  
  - Get current mode of axle 0.
    
    `[root]# nservo_client -a 0 -c get_mode
    get_mode of the axle 0 : Profile Velocity Mode`

  - Set target speed of axle 0.
    
    `[root]# nservo_client -a 0 -c set_target_velocity:40000
    set_target_velocity of the axle 0 : 40000`

  The value 40000 means that the motor will turn with 10 revolutions per second.

- Get current speed of axle 0.
  
  `[root]# nservo_client -a 0 -c get_current_velocity
  get_current_velocity of the axle 0 : 32000`

- Get target speed of axle 0.
  
  `[root]# nservo_client -a 0 -c get_target_velocity
  get_target_velocity of the axle 0 : 40000`

- Exit
  
  `[root]# nservo_client -c exit`
7.1.3.3.4.5 Starting test with ASDA-B3-E servo system

**Note:** The position encoder resolution of "ASDA-B3-E" servo system is 16777216 (24 bits). It signifies that the ratio of encoder increments per motor revolution.

**Profile Position mode test**

1. Start the test service as below.
   
   ```
   [root]# nservo_run -f /home/root/nservo_example/Delta-ASDA-B3-pp.xml &
   ```

   Check whether the status of the slave has been transferred from "PREOP" to "OP".

   ```
   [root]# ethercat slaves
   0 0:0 OP + Delta ASDA-B3-E EtherCAT(CoE) Drive Rev0.00
   ```

2. Check whether the phase of the master has been transferred from "Idle" to "Operation".

   ```
   [root]# ethercat master | grep Phase
   Phase: Operation
   ```

3. Run the below commands to test whether the motor works.
   
   a. Get the current mode of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c get_mode
      get_mode of the axle 0 : Profile Position Mode
      ```
   
   b. Get the current position of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c get_current_position
      get_current_position of the axle 0 : 0
      ```
   
   c. Get the profile speed of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c get_profile_velocity
      get_profile_velocity of the axle 0 : 0
      ```
   
   d. Set profile speed of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c set_profile_velocity:16777216
      set_profile_velocity of the axle 0 : 16777216
      ```
   
   e. Set profile speed to 1 revolutions per second. Set target position of axle 0.
      
      ```
      [root]# nservo_client -c set_target_position:167772160
      set_target_position of the axle 0 : 167772160
      ```
      
      The value 167772160 means that the motor turns 10 rounds. (target_position: 167772160 - current_position:0) / 16777216 = 10
   
   f. Get the current position of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c get_current_position
      get_current_position of the axle 0 : 167772152
      ```
   
   g. Get target position of axle 0
      
      ```
      [root]# nservo_client -a 0 -c get_target_position
      get_target_position of the axle 0 : 167772160
      ```

4. Exit.

   ```
   [root]# nservo_client -c exit
   ```

**Profile Velocity mode test**
1. Start the test service as below.
   
   ```
   [root]# nservo_run -f /home/root/nservo_example/Delta-ASDA-B3-pv.xml &
   ```

2. Check whether the status of the slave has been transferred from "PREOP" to "OP".
   
   ```
   [root]# ethercat slaves
   0 0:0 OP + Delta ASDA-B3-E EtherCAT(CoE) Drive Rev0.00
   ```

3. Check whether the phase of the master has been transferred from "Idle" to "Operation".
   
   ```
   [root]# ethercat master | grep Phase Phase: Operation
   ```

4. Run the below commands to test whether the motor works.
   a. Get current mode of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c get_mode
      get_mode of the axle 0 : Profile Velocity Mode
      ```
   b. Set target speed of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c set_target_velocity:600
      set_target_velocity of the axle 0 : 600
      ```
      The value 600 means that the motor turns with 60 revolutions per minute for ASDA-B3-E servo.
   c. Get current speed of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c get_current_velocity
      get_current_velocity of the axle 0 : 600
      ```
   d. Get target speed of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c get_target_velocity
      get_target_velocity of the axle 0 : 600
      ```

5. Exit
   
   ```
   [root]# nservo_client -c exit
   ```

**Cyclic Sync Position mode test**

1. Start the test service as below.
   
   ```
   [root]# nservo_run -f /home/root/nservo_example/Delta-ASDA-B3-csp.xml &
   ```

2. Check whether the status of the slave has been transferred from "PREOP" to "OP".
   
   ```
   [root]# ethercat slaves
   0 0:0 OP + Delta ASDA-B3-E EtherCAT(CoE) Drive Rev0.00
   ```

3. Check whether the phase of the master has been transferred from "Idle" to "Operation".
   
   ```
   [root]# ethercat master | grep Phase Phase: Operation
   ```

4. Run the below commands to test whether the motor works.
   a. Get current mode of axle 0.
      
      ```
      [root]# nservo_client -a 0 -c get_mode
      ```
b. Load trajectory planning information from the command line for axle 0.

```
./nservo_client -a 0 -c set_tparrays:"Cyclic=1; Scale=46603; Bias=0;
Accel=8; Decel=8; Max_speed=3600; TpArrays=[(0:1000),(45:1000),(45:1000),
(90:1000)];"
```

The command parameters are described below:

- **Cyclic**: if this field is set, the motor will come back to the first point of TpArrays and begin running again.
- **Scale**: this field is used to set the resolution of per unit. For example, the unit of position point in TpArrays is degree. So the Scale should be set to $167772160/360 = 46603$. 167772160 is the resolution per motor revolution.
- **Bias**: this field is used to set the bias value for the position point in TpArrays.
- **Accel**: the acceleration, unit$^2$ per second.
- **Decel**: the deceleration, unit$^2$ per second.
- **Max_speed**: The maximum speed, unit per second.
- **TpArrays**: is used to save trajectory planning information. Each element represents a position point and the time taken for the motor to rotate to this point from the last point. The unit of the time field is ms. Load trajectory planning information from a tp file for axle 0. Except loading the trajectory planning information from command line. This information could also be loaded from a tp file. The format of a sample tp file is shown below.

```
cat example/x6e_sv680_delta_tp_arrays
Axis=0; Cyclic=1; Scale=364; Bias=0; Accel=8; Decel=8; Max_speed=3600;
(270:2000),(0:1000)];
Axis=1; Cyclic=1; Scale=364; Bias=0; Accel=8; Decel=8; Max_speed=3600;
(270:2000),(0:1000)];
```

- **Axis**: the index of the axle.
- **Cyclic**: if this field is set, the motor will come back to the first point of TpArrays and run again.
- **Scale**: this field is used to set the resolution of per unit. For example, the unit of position point in TpArrays is degree. So the Scale should be set to $167772160/360 = 46603$. 167772160 is the resolution per motor revolution.
- **Bias**: this field is used to set bias value for the position point in TpArrays.
- **Accel**: the acceleration in unit$^2$ per second.
- **Decel**: the deceleration in unit$^2$ per second.
- **Max_speed**: The maximum speed, unit per second.
- **TpArrays**: are used to save trajectory planning information. Each element represents a position point and the time taken for the motor rotating to the this point from the last point. The unit of the time field is ms.

```
./nservo_client -c load_tp_file:"/home/root/nservo_example/Delta-ASDA-B3-tp_arrays"
```

- **Cyclic**: if this field is set, the motor will come back to the first point of TpArrays and begin running again.
- **Scale**: this field is used to set the resolution of per unit. For example, the unit of the position point in TpArrays is degree. So the Scale should be set to $167772160/360 = 46603$. 167772160 is the resolution per motor revolution.
- **Accel**: the acceleration, unit$^2$ per second.
• **Decel**: the deceleration, unit^2 per second.
• **Max_speed**: The maximum speed, unit per second.
• **TpArrays**: used to save trajectory planning information. Each element represents a position point and the time taken for the motor to rotate to this point from the last point. The unit of the time field is ms.

Set the axle 0 to start running using the below command:
```
[root]$ nservo_client -a 0 -c set_start
set_start of the axis 0
```

Set the current_position of axle 0 using the below command:
```
[root]$ nservo_client -a 0 -c get_current_position
get_current_position of the axis 0 : 2815922
```

Set axle 0 to stop running using the below command:
```
[root]$ nservo_client -a 0 -c set_stop
set_stop of the axis 0
```

Set all axles in CSP mode to start running using the below command:
```
[root]$ nservo_client -c set_start_all
```
All CSP axis in ready status start to run.

Set all axles in CSP mode to stop running using the below command:
```
[root]$ nservo_client -c set_stop_all
```
All CSP axis in running status begin to stop.

• Exit
```
[root]$ nservo_client -c exit
```

### 7.1.3.3.5 EtherCAT multiple axes control system

#### 7.1.3.3.5.1 HCFA 60-axes servo system

Below is a 60-axes servo system built by HCFA, and this system consists of 60 X3E servo motors and also 60 pointers in the screen which could rotate 360 degrees under the corresponding servo motor control. As shown in the figure below, this system can render any character on the screen by rotating these pointers.

Any platforms supported on Real-time Edge could be used as the controller of this servo system, and the software is based on real-time-edge-servo stack.
Figure 98. 60-axes servo system built by HCFA

HCFA performance

The task period is 1 ms, and servo control mode is CSP.
- Native EtherCAT driver + IGH stack: 26 µs
- Schedule latency: 200 µs on i.MX 8MP, 220 µs on i.MX 8M Mini
- Link propagation latency: 64 µs
- Customer task: 690-700 µs saved for app

Running this case (HCFA)

All software associated with this controller is integrated in Real-time Edge by default, and can be set up using the below steps:
- Start the nservo service as below:
  
  [root]# nservo_run -f /home/root/nservo_example/x3e_csp_60_config.xml &

- Run the below commands to confirm whether the motor works:

  [root]# nservo_client -a 59 -c get_mode
Load the trajectory planning information:

```
[root]# nservo_client -c load_tp_file="/home/root/nservo_example/x3e_60_axis_nxp_logo" file x3e_60_axis_nxp_logo includes all of trajectory planning information for 60-axes to control this servo system to present NXP logo.
```

Start all servo motors:

After the trajectory planning information file is loaded, use the below command to start the system.

```
[root]# nservo_client -c set_start_all
```

### 7.1.4 SOEM EtherCAT Master

The SOEM is a library that provides the user application the means to send and receive EtherCAT frames. The application provides for:

- Reading and writing process data to be sent/received by SOEM
- Keeping local IO data synchronized with the global IO map
- Detecting errors reported by SOEM
- Managing errors reported by SOEM

Refer [http://openetherecatsociety.github.io/doc/soem/tutorial_8txt.html](http://openetherecatsociety.github.io/doc/soem/tutorial_8txt.html) for more details. On Real-time Edge software, SOEM is only supported on Cortex-M core on the below platform:

- i.MX 8M Plus LPDDR4 EVK platform
- i.MX 8M Mini LPDDR4 EVK platform

And SOEM runs on FreeRTOS or without an operating system (baremetal).

There are two SOEM demos provided in this release. Both of them have the same function, but is based on different layer. Demo "freertos_soem_gpio_pulse" is based on FreeRTOS, while demo "soem-gpio-pulse" is based on bare metal (without an operating system).

### 7.1.4.1 SOEM for i.MX 8M Plus LPDDR4 EVK platform

#### 7.1.4.1.1 Setup hardware environment

The below hardware is required to set up these two demos.

- i.MX 8M Plus LPDDR4 EVK platform
- EK1100 - EtherCAT Coupler
- EL2008 - EtherCAT Terminal, 8-channel digital output, 24V DC
- EL1018 - EtherCAT Terminal, 8-channel digital input, 24 V DC
- 24V DC Power Supply
The channel-1 of the EL1018 is connected to a button to generate a 24 V pulse as an input signal to change the output of the channel-1 of the EL2008 labeled as "Dir". The channel-2 of the EL2008 outputs a square-wave signal with a period of 250 µs.

### 7.1.4.1.2 Building the demo images

By default, the demo images "freertos_soem_gpio_pules" and "soem_gpio_pules" are compiled with the i.MX 8M Plus LPDDR4 EVK target image compiling, and they are installed into the directory "example" of the target rootfs. For JTAG download, these images can also be found in the directory "<image-build-dir>/tmp/deploy/images/imx8mpevk/examples/" on building host.

```bash
cd <image-build-dir>/
tree tmp/deploy/images/imx8mpevk/examples/mcux-sdk/soem-gpio-pulse
|-- ddr_release
|   |-- soem_gpio_pulse.elf
|   |-- soem_gpio_pulse.bin
|   └── release
|       |-- soem_gpio_pulse.elf
|       |-- soem_gpio_pulse.bin

tree tmp/deploy/images/imx8mpevk/examples/mcux-sdk/freertos-soem-gpio-pulse
|-- ddr_release
|   |-- freertos_soem_gpio_pulse.elf
|   |-- freertos_soem_gpio_pulse.bin
|   └── release
|       |-- freertos_soem_gpio_pulse.elf
|       |-- freertos_soem_gpio_pulse.bin
```

Also, the demo images can be compiled using the below command on i.MX 8M Plus LPDDR4 EVK build directory:

```
bitbake soem-gpio-pulse
```
or

```
bitbake freertos-soem-gpio-pulse
```

### 7.1.4.1.3 Running SOEM demo images using J-Link GDB Server

This section describes the steps to run a demo application using J-Link GDB Server application. After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the i.MX 8M Plus LPDDR4 EVK platform to your PC via USB cable between the USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.

2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number. Configure the terminal with these settings:
   - 115200 baud rate
   - No parity
   - 8 data bits
   - 1 stop bit

3. Open the J-Link GDB Server application on Linux host.

   If the OS of your PC is Linux, using the below configuration to set up GDB Server. Assuming the J-Link software is installed, the application can be launched from a new terminal for the MIMX8ML_M7 device:

   ```
   $ JLinkGDBServer -if JTAG -device
   SEGGER J-Link GDB Server Command Line Version
   JLinkARM.dll
   Command line: -if JTAG -device MIMX8ML8_M7
   -----GDB Server start settings-----
   GDBInit file: none
   GDB Server Listening port: 2331
   SWO raw output listening port: 2332
   Terminal I/O port: 2333
   Accept remote connection: yes
   < -- Skipping lines -->
   Target connection timeout: 0 ms
   ------J-Link related settings------
   J-Link Host interface: USB
   J-Link script: none
   J-Link settings file: none
   ------Target related settings------
   Target device:
   Target interface: JTAG
   Target interface speed: 1000 kHz
   Target endian: little
   Connecting to J-Link...
   J-Link is connected.
   Firmware: J-Link V10 compiled Feb 2 2020 18:12:40
   Hardware: V10.10
   S/N: 600109545 Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
   Checking target voltage...
   Target voltage: 1.82 V
   Listening on TCP/IP port 2331
   Connecting to target...
   J-Link found 1 JTAG device, Total IRLen = 4
   JTAG ID: 0x5BA00477 (Cortex-M7)
   Connected to target
   Waiting for GDB connection...
   ```
4. Open the J-Link GDB Server application on Window host.
   If the OS of your PC is Windows, use the below image to set up GDB Server. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system “Start” menu and selecting “Programs -> SEGGER -> J-Link <version> J-Link GDB Server”.
   After server is launched, modify the settings as below. The target device selected for this demo is MIMX8ML8_M7.

After GDB server is running, the screen looks like as below:

After GDB server is running, the screen looks like as below:
5. Start the GDB client:
Assuming the arm-none-eabi-gdb is installed, and change to the directory that contains the demo images output "<build-dir>/tmp/deploy/images/imx8mpevk/examples/soem-gpio-pulse".

```
$ arm-none-eabi-gdb ./release/soem_gpio_pulse.elf
```

GNU gdb (GNU Tools for Arm Embedded Processors 9-2019-q4-major) 8.3.0.20190709-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later < http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>
For help, type "help". Type "apropos word" to search for commands related to "word"...
Reading symbols from soem_gpio_pulse.elf...
(gdb)

6. Connect to the GDB server and load the binary by running the following commands:
   a. target remote <GDB Server IP>:2331
   b. monitor reset
   c. monitor halt
   d. load

```
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
```
0x00000008 in __isr_vector ()
(gdb) monitor reset Resetting target
(gdb) monitor halt
(gdb) load
Loading section .interrupts, size 0x240 lma 0x0
Loading section .text, size 0x3ab8 lma 0x240
Loading section .ARM, size 0x8 lma 0x3cf8
Loading section .init_array, size 0x4 lma 0x3d00
Loading section .fini_array, size 0x4 lma 0x3d04
Loading section .data, size 0x64 lma 0x3d08
Start address 0x2f4, load size 15724
Transfer rate: 264 KB/sec, 2620 bytes/write.
(gdb)

The application is now downloaded and stopped at the reset vector. Execute the monitor go command to start the demo application.

(gdb) monitor go

7.1.4.1.4 Running SOEM demo images by U-Boot

This section describes the steps to write SOEM demo image file to TCM or DRAM with the Real-time Edge image. The following steps describe how to use the U-Boot:

1. Connect the DEBUG UART slot on the board to your PC through the USB cable. The Windows OS installs the USB driver automatically, and the Ubuntu OS finds the serial devices as well.

2. On Windows OS, open the device manager, find USB serial Port in Ports (COM and LPT). Assume that the ports are COM9 and COM10. One port is for the debug message from the Cortex-A53 and the other is for the Cortex-M7. The port number is allocated randomly, so opening both is beneficial for development. On Ubuntu OS, find the TTY device with name /dev/ttyUSB* to determine your debug port. Similar to Windows OS, opening both is beneficial for development.

3. Build and flash the nxp-image-real-time-edge image to a SD card and insert the SD card to the target board. Make sure to use the default boot SD slot and check the DIP boot switch configuration.

4. Open your preferred serial terminal for the serial devices. Set the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity. Then, power on the board.

5. Power on the board and hit any key to stop autoboot in the terminal window. Then enter the U-Boot command line mode. You can then write the image and run it from TCM or DRAM with the following commands:

   a. If the soem_gpio_pulse.bin is made from the <examples/..>/release target, which means the binary file will run at TCM, use the following commands to boot:

   ```
   => ext4load mmc 1:2 0x48000000 /examples/mcux-sdk/soem-gpio-pulse/release/
   soem_gpio_pulse.bin;
   => cp.b 0x48000000 0x7e0000 0x20000;
   => bootaux 0x7e0000
   ```

   b. If the soem_gpio_pulse.bin is made from the <examples/..>/ddr_release target, which means the binary file runs at DRAM, use the following commands:

   ```
   => ext4load mmc 1:2 0x80000000 /examples/mcux-sdk/soem-gpio-pulse/
   ddr_release/soem_gpio_pulse.bin;
   => dcache flush
   => bootaux 0x80000000
   ```
**Note:** If the Linux OS kernel runs together with M7, make sure the correct dtb file is used. This dtb file reserves resources used by M7 and avoids the Linux kernel from configuring them. Use the following command in U-Boot before running the kernel:

```
setenv fdtfile imx8mp-evk-rpmsg.dtb
```

**Note:**

You should perform a modification in case you require that SOEM app still works when running Linux. M7 core needs exclusive access to Ethernet. So, you should remove the Ethernet access from Linux Kernel. Perform the following steps:

1. In `kernel-source/arch/arm64/boot/dts/freescale/imx8mp-evk-rpmsg.dts` add at the end of the kernel device tree the following lines:

   ```
   &fec {
     status = "disabled";
   };
   
   Recompile the kernel:
   $ bitbake -f -c compile virtual/kernel
   $ bitbake virtual/kernel
   ```

2. Copy the new device tree and the kernel image to SD boot partition.

Now, SOEM still works while Linux is running.

### 7.1.4.2 SOEM for i.MX 8M Mini LPDDR4 EVK platform

#### 7.1.4.2.1 Setup hardware environment

The below hardware are required to set up these two demos.

- i.MX 8M Mini LPDDR4 EVK platform
- EK1100 - EtherCAT Coupler
- EL2008 - EtherCAT Terminal, 8-channel digital output, 24V DC
- EL1018 - EtherCAT Terminal, 8-channel digital input, 24 V DC
- 24V DC Power Supply

![Figure 101. Hardware setup for SOEM on i.MX 8M Mini LPDDR4 EVK platform](image)
The channel-1 of the EL1018 is connected to a button to generate a 24 V pulse as an input signal to change the output of the channel-1 of the EL2008 labeled as "Dir". The channel-2 of the EL2008 outputs a square-wave signal with a period of 250 µs.

### 7.1.4.2.2 Building the demo images

The demo images "freertos_soem_gpio_pulses" and "soem_gpio_pulses" are by default compiled with the i.MX 8M Mini LPDDR4 EVK target image compiling. They are installed into the directory "/example" of the target rootfs. For JTAG download, these images can also be found on the directory "<image-build-dir>/tmp/deploy/images/imx8mmevk/examples/" on building host.

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mmevk/examples/mcxux-sdk/soem-gpio-pulse
|-- ddr_release
 |  |-- soem_gpio_pulse.elf
 |  |-- soem_gpio_pulse.bin
|-- release
 |  |-- soem_gpio_pulse.elf
 |  |-- soem_gpio_pulse.bin
tree tmp/deploy/images/imx8mmevk/examples/mcxux-sdk/freertos-soem-gpio-pulse
|-- ddr_release
 |  |-- freertos_soem_gpio_pulse.elf
 |  |-- freertos_soem_gpio_pulse.bin
|-- release
 |  |-- freertos_soem_gpio_pulse.elf
 |  |-- freertos_soem_gpio_pulse.bin
```

Also, the demo images could be compiled using the below command on i.MX 8M Mini LPDDR4 EVK build directory:

```
bitbake soem-gpio-pulse
```

or

```
bitbake freertos-soem-gpio-pulse
```

### 7.1.4.2.3 Running SOEM demo images

This section describes the steps to run a demo application using J-Link GDB Server application. After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the i.MX 8M Mini LPDDR4 EVK platform to your PC via USB cable between the USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number. Configure the terminal with these settings:
   - 115200 baud rate
   - No parity
   - 8 data bits
   - 1 stop bit
3. Open the J-Link GDB Server application on Linux host.
   If the OS of your PC is Linux, using the below configuration to set up GDB Server. Assuming the J-Link software is installed, the application can be launched from a new terminal for the MIMX8MM6_M4 device:
   ```
   $ JLinkGDBServer -if JTAG -device
   ```
4. Open the J-Link GDB Server application on Window host.

If the OS of your PC is Windows, use the settings listed in the below figure to set up GDB Server. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system “Start” menu and selecting “Programs -> SEGGER -> J-Link <version> J-Link GDB Server”.

After launch, modify the settings as below. The target device selected for this demo is MIMX8MM6_M4.
After GDB server is running, the screen looks like as below:

5. Start the GDB client:
   Assuming the arm-none-eabi-gdb is installed, and change to the directory that contains the demo images output "<build-dir>/tmp/deploy/images/imx8mmevk/examples/soem-gpio-pulse".

   $ arm-none-eabi-gdb ./release/soem_gpio_pulse.elf
   GNU gdb (GNU Tools for Arm Embedded Processors 9-2019-q4-major)
   8.3.0.20190709-git
6. Connect to the GDB server and load the binary by running the following commands:
   a. target remote <GDB Server IP>:2331
   b. monitor reset
   c. monitor halt
   d. load

   ```
   (gdb) target remote localhost:2331
   Remote debugging using localhost:2331
   0x00000008 in __isr_vector ()
   (gdb) monitor reset Resetting target
   (gdb) monitor halt
   (gdb) load
   Loading section .interrupts, size 0x240 lma 0x0
   Loading section .text, size 0x3ab8 lma 0x240
   Loading section .ARM, size 0x8 lma 0x3cf8
   Loading section .init_array, size 0x4 lma 0x3d00
   Loading section .fini_array, size 0x4 lma 0x3d04
   Loading section .data, size 0x64 lma 0x3d08
   Start address 0x2f4, load size 15724
   Transfer rate: 264 KB/sec, 2620 bytes/write.
   (gdb)
   ```

   The application is now downloaded and stopped at the reset vector. Execute the monitor go command to start the demo application.

   ```
   (gdb) monitor go
   ```

7.1.4.2.4 Running SOEM demo images by U-Boot

The steps to write SOEM demo image file to TCM or DRAM using the Real-time Edge image on i.MX 8M Mini LPDDR4 EVK are same as followed for i.MX 8M Plus LPDDR4 EVK board. Refer Section 7.1.4.1.4.

7.1.5 CODESYS EtherCAT Master

7.1.5.1 Overview

Based on Real-time Edge yocto project, a new yocto distro named "nxp-real-time-edge-plc" is added, which is specific to the PLC use case. This document describes how to build a Real-time Edge PLC image and how to setup a CODESYS test project as an example to drive motors.
7.1.5.2 Features

Features are as follows:

- **Optimized native driver**
  
  Typical industrial software, for example CODESYS or SOEM EtherCAT master stack, work on user space and communicate using Linux standard network interface. To reduce the latency when Ethernet raw packets pass from user space through Linux standard network, the network driver is optimized to avoid memory reallocation and copying, task rescheduling. The latency of the critical path is reduced for packet transmitting and receiving.

- **Light root filesystem**
  
  Lighter root filesystem saves CPU cycles that could be used by the PLC user program.

- **Supported platforms**
  
  - imx6ull14x14evk
  - imx8mm-lpddr4-evk
  - imx8mp-lpddr4-evk
  - imx93evk

7.1.5.3 Building the image

This section provides detailed information along with the procedure for building an image.

7.1.5.3.1 Build configurations

A new yocto distro named "nxp-real-time-edge-plc" is added for the PLC use case, and below platforms are supported:

- imx6ull14x14evk
- imx8mm-lpddr4-evk
- imx8mp-lpddr4-evk
- imx93evk

Real-time Edge provides the script real-time-edge-setup-env.sh to simply the setup for both i.MX and Layerscape boards. To use the script, the name of the specific machine to be built for and the desired distro must be specified. The script sets up a directory and the configuration files for the specified machine and distro.

The syntax for the `real-time-edge-setup-env.sh` script is shown below:

```
$ DISTRO=nxp-real-time-edge-plc MACHINE=<machine name> source real-time-edge-setup-env.sh -b <build dir>
```

where:

- **MACHINE=<machine name>** is the name of the supported platforms.
- **-b <build dir>** specifies the name of the build directory created by the `real-time-edge-setup-env.sh` script.

After the script runs, the working directory is the one just created by the script, specified with the `-b` option. A `conf` folder is created containing the files `bblayers.conf` and `local.conf`. 
The local.conf file contains the machine and distro specifications. An example is shown below:

<table>
<thead>
<tr>
<th>MACHINE</th>
<th>DISTRO</th>
<th>ACCEPT_FSL_EULA</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>??='imx8mp-lpddr4-evk'</code></td>
<td><code>?='nxp-real-time-edge-plc'</code></td>
<td><code>'1'</code></td>
</tr>
</tbody>
</table>

The MACHINE configuration can be changed by editing this file, if necessary.

### 7.1.5.3.2 Build scenarios

The following are build setup scenarios for various configurations. Set up the manifest and populate the Yocto Project layer sources using the commands below:

```bash
$ mkdir yocto-real-time-edge
$ cd yocto-real-time-edge
$ repo init -u https://github.com/nxp-real-time-edge-sw/yocto-real-time-edge.git -b real-time-edge-mickledore -m real-time-edge-2.7.0.xml
$ repo sync
```

The following sections give some specific examples.

- **Real-time Edge PLC image on i.MX 6ULL EVK**

  ```bash
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx6ull14x14evk source real-time-edge-setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

- **Real-time Edge PLC image on i.MX 8M Mini EVK**

  ```bash
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx8mm-lpddr4-evk source real-time-edge-setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

- **Real-time Edge PLC image on i.MX 8M Plus EVK**

  ```bash
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx8mp-lpddr4-evk source real-time-edge-setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

- **Real-time Edge PLC image on i.MX 93 EVK**

  ```bash
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx93evk source real-time-edge-setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

### 7.1.5.3.3 Image deployment

All filesystem images are deployed to the `<build directory>/tmp/deploy/images` directory. Each image building creates a U-Boot image, a kernel image, and an image type based on the `IMAGE_FSTYPES` variable defined in the machine configuration file. Most machine configurations provide an SD card image (.wic) and a rootfs image (.tar). The SD card image contains a partitioned image (with U-Boot, kernel, rootfs, and other such files) suitable for booting the corresponding hardware.

**Copy image to SD card:**


The SD card image file, \texttt{(<image\_name>.wic)} contains a partitioned image (with U-Boot, kernel, rootfs, and other files) suitable for booting the corresponding hardware. Run the following commands to copy the image to an SD card:

\begin{verbatim}
$ zstd -d <image\_name>.wic.zst
$ sudo dd if=<image\_name>.wic of=/dev/sd<disk> bs=1M conv=fsync
\end{verbatim}

### 7.1.5.4 Running CODESYS on the boards

### 7.1.5.4.1 Running CODESYS on i.MX 6ULL

Execute the following command to update the dtb file and boot into kernel.

\begin{verbatim}
uboot ==> setenv fdt_file imx6ull-14x14-evk-ecat.dtb
uboot ==> run bootcmd
\end{verbatim}

Set the CPU frequency governor to "performance" for CPU core to use the highest frequency.

\begin{verbatim}
# echo performance > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
\end{verbatim}

Enable the EtherCAT port.

\begin{verbatim}
# eth'ercat_port=`ls /sys/bus/platform/devices/2188000.ethernet/net` && echo
$eth'ercat_port
# ifconfig $eth'ercat_port up
\end{verbatim}

Assign IP address for the Ethernet port automatically if there is a DHCP server in the network.

\begin{verbatim}
# ethernet_port=`ls /sys/bus/platform/devices/20b4000.ethernet/net` && echo
$ethernet_port
# udhcpc -i $ethernet_port
\end{verbatim}

Else assign IP address manually.

\begin{verbatim}
# ethernet_port=`ls /sys/bus/platform/devices/20b4000.ethernet/net` && echo
$ethernet_port
# ifconfig $ethernet_port 192.168.1.xx
\end{verbatim}

Use \texttt{scp} command to copy \texttt{codesyscontrol.bin} into the board and start it:

\begin{verbatim}
# ./codesyscontrol.bin > CODESYS.log &
\end{verbatim}

**Note:**

- Please refer to the next section to obtain \texttt{codesyscontrol.bin}.
- CODESYS logs can be viewed in the \texttt{CODESYS.log} file.
7.1.5.4.2 Running CODESYS on i.MX 8M Mini and i.MX 8M Plus

Execute the following command to isolate CPU1 (by default, the “EtherCAT_Task” thread created by the Realtime of CODESYS is bound into CPU1).

```
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=\n${console} root=${mmcroot} isolcpus=1'
```

Execute the following command to update the dtb file and boot into kernel.

```
// on i.MX 8M Mini
uboot ==> setenv fdtfile imx8mm-evk-ecat.dtb
// on i.MX 8M Plus
uboot ==> setenv fdtfile imx8mp-evk-ecat.dtb
uboot ==> run bootcmd
```

Set the CPU frequency governor to “performance” for CPU1 core to use the highest frequency.

```
echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

Stop the “udhcpc” service which interface is connected to the EtherCAT servos.

```
# ethercat_net=`ls /sys/bus/platform/devices/30be0000.ethernet/net` && echo
  $ethercat_net
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i
  $ethercat_net”
# kill -9 PID
```

Enable the EtherCAT port.

```
# ethercat_port=`ls /sys/bus/platform/devices/30be0000.ethernet/net` && echo
  $ethercat_port
# ifconfig $ethercat_port up
```

Assign IP address for the Ethernet port automatically if there is a DHCP server in the network.

```
// assume the name of the USB network port is eth1
# udhcpc -i eth1
```

Else assign IP address manually.

```
// assume the name of the USB network port is eth1
# ifconfig eth1 192.168.1.xx
```

Use scp command to copy codesyscontrol.bin into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

**Note:**

- Please refer to the next section to obtain codesyscontrol.bin.
- CODESYS logs can be viewed in the CODESYS.log file.
7.1.5.4.3 Running CODESYS on i.MX 93

Execute the following command to isolate CPU1 (by default, the “EtherCAT_Task” thread created by the Realtime of CODESYS is bound into CPU1).

```shell
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=\n{console} root=${mmcroot} isolcpus=1'
```

Execute the following command to update the dtb file and boot into kernel.

```shell
uboot ==> setenv fdtfile imx93-11x11-evk-ecat.dtb
uboot ==> run bootcmd
```

Stop the “udhcpc” service which interface is connected to the EtherCAT servos.

```shell
# ethercat_net=`ls /sys/bus/platform/devices/42890000.ethernet/net` && echo
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i
# $ethercat_net”
# kill -9 PID
```

Enable the EtherCAT port.

```shell
# ethercat_port=`ls /sys/bus/platform/devices/42890000.ethernet/net` && echo
# ifconfig $ethercat_port up
```

Assign IP address for the Ethernet port automatically if there is a DHCP server in the network.

```shell
# ethernet_port=`ls /sys/bus/platform/devices/428a0000.ethernet/net` && echo
# udhcpc -i $ethernet_port
```

Else assign IP address manually.

```shell
# ethernet_port=`ls /sys/bus/platform/devices/428a0000.ethernet/net` && echo
# ifconfig $ethernet_port 192.168.1.xx
```

Execute the following command to disable the "CPU1" idle state1.

```shell
# echo "1" > /sys/devices/system/cpu/cpu1/cpuidle/state1/disable
```

Execute the following command to disable the DDR auto clock gating.

```shell
# echo 0 > /sys/devices/platform/imx93-1pm/auto_clk_gating
```

Use `scp` command to copy `codesyscontrol.bin` into the board and start it:

```shell
# ./codesyscontrol.bin > CODESYS.log &
```

**Note:**
- Please refer to the next section to obtain `codesyscontrol.bin`.
- CODESYS logs can be viewed in the `CODESYS.log` file.
7.1.5.5 Setting up CODESYS

CODESYS is a powerful commercial PLC software programming tool. It supports IEC61131-3 standard IL, ST, FBD, LD, CFC, SFC, and six kinds of PLC programming languages. Users can choose different language editing subroutines in the same project, function module, and so on.

7.1.5.5.1 Board environment setup

i.MX 6ULL EVK Boards

• Hardware requirements
  – Two network cables
  – Mini/micro USB cable
  – One servo motor (DELTA ASDA-B3)
  – Personal Computer on which the CODESYS has been installed

• Preparing the example
  – Connect the PC and the network port labeled J1501 B on the board.
  – Connect the servo motors to the network port labeled J1501 A on the board.
  – Connect a USB cable between the host PC and the OpenSDA USB port on the target board. Then, open a serial terminal with the following settings
    – 115200 baud rate
    – 8 data bits
    – No parity
    – One stop bit
    – No flow control

i.MX 8M Mini EVK Boards

• Hardware requirements
  – Two Network cables
  – Mini/micro USB cable
  – Network port converter with type-C interface

  Note: i.MX 8M Mini has only one network port, so it is needs another network port connected with CODESYS IDE.
  – One servo motor (DELTA ASDA-B3)
  – Personal Computer on which the CODESYS has been installed.

• Preparing the example
  – Connect the servo and the network port of the board.
  – Plug the Network port converter into the USB port (labeled PORT1) of the board, and then connect the Network port converter to the PC using a network cable.
  – Connect a USB cable between the host PC and the OpenSDA USB port on the target board, open a serial terminal with the following settings:
    – 115200 baud rate
    – 8 data bits
    – No parity
    – One stop bit
    – No flow control

The connection diagram for i.MX 8M Mini board is as follows.
i.MX 8M Plus EVK Boards

• Hardware requirements
  – Two Network cables
  – Mini/micro USB cable
  – One servo motor (DELTA ASDAB3)
  – Personal Computer on which the CODESYS has been installed.

• Preparing the example
  – Connect the PC to the network port labeled **ENET2** on the board.
  – Connect the servo motors to the network port labeled **ENET1** on the board.
  – Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
    – 115200 baud rate
    – 8 data bits
    – No parity
    – One stop bit
    – No flow control

i.MX 93 EVK Boards

• Hardware requirements
  – Two Network cables
  – Mini/micro USB cable
  – One servo motor (DELTA ASDAB3)
  – Personal Computer on which the CODESYS has been installed
• **Preparing the example**
  – Connect the PC to the network port labeled **ENET1** on the board.
  – Connect the servo motors to the network port labeled **ENET2** on the board.
  – Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
    – 115200 baud rate
    – 8 data bits
    – No parity
    – One stop bit
    – No flow control

7.1.5.5.2 CODESYS project setup

7.1.5.5.2.1 Downloading CODESYS runtime binary

Download the corresponding runtime binary file from CODESYS website and decompress it using the commands below:

```
# unzip CODESYS\ Control\ for\ Linux\ ARM64\ SL\ 4.7.0.0.package CODESYS.package
# cd Delivery/
# dpkg-deb -R codesyscontrol_linuxarm64_4.7.0.0-b.trunk.39_arm64.deb software
# cd software/opt/codesys/bin/
```

The runtime binary **codesyscontrol.bin** is located in the **software/opt/codesys/bin/** directory.

**Note:**

• **Download 32-bit version for the platform:**
  - i.MX 6ULL EVK
• **Download 64-bit version for the platforms:**
  - i.MX 8M Mini EVK
  - i.MX 8M Plus EVK
  - i.MX 93 EVK

7.1.5.5.2.2 Starting CODESYS runtime

Use the **scp** command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

CODESYS logs can be viewed in the **CODESYS.log** file.

7.1.5.5.2.3 Creating a new CODESYS project

This section describes the steps to create the CODESYS project in the machine on which the CODESYS tool has been installed.

• Open **CODESYS** and build a new project using the steps below:
  1. Click **“File”** -> **“New project”**.
  2. Select **“Standard project”** as the template.
3. Rename this project.
4. Click “OK” to complete it.

**Set Device and PLC_PRG**

1. Select “CODESYS Control for Linux ARM64 SL” or “CODESYS Control for Linux ARM SL” as the Device.
2. Select “Structured Text(ST)” as the PLC_PRG.
3. Click “OK” to complete it.
7.1.5.5.2.4 Adding EtherCAT Master and Slave

- **Set EtherCAT master**

1. Right-click on the device -> “Add Device”
2. Select “EtherCAT Master SoftMotion”
3. Click “Add Device” to complete it.

- **Download DELTA ASDA-B3 device description file**
• Add DELTA device description file

1. Click “Tools” -> “Device Repository”.
2. Click “Install”.
3. Select “Delta ASDA-x3-E rev0.04_10EMC.XML” file.
4. Click “Close” to complete it.

Figure 106. Adding DELTA device description file
Figure 107. View Installed devices

• Add servo device
  1. Right-click on the "EtherCAT_Master_SoftMotion" -> “Add Device”.
  2. Select the corresponding servo.
  3. Click “Add Device” to complete it.

Figure 108. Adding a Device
Figure 109. Select the device to be added

- **Add CiA402 Axis device**

  1. Right-click on the servo -> “Add SoftMotion CiA402 Axis”.

Figure 110. Adding CiA402 Axis device
7.1.5.5.2.5 Adding PLC program

The below steps describe how to add the PLC program to the drive motor:

1. Right-click on the “Application” -> “Add Object” -> “POU” and name it “PLC_PRG”. (If it already exists, do not add it.) -> Pull the PLC_PRG under the EtherCAT_Task
2. Right-click on the “PLC_PRG” -> “Add Object” -> “action” and name it “axiscontrol”
3. Fill in the following PLC code:

```plaintext
PROGRAM PLC_PRG
VAR CONSTANT
CONST : INT := 1;  // The number of belt axes can be modified from here
END_VAR
VAR
Ton_E   :TON;
TON_F   :TON;
TON_B   :TON;
ECAT_Success  :BOOL;
ECAT_Error  :BOOL;
AXIS     : ARRAY [1..CONST] OF POINTER TO AXIS_REF_SM3;
FB_Power   : ARRAY [1..CONST] OF MC_Power;
FB_Jog    : ARRAY [1..CONST] OF MC_Jog;
N     : INT;
start     : BOOL;
JOG_Velocity   : REAL :=10;
JOG_Forward   :BOOL;
JOG_Backward  :BOOL;
END_VAR

// Master station operation judgment
Ton_E ( IN:=EtherCAT_Master_SoftMotion.xConfigFinished 
   AND EtherCAT_Master_SoftMotion.xDistributedClockInSync 
   AND NOT EtherCAT_Master_SoftMotion.xError , 
   PT:=T#50MS, Q=> , ET=> );
ECAT_Success S= Ton_E.Q;
```

The PLC_PRG (PRG) variable is as follows:

```
PROGRAM PLC_PRG
VAR CONSTANT
CONST : INT := 1;  // The number of belt axes can be modified from here
END_VAR
VAR
Ton_E   :TON;
TON_F   :TON;
TON_B   :TON;
ECAT_Success  :BOOL;
ECAT_Error  :BOOL;
AXIS     : ARRAY [1..CONST] OF POINTER TO AXIS_REF_SM3;
FB_Power   : ARRAY [1..CONST] OF MC_Power;
FB_Jog    : ARRAY [1..CONST] OF MC_Jog;
N     : INT;
start     : BOOL;
JOG_Velocity   : REAL :=10;
JOG_Forward   :BOOL;
JOG_Backward  :BOOL;
END_VAR

// Master station operation judgment
Ton_E ( IN:=EtherCAT_Master_SoftMotion.xConfigFinished 
   AND EtherCAT_Master_SoftMotion.xDistributedClockInSync 
   AND NOT EtherCAT_Master_SoftMotion.xError , 
   PT:=T#50MS, Q=> , ET=> );
ECAT_Success S= Ton_E.Q;
```
ECAT_Error := NOT Ton_E.Q AND ECAT_Success;
// Axis pointer acquisition
AXIS[1] := ADR(SM_Drive_GenericDSP402);
Axiscontrol();
IF ECAT_Success THEN
  start := TRUE;
ELSE
  start := FALSE;
END_IF
IF JOG_Forward=FALSE AND JOG_Backward =FALSE THEN
  JOG_Backward := TRUE;
END_IF
TON_F (IN:= JOG_Forward, PT:= T#10S, Q=> , ET=> );
TON_B (IN:= JOG_Backward, PT:= T#10S, Q=> , ET=> );
IF TON_F.Q THEN
  JOG_Forward   := FALSE;
  JOG_Backward  := TRUE;
ELSIF TON_B.Q THEN
  JOG_Forward   := TRUE;
  JOG_Backward  := FALSE;
END_IF

The axis control code is as follows:

FOR N :=1 TO CONST BY 1 DO
  FB_Power[N](
    Axis:= AXIS[N]^,
    Enable:= TRUE,
    bRegulatorOn:= start,
    bDriveStart:= TRUE,
    Status=> ,
    bRegulatorRealState=> ,
    bDriveStartRealState=> ,
    Busy=> ,
    Error=> ,
    ErrorID=> );
  FB_Jog[N](
    Axis:= AXIS[N]^,
    JogForward:= FB_Power[N].Status AND JOG_Forward,
    JogBackward:= FB_Power[N].Status AND JOG_Backward,
    Velocity:= JOG_Velocity,
    Acceleration:= JOG_Velocity *10,
    Deceleration:= JOG_Velocity *10,
    Jerk:= ,
    Busy=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorId=> );
END_FOR

7.1.5.5.2.6 Running CODESYS Project

1. In the Codesys Control window, fill in the IP address of the board in respective textbox. Then, press Enter to connect to the board as shown in the figure below:
2. Select the network port on the board that is used to communicate with the servo.

![Figure 112. Adding IP address of the board](image)

Note: For network port selection, refer to Section 7.1.5.5.1.

3. Click “Login” and “Start” button to run the PLC program:

![Figure 113. Selecting the network port](image)
7.2 FlexCAN and CANopen

The following sections provide an introduction to the FlexCAN standard, details of the CAN bus, the CANopen communication system, details of how to integrate FlexCAN with Real-time Edge, and running a FlexCAN application.

7.2.1 Introduction

The LS1028ARDB board have the FlexCAN module. The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0 B protocol specification. The main sub-blocks implemented in the FlexCAN module include an associated memory for storing message buffers, Receive (RX) Global Mask registers, Receive Individual Mask registers, Receive FIFO filters, and Receive FIFO ID filters. A general block diagram is shown in the following figure. The functions of these submodules are described in subsequent sections.
7.2.1.1 CAN bus

CAN (Controller Area Network) is a serial bus system. A CAN bus is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. Bosch published several versions of the CAN specification and the latest is CAN 2.0 published in 1991. This specification has two parts; part A is for the standard format with an 11-bit identifier, and part B is for the extended format with a 29-bit identifier. A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A and a CAN device that uses 29-bit identifiers is commonly called CAN 2.0B.

CAN is a multi-master serial bus standard for connecting Electronic Control Units [ECUs] also known as nodes. Two or more nodes are required on the CAN network to communicate. The complexity of the node can range from a simple I/O device up to an embedded computer with a CAN interface and sophisticated software. The node may also be a gateway allowing a standard computer to communicate over a USB or Ethernet port to the devices on a CAN network. All nodes are connected to each other through a two wire bus. The wires are a twisted pair with a 120 Ω (nominal) characteristic impedance.

High speed CAN signaling drives the CAN high wire towards 5 V and the CAN low wire towards 0 V when transmitting a dominant (0), and does not drive either wire when transmitting a recessive (1). The dominant differential voltage is a nominal 2 V. The termination resistor passively returns the two wires to a nominal differential voltage of 0 V. The dominant common mode voltage must be within 1.5 V to 3.5 V of common and the recessive common mode voltage must be within +/-12 V of common.
7.2.1.2 CANopen

CANopen is a CAN-based communication system. It comprises higher-layer protocols and profile specifications. CANopen has been developed as a standardized embedded network with highly flexible configuration capabilities. Today it is used in various application fields, such as medical equipment, off-road vehicles, maritime electronics, railway applications, and building automation.

CANopen provides several communication objects, which enable device designers to implement desired network behavior into a device. With these communication objects, device designers can offer devices that can communicate process data, indicate device-internal error conditions or influence and control the network.
behavior. As CANopen defines the internal device structure, the system designer knows exactly how to access a CANopen device and how to adjust the intended device behavior.

• CANopen lower layers
  CANopen is based on a data link layer according to ISO 11898-1. The CANopen bit timing is specified in CiA 301 and allows the adjustment of data rates from 10 kbit/s to 1000 kbit/s. Although all specified CAN-ID addressing schemata are based on the 11-bit CAN-ID, CANopen supports the 29-bit CAN-ID as well. Nevertheless, CANopen does not exclude other physical layer options.

• Internal device architecture
  A CANopen device consists of three logical parts. The CANopen protocol stack handles the communication via the CAN network. The application software provides the internal control functionality. The CANopen object dictionary interfaces the protocol as well as the application software. It contains indices for all used data types and stores all communication and application parameters. The CANopen object dictionary is most important for CANopen device configuration and diagnostics.

• CANopen protocols
  – SDO protocol
  – PDO protocol
  – NMT protocol
  – Special function protocols
  – Error control protocols

The following figure shows the CANopen architecture.
7.2.2 Introducing the function of CAN example code

CAN example code supports the CANopen protocol. It mainly implements three parts of functions: network manage function (NMT protocol), service data transmission function (SDO protocol), and process data transmission function (PDO protocol). NMT protocol can manage and monitor slave nodes, include heart beat message. SDO protocol can transmit single or block data. The PDO protocol can transmit process data that requires real time.

CAN example calls the CANopen interfaces, described in the table below:

Table 85. CAN Net APIs and their description

<table>
<thead>
<tr>
<th>API name (type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNS8 canReceive_driver (CAN_HANDLE fd0, Message * m)</td>
<td>SocketCAN receives CAN messages</td>
</tr>
<tr>
<td></td>
<td>• fd0 – SocketCAN handle</td>
</tr>
<tr>
<td></td>
<td>• m – Receive buffer</td>
</tr>
<tr>
<td>UNS8 canSend_driver (CAN_HANDLE fd0, Message const * m)</td>
<td>SocketCAN sends CAN messages</td>
</tr>
<tr>
<td></td>
<td>• fd0 – SocketCAN handle</td>
</tr>
<tr>
<td></td>
<td>• m – CAN message to be sent</td>
</tr>
</tbody>
</table>
Table 85. CAN Net APIs and their description...continued

<table>
<thead>
<tr>
<th>API name (type)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void setNodeId(CO_Data* d, UNS8 nodeId)</td>
<td>Set this node id value.</td>
</tr>
<tr>
<td></td>
<td>• d – object dictionary</td>
</tr>
<tr>
<td></td>
<td>• nodeId – id value (up to 127)</td>
</tr>
<tr>
<td>UNS8 setState(CO_Data* d, e_nodeState newState)</td>
<td>Set node state</td>
</tr>
<tr>
<td></td>
<td>• d – object dictionary</td>
</tr>
<tr>
<td></td>
<td>• newState – The state that needs to be set</td>
</tr>
<tr>
<td></td>
<td>Returns 0 if OK, &gt; 0 on error</td>
</tr>
<tr>
<td>void canDispatch(CO_Data* d, Message *m)</td>
<td>CANopen handles data frames that CAN receive.</td>
</tr>
<tr>
<td></td>
<td>• d – object dictionary</td>
</tr>
<tr>
<td></td>
<td>• m – Received CAN message</td>
</tr>
<tr>
<td>void timerForCan(void)</td>
<td>CANopen virtual clock counter.</td>
</tr>
<tr>
<td>UNS8 sendPDOrequest (CO_Data * d, UNS16 RPDOIndex)</td>
<td>Master node requests slave node to feedback specified data.</td>
</tr>
<tr>
<td></td>
<td>• d – object dictionary</td>
</tr>
<tr>
<td></td>
<td>• RPDOIndex – index value of specified data</td>
</tr>
<tr>
<td>UNS8 readNetworkDictCallback (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS8 dataType, SDOCallback_t Callback, UNS8 useBlockMode)</td>
<td>The master node gets the specified data from the slave node.</td>
</tr>
<tr>
<td></td>
<td>• d – object dictionary</td>
</tr>
<tr>
<td></td>
<td>• nodeId – the id value of slave node</td>
</tr>
<tr>
<td></td>
<td>• index – the index value of the specified data</td>
</tr>
<tr>
<td></td>
<td>• subIndex – the subindex value of the specified data</td>
</tr>
<tr>
<td></td>
<td>• dataType – the data type of the specified data</td>
</tr>
<tr>
<td></td>
<td>• Callback – callback function</td>
</tr>
<tr>
<td></td>
<td>• useBlockMode – specifies whether it is a block transmission</td>
</tr>
<tr>
<td>UNS8 writeNetworkDictCallBack (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS32 count, UNS8 dataType, void *data, SDOCallback_t Callback, UNS8 useBlockMode)</td>
<td>The master node sets the specified data to the slave node.</td>
</tr>
<tr>
<td></td>
<td>• d – object dictionary</td>
</tr>
<tr>
<td></td>
<td>• nodeId – the id value of slave node</td>
</tr>
<tr>
<td></td>
<td>• index – the index value of the specified data</td>
</tr>
<tr>
<td></td>
<td>• subIndex – the subindex value of the specified data</td>
</tr>
<tr>
<td></td>
<td>• count – the length of the specified data</td>
</tr>
<tr>
<td></td>
<td>• dataType – the data type of the specified data</td>
</tr>
<tr>
<td></td>
<td>• Callback – callback function</td>
</tr>
<tr>
<td></td>
<td>• useBlockMode – specifies whether it is a block transmission</td>
</tr>
</tbody>
</table>

7.2.3 Running a CAN application

The following sections describe the hardware and software preparation steps for running a CAN application.

7.2.3.1 Hardware preparation for LS1028ARDB

For LS1028ARDB, below hardware is required:

- LS1028ARDB board
- Two cables to connect CAN1 and CAN.

The hardware connection diagram is as shown in the following figure.
7.2.3.2 Running the SocketCAN commands

This section describes the steps for running SocketCAN commands that can be performed on LS1028ARDB. These commands are executed on Linux. The standard SocketCAN commands are the following:

1. Open the can0 port.
   $ ip link set can0 up

2. Close the can0 port.
   $ ip link set can0 down

3. Set the baud rate to 500K for the can0 port
   $ ip link set can0 type can bitrate 500000

4. Set can0 port to Loopback mode.
   $ ip link set can0 type can loopback on

5. Send a message through can0. 002 (HEX) is node id, and this value must be 3 characters. 2288DD (HEX) is a message, and can take a value up to 8 bytes.
   $ cansend can0 002#2288DD

6. Monitor can0 port and wait for receiving data.
   $ candump can0
7. See can0 port details.

```bash
$ ip -details link show can0
```

*Note: The third and fourth commands are valid when the state of can0 port is closed.*

### 7.2.3.3 Testing CAN bus

Below is the sample code for testing the CAN bus on LS1028ARDB.

```bash
[root]# ip link set can0 down
[root]# ip link set can1 down
[root]# ip link set can0 type can loopback off
[root]# ip link set can1 type can loopback off
[root]# ip link set can0 type can bitrate 500000
[root]# ip link set can1 type can bitrate 500000
[root]# ip link set can0 up
[root]# ip link set can1 up
[root]# candump can0 &
[root]# candump can1 &
[root]# cansend can0 001#224466
    can0 0001 [3] 22 44 66
[root]# can1 001 [3] 22 44 66
[root]# cansend can1 001#224466
    can0 001 [3] 22 44 66
    can1 001 [3] 22 44 66
[root]# cansend can0 001#113355
    can0 001 [3] 11 33 55
    can1 001 [3] 11 33 55
[root]# cansend can0 000#224466
    can0 000 [3] 22 44 66
```

### 7.3 OPC UA

OPC (originally known as “OLE for Process Control”, now “Open Platform Communications”) is a collection of multiple specifications, most common of which is OPC Data Access (OPC DA).

OPC Unified Architecture (OPC UA) was released in 2010 by the OPC Foundation as a backward incompatible standard to OPC Classic, under the name of IEC 62541.

OPC UA has turned away from the COM/DCOM (Microsoft proprietary technologies) communication model of OPC Classic, and switched to a TCP/IP based communication stack (asynchronous request/response), layered into the following:

- Raw connections
- Secure channels
- Sessions

#### 7.3.1 OPC introduction

OPC UA defines:

- The transport protocol for communication (that can take place over HTTP, SOAP/XML or directly over TCP).
- A set of 37 ‘services’ that run on the OPC server, and which clients call into, via an asynchronous request/response RPC mechanism.
- A basis for creating information models of data using object-oriented concepts and complex relationships.
The primary goal of OPC is to extract data from devices in the easiest way possible.

The Information Model provides a way for servers to not only provide data, but to do so in the most self-explanatory and intuitive way possible.

**Note:** Further references to 'OPC' in this document will imply OPC UA. OPC Classic is not discussed in this document.

Following are the typical scenarios for embedding an OPC-enabled device into a project:

- Manually investigate ("browse") the server’s Address Space looking for the data user need using a generic, GUI client (such as UaExpert from Unified Automation, or the FreeOpcUa covered in this chapter).
- Using References and Attributes, understand the format it is in, and the steps that may be needed to convert the data.
- Have a custom OPC client (integrated into the application) subscribe directly to data changes of the node that contains the desired data.

In a typical use case:

- The OPC server runs near the source of information (in industrial contexts, this means near the physical process – for example, on a PLC on a plant floor).
- Clients consume the data at runtime (for example, logging into a database, or feeding it into another industrial process).

OPC-enabled applications can be composed: an industrial device may run an OPC client and feed the collected data into another physical process, while also exposing the latter by running an OPC server.

### 7.3.2 The node model

Data in an OPC server is structured in **Nodes**. The collection of all nodes that an OPC server exposes to its clients is known as an **Address Space**. Some nodes have a predefined meaning, while others have meaning that is unique to the Information Model of that specific OPC server.

Every Node has the following **Attributes**:

- an **ID** (unique)
- a **Class** (what type of node it is)
- a **BrowseName** (a string for machine use)
- a **DisplayName** (a string for human use)
Figure 121. OPC UA address space

Shown on the left-hand side of the figure is the Address Space (collection of information that the server makes available to clients) of the OPC server found at opc.tcp://192.168.15.4:16664.

Selected is a node with NodeID ns=1;i=118, BrowseName=1:SJA1105 and of NodeClass Object. The full path of the selected node is 0:Root,0:Objects,1:SJA1105.

7.3.3 Node Namespaces

Namespaces are the means for separating multiple Information Models present in the same Address Space of a server.

- Nodes that do not have the ns= prefix as part of the NodeID have an implicit ns=0; prefix (are part of the namespace zero).
- Nodes in namespace * 0 have NodeID’s pre-defined by the OPC UA standard. For example, the 0:Server object, which holds self-describing information (capabilities, diagnostics, and vendor information), has a predefined NodeID of ns=0;i=2253;.

It is considered a good practice to not alter any of the nodes exposed in the namespace * 0.

7.3.4 Node classes

OPC nodes have an inheritance model, based on their NodeClass.

There are eight base node classes defined by the standard:
• Object
• Variable
• Method
• View
• ObjectType
• VariableType
• ReferenceType
• DataType

All nodes have the same base Attributes (inherited from the Node object), plus additional ones depending on their NodeClass.

7.3.5 Node graph and references

It may appear that nodes are only chained hierarchically, in a simple parent-child relationship. However, in reality nodes are chained in a complex directed graph, through References to other nodes.

![Hierarchy of the standard ReferenceTypes, defined in Part 3 of the OPC UA specification (Image taken from www.open62541.org)](image)

In OPC, even ReferenceTypes are Nodes, and as such are structured hierarchically, as can be seen in the figure above.

The definitions of all OPC ReferenceTypes can be found under the `0:Root,0:Types,0:ReferenceTypes` path.

The semantics of OPC references can be enriched by creating custom ReferenceType nodes.
Figure 123. The ‘Attributes’ and ‘References’ views of the FreeOpCua Client populated with details of the RGMII4 node

Selected in the Address Space is node ns=1;i=197. Conceptually, this represents one of the five Ethernet ports of the SJA1105 TSN switch. Its NodeClass is Object, but it has a reference of type HasTypeDefinition to NodeID ns=1;i=117 which is 1:EthPortType. For this reason, the 1:RGMII4 node is of the custom ObjectType EthPortType.

7.3.6 Open62541

Real-time Edge integrates the Open62541 software stack (https://open62541.org/). This supports both server-side and client-side API for OPC UA applications. Only server-side capabilities of open62541 are being shown here.

Open62541 is distributed as a C-based dynamic library (libopen62541.so). The services run on pthreads, and the application code runs inside an event loop.

Enable open62541 in Real-time Edge file ./recipes-nxp/packagegroups/packagegroup-real-time-edge-industrial.bb:

```
libopen62541
```

In order to install Open62541 example application, file "meta-real-time-edge/conf/distro/include/libopen62541.inc" has been included in distro configuration.

The following Open62541 example applications are included in the target image:
- open62541_access_control_client
7.3.7 OPC UA Pub/Sub over TSN

This section introduces OPC UA Pub/Sub and demonstrates how TSN can be used to make deterministic and reliable transmission of OPC UA Pub/Sub traffic as well as PTP traffic on a network co-existing with best effort traffic.

7.3.7.1 OPC UA Pub/Sub introduction

The 14th part of the OPC UA specification defines the OPC UA PubSub communication model. It provides an OPC UA Publish Subscribe model that complements the Client/Server communication model.

In PubSub, the participating OPC UA applications can assume the roles of Publishers and Subscribers. Publishers are the sources of data, while Subscribers consume that data. Communication in PubSub is message-based. Publishers send messages to a Message Oriented Middleware, without knowledge of what, if any, Subscribers there may be. Similarly, Subscribers express interest in specific types of data, and process messages that contain this data, without knowledge of what Publishers there are.

To cover a large number of use cases, OPC UA PubSub supports two largely different Message Oriented Middleware variants. These are:

1. A broker-based form, where the core component of the Message Oriented Middleware is a message Broker. Subscribers and Publishers use standard messaging protocols like AMQP or MQTT to communicate with the Broker.
2. A broker-less form, where the Message Oriented Middleware is the network infrastructure that is able to route datagram-based messages. Subscribers and Publishers use datagram protocols such as UDP or raw Ethernet as the transport protocol. In this form, the data sources (Publishers) and the data consumers (Subscribers) join a multicast group. Any data sent by a source to the group goes to all consumers subscribed to the same group. Joining is trivial in Ethernet (Layer 2): the network broadcasts multicast frames everywhere, leaving it to receivers to decide whether to pick up the frame based on the destination address. The OPC UA PubSub sample applications in this section will use this form.

Compared with client-server, the Publishers and Subscribers are decoupled. The number of Subscribers receiving data from a Publisher does not influence the Publisher. This makes PubSub suitable for applications where location independence and/or scalability are required.

One example use case for PubSub is generating logs to multiple systems. For example, sensors or actuators can write logs to a monitoring system, an HMI, an archive application for later querying, and so on. In this case, the data is sent cyclically.

7.3.7.2 OPC UA PubSub over TSN

In general, OPC UA operates at the upper layers of the OSI reference model for networking, whereas TSN is a Layer 2 protocol. TSN adds real-time capability to standard Ethernet. Operating at different layers, TSN and OPC UA PubSub complement each other, yielding a complete communication stack for the industrial Internet of Things. OPC UA standardizes the protocols by which applications exchange data and TSN enables this exchange to meet factories’ timing requirements.

One of the key things is to define a mechanism for OPC UA nodes to tell the TSN layers how to prioritize data streams. This cross-layer control is essential to enabling operations technology (OT) using the OPC UA framework to get the data they need when they need it. It also enables time-sensitive OT to coexist on the same network as information technology (IT) functions. In this section, standard Linux tools (that is, tc) are used to map packets from different sources to different traffic classes in order to use TSN features like IEEE 802.1AS and IEEE 802.1Qbv.

7.3.7.3 OPC UA PubSub components

The following figure shows the different components of OPC UA PubSub and their relation to each other. The WriterGroup, DataSetWriter, and PublishedDataSet components define the data acquisition for DataSets, message generation, and transmission on the Publisher side. These parameters should be known on the Subscriber side to configure DataSetReaders to filter and process DataSetMessages.
1. PubSubConnection: It represents settings needed for the transport protocol. One connection can have a number of writer groups and reader groups. A PubSub connection defines the used protocol and the network address for sending or receiving messages. In the case of using raw Ethernet as transport protocol, the network address can be an MAC multicast address. The Ethernet frame uses EtherType 0xB62C to encapsulate UADP (UA Datagram Protocol) NetworkMessages as payload without IP or UDP headers.

2. PublishedDataSet: It contains the collection of the published fields.

3. WriterGroup: Each writer group can have one or more DataSetWriters. A WriterGroup defines the timing (that is, publishing interval) and header settings for PubSub NetworkMessages sent by a Publisher.

4. DataSetWriter: It is the glue between the WriterGroup and the PublishedDataSet. Each DataSetWriter is bound to a single PublishedDataSet. A PublishedDataSet can have multiple DataSetWriters.

5. ReaderGroup: It is used to group a list of DataSetReaders and contains a few shared settings for them.

6. DataSetReader: It is the counterpart to a DataSetWriter on the Subscriber side. It defines the filter for the selection of the Publisher and DataSetWriter of interest. The parameters for the filter include the publisher identifier, WriterGroup identifier and DataSetWriter identifier.

7. SubscribedDataSet: Its parameters define the processing of the decoded DataSet in the Subscriber for one DataSetReader. The default processing is a mapping to target variables in the Subscriber address space.

7.3.7.4 OPC UA PubSub sample application

There are two sample applications for demonstrating OPC UA PubSub on NXP development boards. One acts as Publisher and the other acts as Subscriber.

On the Publisher:

1. A PubSubConnection is created with the required parameters passed in via command line arguments. These includes the network address URL (for example, opc.eth://01-00-5E-00-00-01) and the Ethernet interface (for example, eth1 for ENET2 on i.MX8M Plus LPDDR4 EVK). Also the Publisher ID is hard-coded to 2234.

2. A PublishedDataSet is added with several DataSetFields added. One of the DataSetFields is the CPU temperature measured by the thermal monitoring unit on i.MX8M Plus. Another DataSetField is the TX HW timestamp of the published packet.
3. A WriteGroup is added with WriterGroup ID hard-coded to 100 and the publishing interval set to 1 second. The Publisher transmits one packet per second cyclically. Each cycle aligns with whole second using Linux system clock \texttt{CLOCK_REALTIME}.

4. A DataSetWriter is created with DataSetWriter ID hard-coded to 62541.

On the Subscriber:

1. A PubSubConnection is created with the required parameters passed in via command line arguments. These includes the network address URL (for example, \texttt{opc.eth://01-00-5E-00-00-01}) and the Ethernet interface (for example, eth1 for ENET2 on i.MX8M Plus LPDDR4 EVK). Note that the Subscriber uses the same network address URL as the Publisher.

2. A ReaderGroup is added.
   \textbf{Note:} The Subscriber also runs cyclically with 1 second cycle time to receive packet. Each cycle aligns with whole second with 500 µs offset to account for the application delay on the publisher and the path delay from publisher to subscriber. Linux system clock \texttt{CLOCK_REALTIME} is used.

3. A DataSetReader is added and configured with Publisher ID of 2234, WriterGroup ID of 100, and DataSetWriter ID of 62541. Note that all these parameters match the corresponding settings on the Publisher in order to filter the DataSetMessages to be processed by the DataSetReader.

4. A SubscribedDataSet is added with a list of targetVariables. The targetVariables corresponds to the DataSetFields in the PublishedDataSet on the Publisher.

5. Besides the above, the RX HW timestamp of the received packet is taken and the path delay is calculated by subtracting the RX HW timestamp taken on the Subscriber from the TX HW timestamp taken on the Publisher for the same packet. To achieve this, both the Publisher and the Subscriber must have synchronized time. This is achieved by running gPTP.

Both the Publisher and the Subscriber also run a OPC UA server. Users can use a OPC UA client running on a host PC to browse the server’s Address Space on either the Publisher or the Subscriber.

7.3.7.5 OPC UA PubSub sample application over TSN

Hardware Requirements:

1. Two or three i.MX 8DXL LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, or i.MX 93 EVK boards
2. One LS1028ARDB board

Software Requirements:

1. linuxptp package which provides tools such as \texttt{ptp4l}, \texttt{phc2sys}, \texttt{phc_ctl}, and \texttt{hwstamp_ctl}.
2. iproute2 package which provides tools such as \texttt{tc}.
3. Open source OPC UA stack open62541 compiled as shared library (libopen62541.so).
4. OPC UA PubSub sample application \texttt{opcua_pubsub_publisher} and \texttt{opcua_pubsub_subscriber} under \texttt{/home/root/open62541_example}.

All the above software tools and binaries are already in the rootfs.

The following sections use i.MX 8M Plus LPDDR4 EVK board as an example.

7.3.7.5.1 Case #1: two i.MX8M Plus LPDDR4 EVK connected back-to-back

A simple setup could be made by connecting two i.MX8M Plus LPDDR4 EVK boards back-to-back via ENET2 as shown in Figure 125. One i.MX8M Plus LPDDR4EVK (Board A) acts as Publisher and the other (Board B) acts as Subscriber. Also the ENET1 interface on both boards is connected to LAN (that is, office network). Note that the actual device name in Linux might be different.
• On both boards, bring up ENET2 (that is, eth1):

```bash
# ip link set eth1 up
# ethtool eth1
```

Using the command `ethtool eth1` should display a message: `Link detected: yes`. Otherwise, check the hardware connection.

• On the Publisher (i.MX8M Plus LPDDR4 EVK - Board A), add one tc filter rule to match OPC UA PubSub packet (EtherType 0xb62c) on ENET2 (that is, eth1) and modify SKB priority to 2.

```bash
# tc qdisc add dev eth1 clsact
# tc filter add dev eth1 egress prio 1 u32 match u16 0xb62c 0xffff at -2 action skbedit priority 2
# tc filter show dev eth1 egress
```

• On both boards, run ptp4l for PTP time synchronization and run phc2sys to synchronize PHC clock to Linux system clock (CLOCK_REALTIME).

Also on the Subscriber (i.MX8M Plus LPDDR4 EVK - Board B), use `hwstamp_ctl` to change the RX hardware timestamp setting to 'time stamp any incoming packet' in order to get the RX hardware timestamp of the packets transmitted by the Publisher.

On the Publisher (i.MX8M Plus LPDDR4 EVK - Board A):

```bash
# cp /etc/ptp4l_cfg/gPTP.cfg .
# sed -i 's/priority1.*248/priority1		246/g' ./gPTP.cfg
# ptp4l -i eth1 -p /dev/ptp1 -f ./gPTP.cfg -m > /var/log/ptp4l.log 2>&1 &
# phc2sys -s eth1 -O 0 -S 0.00002 -m > /var/log/phc2sys.log 2>&1 &
```

On the Subscriber (i.MX8M Plus LPDDR4 EVK - Board B):

```bash
# ptp4l -i eth1 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m > /var/log/ptp4l.log 2>&1 &
# phc2sys -s eth1 -O 0 -S 0.00002 -m > /var/log/phc2sys.log 2>&1 &
# hwstamp_ctl -i eth1 -r 1
```
It is recommended to SSH to both boards to check the logs of ptp4l and phc2sys so that we can continue to execute other commands on the serial console. On both boards, observe the logs of ptp4l and phc2sys to check the time synchronization progress by using below commands:

```bash
# tail -f /var/log/ptp4l.log
# tail -f /var/log/phc2sys.log
```

On the Subscriber, the rms value reported by ptp4l shows the root mean square of the time offset between the PHC and the GM clock. If ptp4l consistently reports rms lower than 100 ns, the PHC is synchronized. Example ptp4l log below:

```
root@imx8mpevk:~# tail -f /var/log/ptp4l.log
ptp4l[101.594]: rms 23 max 42 freq -559 +/- 14 delay 779 +/- 0
ptp4l[102.596]: rms 7 max 15 freq -545 +/- 9 delay 780 +/- 0
ptp4l[103.599]: rms 698 max 1282 freq -1344 +/- 810 delay 781 +/- 0
ptp4l[104.600]: rms 242 max 491 freq -1259 +/- 293 delay 782 +/- 0
ptp4l[105.601]: rms 289 max 310 freq -583 +/- 107 delay 781 +/- 0
ptp4l[106.603]: rms 224 max 281 freq -396 +/- 12 delay 782 +/- 0
ptp4l[107.604]: rms 87 max 134 freq -432 +/- 18 delay 785 +/- 0
ptp4l[108.607]: rms 12 max 26 freq -497 +/- 14 delay 785 +/- 0
ptp4l[109.608]: rms 15 max 26 freq -531 +/- 8 delay 785 +/- 0
ptp4l[110.610]: rms 13 max 22 freq -540 +/- 8 delay 785 +/- 0
ptp4l[111.612]: rms 10 max 14 freq -547 +/- 6 delay 785 +/- 0
ptp4l[112.615]: rms 9 max 19 freq -533 +/- 11 delay 784 +/- 0
ptp4l[113.616]: rms 11 max 20 freq -521 +/- 10 delay 783 +/- 0
ptp4l[114.618]: rms 8 max 11 freq -537 +/- 9 delay 782 +/- 0
ptp4l[115.620]: rms 6 max 8 freq -533 +/- 8 delay 783 +/- 0
ptp4l[116.622]: rms 8 max 12 freq -531 +/- 11 delay 783 +/- 0
ptp4l[117.624]: rms 8 max 13 freq -534 +/- 11 delay 782 +/- 0
ptp4l[118.626]: rms 6 max 9 freq -539 +/- 7 delay 782 +/- 0
ptp4l[119.628]: rms 8 max 17 freq -529 +/- 9 delay 782 +/- 0
ptp4l[120.630]: rms 10 max 16 freq -518 +/- 10 delay 783 +/- 0
ptp4l[121.632]: rms 5 max 8 freq -527 +/- 7 delay 783 +/- 0
ptp4l[122.635]: rms 8 max 15 freq -534 +/- 10 delay 784 +/- 0
ptp4l[123.636]: rms 7 max 13 freq -536 +/- 9 delay 784 +/- 0
```

Figure 126. A sample ptp4l log

On both the Publisher and the Subscriber, the offset information reported by phc2sys shows the time offset between the PHC and the system clock (CLOCK_REALTIME). If phc2sys consistently reports offset lower than 100 ns, the System clock is synchronized. A sample phc2sys log is shown below:

```
root@imx8mpevk:~# tail -f /var/log/phc2sys.log
phc2sys[7349.412]: CLOCK_REALTIME phc offset 61.52s freq +58 delay 750
phc2sys[7350.413]: CLOCK_REALTIME phc offset -81.52s freq -66 delay 750
phc2sys[7351.413]: CLOCK_REALTIME phc offset -20.52s freq -29 delay 750
phc2sys[7352.413]: CLOCK_REALTIME phc offset -54.52s freq -39 delay 750
phc2sys[7353.414]: CLOCK_REALTIME phc offset 20.52s freq -21 delay 750
phc2sys[7354.414]: CLOCK_REALTIME phc offset -31.52s freq -24 delay 750
phc2sys[7355.414]: CLOCK_REALTIME phc offset 48.52s freq +46 delay 750
phc2sys[7356.415]: CLOCK_REALTIME phc offset -7.52s freq +5 delay 750
phc2sys[7357.415]: CLOCK_REALTIME phc offset 57.52s freq +47 delay 750
phc2sys[7358.416]: CLOCK_REALTIME phc offset 20.52s freq +13 delay 750
phc2sys[7359.416]: CLOCK_REALTIME phc offset -28.52s freq -29 delay 750
phc2sys[7360.416]: CLOCK_REALTIME phc offset 26.52s freq +16 delay 750
phc2sys[7361.417]: CLOCK_REALTIME phc offset 20.52s freq +18 delay 750
phc2sys[7362.417]: CLOCK_REALTIME phc offset -14.52s freq -10 delay 750
```

Figure 127. A sample phc2sys log

After establishing the time synchronization successfully on both the Publisher and the Subscriber, we can configure TSN Qbv and run the OPC UA PubSub sample applications as in the following steps.
On the Publisher (i.MX8M Plus LPDDR4 EVK - Board A), configure TSN Qbv on ENET2 (that is, eth1) to map SKB priority to traffic class to hardware queue as below, set gate control list to have 2 entries and total cycle time of 1ms (queue 4 has 500 µs for best effort traffic, queue 0 and queue 2 share 500us for OPC UA PubSub and PTP traffic as well as other traffic like ping), also set base time to 1ms so that the schedule is aligned to 1ms. This is just an example configuration for the schedule.

- SKB priority 0 -> traffic class 0 -> queue 0
- SKB priority 1 -> traffic class 1 -> queue 1
- SKB priority 2 -> traffic class 2 -> queue 2
- SKB priority 3 -> traffic class 3 -> queue 3
- SKB priority 4 -> traffic class 4 -> queue 4

```
# tc qdisc replace dev eth1 parent root handle 100 taprio num_tc 5 map 0 1 2 3 4 queues 1@0 1@1 1@2 1@3 1@4 base-time 001000000 sched-entry S 0x10 500000 flags 2
# tc -g qdisc show dev eth1
```

Together with the tc filter rule configured previously, the above TSN Qbv configuration on ENET2 distributes OPC UA PubSub traffic into Tx hardware queue 2, PTP traffic into Tx hardware queue 0. Also, we will send best effort traffic to Tx hardware queue 4. Other traffic such as pings can still go into Tx hardware queue 0. Because the OPC UA PubSub and PTP traffic have different Tx hardware queues and time slot than the best effort traffic, the latter cannot influence the former.

On the Subscriber (i.MX8M Plus LPDDR4 EVK - Board B), run the OPC UA PubSub Subscriber sample application. Run the Subscriber application before the Publisher application so that no packets sent by the Publisher are missed.

Note that octets in the MAC address should be separated by hyphens (-).

```
# /home/root/open62541_example/opcua_pubsub_subscriber -u opc.eth://01-00-5E-00-00-01 -d eth1
```

On the Publisher (i.MX8M Plus LPDDR4 EVK - Board A), run the OPC UA PubSub Publisher sample application.

Note that octets in the MAC address should be separated by hyphens (-).

```
# /home/root/open62541_example/opcua_pubsub_publisher -u opc.eth://01-00-5E-00-00-01 -d eth1
```

Example log on the Publisher:

```
```

Example log on the Subscriber:

```
[2020-09-20 11:02:26.438 (UTC+0000) info/overland] Publisher thread priority : 78
```

Figure 128. Example log on the Publisher
On a PC connected to office network and with OPC UA Client installed (that is, UaExpert as in below snapshots), we can browser either the OPC UA server’s Address Space on either the Publisher or the Subscriber. (We assume that eth0 has obtained the IP address by DHCP automatically).

The URL of the OPC UA server on the Publisher is below:
```
opc.tcp://<IP_of_eth0_on_Publisher>:4840/
```

The URL of the OPC UA server on the Subscriber is below:
```
opc.tcp://<IP_of_eth0_on_Subscriber>:4801/
```

Example snapshot of UaExpert connected to the Publisher:

Example snapshot of UaExpert connected to the Subscriber:
On the UAExpert client connected to the Subscriber, we can observe the CPU temperature published by the Publisher and the path delay from Publisher to Subscriber which is close to 800 ns.

- On the Publisher (i.MX8M Plus LPDDR4 EVK - Board A), we can use pktgen to simulate high load best effort traffic which is sent to queue 4 of ENET2.

```
# /home/root/samples/pktgen/pktgen_sample01_simple.sh -i eth1 -q 4 -s 1000 -n 0
```

The OPC UA PubSub traffic and PTP traffic are protected by TSN Qbv by having different Tx hardware queue and time slot than the best effort traffic. Hence, users can see consistent output on the console of the Publisher and the Subscriber, and the path delay from Publisher to Subscriber is still close to 800 ns.

In case TSN Qbv was not configured, after pktgen starts running, various issues may happen. First of all, the ptp4l application will show timeout issue as below.

Example error log of ptp4l on the Publisher:

```
[435.587] timed out while polling for tx timestamp
```

Example error log of ptp4l on the Subscriber

```
[435.600] selected local clock 00049ef.06f61b as best master
```

Example error log of ptp4l on the Subscriber

```
[435.600] selected local clock 00049ef.06f61b as best master
```

Example error log of ptp4l on the Subscriber

```
[435.600] selected local clock 00049ef.06f61b as best master
```

Example error log of ptp4l on the Subscriber

```
[435.600] selected local clock 00049ef.06f61b as best master
```

Example error log of ptp4l on the Subscriber

```
[435.600] selected local clock 00049ef.06f61b as best master
```

Example error log of ptp4l on the Subscriber

```
[435.600] selected local clock 00049ef.06f61b as best master
```
The OPC UA PubSub sample application may also be impacted by best effort traffic without TSN, especially when the publish cycle time is very short (that is, 2 ms). Note that the publish cycle time is hard-coded to 1 second in the sample application to make observation easier on the Subscriber console as well as on the UaExpert client. Several issues can be observed under high load best effort traffic without TSN. For example, the Publisher application may display warning message `timed out while polling for tx timestamp!`. The Subscriber application may show that the packet sequence number stops incrementing, and the path delay from Publisher to Subscriber displayed on UaExpert may show a large number due to packet transmission delay. When this issue happens, it can only be recovered by restarting both the Publisher and Subscriber applications.

7.3.7.5.2 Case #2: two i.MX 8M Plus LPDDR4 EVK boards connected to LS1028ARDB TSN switch

The setup could use one LS1028ARDB as TSN switch plus two or three i.MX 8M Plus LPDDR4 EVK boards. One i.MX8M Plus LPDDR4 EVK (Board A) acts as Publisher and others act as Subscribers. The block diagram of this setup is below. The ENET2 interface on each i.MX8M Plus LPDDR4 EVK is connected to the switch port on LS1028ARDB. Also, the ENET1 interface on each i.MX8M Plus LPDDR4 EVK is connected to LAN (that is, office network). Note that the actual device name in Linux may change.

![Figure 134. Two i.MX 8M Plus LPDDR4 EVK boards connected to LS1028ARDB TSN switch](image)

The following steps assume that two i.MX8M Plus LPDDR4 EVK boards are used. Board A acts as Publisher and Board B acts as Subscriber. In this setup, the switch port `swp0` is the ingress port for OPC UA PubSub traffic and best effort traffic. The switch port `swp1` is the egress ports for OPC UA PubSub traffic and best effort traffic.

Since the TSN switch on LS1028ARDB uses the value of VLAN PCP field to map traffic to different TX hardware queue on egress switch port (that is, `swp1`), we add VLAN header to the OPC UA PubSub packet and best effort packet. Note that the PTP packet is untagged without VLAN header.
• On LS1028ARDB, configure Ethernet bridge on TSN switch and enable VLAN filtering.

```bash
# ip link set eno2 up
# ip link set swp0 up
# ip link set swp1 up
# ip link add name br0 type bridge vlan_filtering 1
# ip link set dev swp0 master br0
# ip link set dev swp1 master br0
# ip link set dev br0 up
# bridge vlan add dev swp0 vid 100
# bridge vlan add dev swp1 vid 100
# bridge vlan show
```

• On both the Publisher (i.MX 8M Plus LPDDR4 EVK - Board A) and the Subscriber (i.MX 8M Plus LPDDR4 EVK - Board B), bring up ENET2 (that is, eth1):

```bash
# ip link set eth1 up
```

• On the Publisher (i.MX 8M Plus LPDDR4 EVK - Board A), add one tc filter to match OPC UA PubSub packet (EtherType 0xb62c) on ENET2 (that is, eth1) and modify SKB priority to 2.

```bash
# tc qdisc add dev eth1 clsact
# tc filter add dev eth1 egress prio 1 u32 match u16 0xb62c 0xffff at -2 action skbedit priority 2
# tc filter show dev eth1 egress
```

• On each board, run `ptp4l` for PTP time synchronization and run `phc2sys` to synchronize PHC clock to Linux system clock (CLOCK_REALTIME).

Also on the Subscriber (i.MX8M Plus LPDDR4 EVK - Board B), use `hwstamp_ctl` to change the RX hardware timestamp setting to 'time stamp any incoming packet' in order to get the RX hardware timestamp of the packets transmitted by the Publisher.

On LS1028ARDB:

```bash
# ptp4l -i swp0 -i swp1 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m > /var/log/ptp4l.log 2>&1 &
# phc2sys -s swp0 -O 0 -S 0.00002 -m > /var/log/phc2sys.log 2>&1 &
```

On the Publisher (i.MX 8M Plus LPDDR4 EVK - Board A):

```bash
# cp /etc/ptp4l_cfg/gPTP.cfg .
# sed -i 's/priority1.*248/priority1		246/g' ./gPTP.cfg
# ptp4l -i eth1 -p /dev/ptp1 -f ./gPTP.cfg -m > /var/log/ptp4l.log 2>&1 &
# phc2sys -s eth1 -O 0 -S 0.00002 -m > /var/log/phc2sys.log 2>&1 &
```

On the Subscriber (i.MX 8M Plus LPDDR4 EVK - Board B):

```bash
# ptp4l -i eth1 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m > /var/log/ptp4l.log 2>&1 &
# phc2sys -s eth1 -O 0 -S 0.00002 -m > /var/log/phc2sys.log 2>&1 &
# hwstamp_ctl -i eth1 -r 1
```

On each board, observe the logs of ptp4l and phc2sys to check the time synchronization progress by using the below commands: (In order to continue to execute other commands on the serial console, performing an SSH to each board to check the logs of ptp4l and phc2sys is recommended).

```bash
# tail -f /var/log/ptp4l.log
# tail -f /var/log/phc2sys.log
```

On LS1028ARDB and the Subscriber, the rms value reported by ptp4l shows the root mean square of the time offset between the PHC and the GM clock. If ptp4l consistently reports rms lower than 100 ns, the PHC is synchronized. Refer to the example log of ptp4l in the back-to-back case.
On each board, the offset information reported by phc2sys shows the time offset between the PHC and the system clock (CLOCK_REALTIME). If phc2sys consistently reports offset lower than 100 ns, the System clock is synchronized. Refer to the example log of phc2sys in the back-to-back case.

After establishing the time synchronization successfully on each board, users can configure TSN Qbv and run the OPC UA PubSub sample applications using the following steps.

- On the Publisher (i.MX 8M Plus LPDDR4 EVK - Board A), configure TSN Qbv on ENET2 to map SKB priority to traffic class to hardware queue as below, set gate control list to have 2 entries and total cycle time of 1 ms (queue 4 has 500 μs for best effort traffic, queue 0 and queue 2 share 500 μs for OPC UA PubSub and PTP traffic as well as other traffic like ping), also set base time to 1 ms so that the schedule is aligned to 1 ms. This is just an example configuration for the schedule.

  SKB priority 0 -> traffic class 0 -> queue 0
  SKB priority 1 -> traffic class 1 -> queue 1
  SKB priority 2 -> traffic class 2 -> queue 2
  SKB priority 3 -> traffic class 3 -> queue 3
  SKB priority 4 -> traffic class 4 -> queue 4

  # tc qdisc replace dev eth1 parent root handle 100 taprio num_tc 5 map 0 1 2 3 4 queues 1@0 1@1 1@2 1@3 1@4 base-time 001000000 sched-entry S 0x10 50000000 flags 2
  # tc -g qdisc show dev eth1

  Together with the tc filter rule configured previously, the above TSN Qbv configuration on ENET2 distributes OPC UA PubSub traffic into TX hardware queue 2, PTP traffic into TX hardware queue 0. Also, send best effort traffic to TX hardware queue 4. Other traffic like ping can still go into TX hardware queue 0. Because the OPC UA PubSub and PTP traffic have different TX hardware queues and time slot than the best effort traffic, the latter cannot influence the former.

- On LS1028ARDB, configure TSN Qbv on swp1, set gate control list to have 2 entries and total cycle time of 200 μs. Ensure that queue 4 has 500 μs for best effort traffic, queue 0 and queue 2 share 500 μs for OPC UA PubSub and PTP traffic as well as other traffic such as ping. In addition, set the base time to 1 ms so that the schedule is aligned to 1 ms as 1 ms. Note that the TSN Qbv configuration on LS1028ARDB TSN switch is used to protect the OPC UA PubSub traffic from traffic which may enter the switch from other switch ports. In this use case, it is optional and is used for demonstration purpose only.

  # tc qdisc replace dev swp1 root taprio num_tc 8 map 0 1 2 3 4 5 6 7 queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 base-time 001000000 sched-entry S 0x10 50000000 flags 0x2
  # tc -g qdisc show dev swp1

- With the above TSN Qbv configuration on egress switch port swp1,
  - OPC UA PubSub traffic goes into TX hardware queue 2 (Add VLAN header with PCP field set to 2 for OPC UA PubSub packet).
  - The best effort traffic goes into TX hardware queue 4 (Add VLAN header with PCP field set to 4 using pktgen for generating best effort traffic). Note that the PTP traffic is untagged without VLAN header and will use TX hardware queue 0 of swp1 to transmit to the Subscriber. Similar to the TSN Qbv configuration on Publisher, the OPC UA PubSub and PTP traffic have different TX hardware queues and time slot than the best effort traffic, the latter cannot influence the former.

- On the Subscriber (i.MX8M Plus LPDDR4 EVK - Board B), run the OPC UA PubSub Subscriber sample application. Run the Subscriber application before the Publisher application so that no packet sent by the Publisher is missed. Note that in the URL of below command 100.2 means VLAN ID 100 and PCP value 2 and it is separated from the MAC address using a colon.

  # /home/root/open62541_example/opcua_pubsub_subscriber -u opc.eth://01-00-5E-00-00-01:100.2 -d eth1
• On the Publisher (i.MX8M Plus LPDDR4 EVK - Board A), run the OPC UA PubSub Publisher sample application. Note that in the URL of below command 100.2 means VLAN ID 100 and PCP value 2 and it is separated from the MAC address using a colon.

```bash
#/home/root/open62541_example/opcua_pubsub_publisher -u opc.eth://01-00-5E-00-00-01:100.2 -d eth1
```

Example log on the Publisher:

![Example log on the Publisher](image1)

Example log on the Subscriber:

![Example log on the Subscriber](image2)

• On a PC connected to office network and with OPC UA Client installed (that is, UaExpert as in below snapshots), we can browser either the OPC UA server’s Address Space on either the Publisher or the Subscriber. (We assume that eth0 have got IP address by DHCP automatically.

The URL of the OPC UA server on the Publisher is below:

```plaintext
opc.tcp://<IP_of_eth0_on_Publisher>:4840/
```

The URL of the OPC UA server on the Subscriber is below:

```plaintext
opc.tcp://<IP_of_eth0_on_Subscriber>:4801/
```

Refer to the example snapshot of UaExpert in the back-to-back case. On the UaExpert client connected to the Subscriber, we can observe the CPU temperature published by the Publisher and the path delay from Publisher to Subscriber which is around 4 μs. Compared to the 800 ns in the back-to-back case, the increased path delay is added by the bridge.
On the Publisher (i.MX8M Plus LPDDR4 EVK - Board A), we can use pktgen to simulate high load best effort traffic with VLAN ID set to 100 and VLAN PCP set to 4 in VLAN header.

```
# cp /home/root/samples/pktgen/pktgen_sample01_simple.sh /home/root/samples/pktgen_sample01_simple_vlan.sh
# sed -i '/^UDP_MAX=.*/a VLAN_ID=100\nVLAN_P=4' /home/root/samples/pktgen/pktgen_sample01_simple_vlan.sh
# /home/root/samples/pktgen/pktgen_sample01_simple_vlan.sh -i eth1 -q 4 -s 1000 -n 0
```

**Note:** In order to protect the OPC UA PubSub traffic and PTP traffic, the TSN Qbv should be configured to have different TX hardware queue and time slot from the best effort traffic on both the Publisher and TSN switch. This ensures that users can get consistent output on the console of the Publisher and the Subscriber, and the path delay from Publisher to Subscriber is still around 4 μs.

In case TSN Qbv was not configured, after pktgen starts running, various issues might occur. Refer to the issues detailed in the back-to-back case.

On LS1028ARDB, it is possible to check the status of TX packets of swp1 by using the command below:

```
# ethtool -S swp1 | grep -i "tx_green_prio_"
```

Example log below: (tx_green_prio_0 mainly for PTP traffic, tx_green_prio_2 mainly for OPC UA PubSub traffic, tx_green_prio_4 mainly for best effort traffic generated by pktgen).

```
root@ls1028ardb:~# ethtool -S swp1 | grep -i "tx_green_prio_"
 tx_green_prio_0: 6207
tx_green_prio_1: 0
tx_green_prio_2: 242
tx_green_prio_3: 0
tx_green_prio_4: 20039651
tx_green_prio_5: 0
tx_green_prio_6: 0
tx_green_prio_7: 16
```

Figure 137. Sample log after checking the stats of TX packets of swp1

### 7.3.8 OPC UA client installation and usage

#### 7.3.8.1 UaExpert

The UaExpert is an OPC UA Client developed by Unified Automation. It is free to download. Before downloading, you need to register on the following link to create an free account. Then login using your account, download the installation file and install it on a host PC. The UaExpert is available for both Windows and Linux.


Below steps shows how to use UaExpert to connect to an OPC UA server on a Window10 PC.

- Open the UaExpert GUI. Click on the 'Add Server' button.
Figure 138. UaExpert GUI
• The 'Add Server' window will pop up. Select Custom Discovery and double click '<Double click to Add Server...>'. The 'Enter URL' window will pop up. Input IP address and port number of the OPC UA server separated by colon. For example, the complete URL is opc.tcp://10.193.20.15:4840 in below snapshot. Click OK.

Figure 139. Adding the OPC UA server
• The new server (that is, opc.tcp://10.193.20.15:4840) will be listed under Custom Discovery. Click to expand it. Then click to expand 'open62541-based OPC UA Application (opc.tcp)'. A 'Replaced Hostname' window will pop up. Click 'Yes'.

![Figure 140. Server listed under Custom Discovery](image)

• Click to select 'None – None (…)' and click OK.

![Figure 141. Selecting the hostname](image)

• Right click on the server listed under ‘Servers’ and click ‘Connect’.
You are now connected to the OPC UA server and can browse or monitor its object. To monitor the value of an object, you can drag and drop the object to the ‘Data Access View’ area.
7.3.8.2 FreeOpcUa

FreeOpcUa is a project to implement an open-source (LGPL/GPL) OPC UA stack and associated tools. A GUI client from FreeOpcUa is available. It is written using freeropcua python api and pyqt. Use below command to install it on a Linux PC using pip3. Make sure python3 and python3-pip is installed.

```bash
$ sudo pip3 install opcua-client
```

For installation on Windows, please refer to the instructions available from below link:

[https://github.com/FreeOpcUa/opcua-client-gui](https://github.com/FreeOpcUa/opcua-client-gui)

Below steps shows how to use FreeOpcUa GUI client to connect to an OPC UA server on a Ubuntu 18.04 PC.

1) Launch the FreeOpcUa GUI client from the terminal on the Linux host PC:

```bash
$ opcua-client
```

In the FreeOpcUa GUI client, input the URL (that is, `opc.tcp://10.193.20.15:4840`) and click 'Connect'. You are now connected to the OPC UA server and can browse or monitor its object.
7.4 NETCONF/YANG

This chapter provides an overview of the NETCONF protocol and Yang (a data modeling language for NETCONF). It describes the applications, installation and configuration steps, operation examples, Web UI demo, and troubleshooting aspects of NETCONF. It also describes how to enable the NETCONF feature in this Real-time Edge software.

7.4.1 Overview

The NETCONF protocol defines a mechanism for device management and configuration retrieval and modification. It uses a remote procedure call (RPC) paradigm and a system of exposing device (server) capabilities, which enables a client to adjust to the specific features of any network equipment. NETCONF further distinguishes between state data (which is read-only) and configuration data (which can be modified). Any NETCONF communication happens on four layers as shown in the table below. XML is used as the encoding format.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Content</td>
<td>Configuration data, Notification data</td>
</tr>
<tr>
<td>2</td>
<td>Operations</td>
<td><code>&lt;edit-config&gt;</code></td>
</tr>
<tr>
<td>3</td>
<td>Messages</td>
<td><code>&lt;rpc&gt;</code>, <code>&lt;rpc-reply&gt;</code>, <code>&lt;notification&gt;</code></td>
</tr>
<tr>
<td>4</td>
<td>Secure</td>
<td>Transport SSH</td>
</tr>
</tbody>
</table>

Table 86. The NETCONF layers

YANG is a standards-based, extensible, hierarchical data modeling language that is used to model the configuration and state data used by NETCONF operations, remote procedure calls (RPCs), and server event notifications. The device configuration data are stored in the form of an XML document. The specific nodes in the document as well as the allowed values are defined by a model, which is usually in YANG format or possibly transformed into YIN format with XML-based syntax. There are many such models created directly by IETF to further support standardization and unification of the NETCONF interface of the common network devices. For example, the general system settings of a standard computer are described in the IETF-system model (rfc7317) or the configuration of its network interfaces defined by the IETF-interfaces model (rfc7223). However, it is common for every system to have some specific parts exclusive to it. In that case there are mechanisms defined to enable extensions while keeping the support for the standardized core. Also, as this whole mechanism is designed in a liberal fashion, the configuration does not have to concern strictly network. Even RPCs additional to those defined by NETCONF can be characterized. Therefore, it allows the client to request an explicit action from the server.

A YANG module defines a data model through its data, and the hierarchical organization of and constraints on that data. A module can be a complete, standalone entity, or it can reference definitions in other modules and sub-modules as well as augment other data models with additional nodes. The module dictates how the data is represented in XML.

A YANG module defines not only the syntax but also the semantics of the data. It explicitly defines relationships between and constraints on the data. This enables user to create syntactically correct configuration data that meets constraint requirements and enables user to validate the data against the model before uploading it and committing it on a device.

For information about NETCONF, see RFC 6241, NETCONF Configuration Protocol.

For information about YANG, see RFC 6020, YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF), and related RFCs.
7.4.2 Netopeer2

7.4.2.1 Overview

Netopeer2 is a set of tools implementing network configuration tools based on the NETCONF protocol. This is the second generation of the toolset, originally available as the Netopeer project. It is based on the new generation of the NETCONF and YANG libraries - libyang and libnetconf2. The Netopeer2 server uses sysrepo as a NETCONF datastore implementation. In Real-time Edge software, version v0.7-r2 was used. It allows developers to control their devices via NETCONF and operators to connect to their NETCONF-enabled devices.

![High level architecture of Netopeer and sysrepo](image)

7.4.2.2 Installing Netopeer2-cli on Ubuntu18.04

Use the following steps for installing Netopeer2-cli on Ubuntu18.04 operating systems.

1. Install the following packages:

   ```bash
   $ sudo apt install -y git cmake build-essential bison autoconf dh-autoreconf flex
   $ sudo apt install -y libavl-dev libprotobuf-c-dev protobuf-c-compiler zlib1g-dev
   $ sudo apt install -y libgcrypt20-dev libssh-dev libev-dev libpcre3-dev
   ```

2. Install libyang:

   ```bash
   $ git clone https://github.com/CESNET/libyang.git
   $ cd libyang;
   $ git checkout v1.0-r4 -b v1.0-r4
   $ mkdir build; cd build
   $ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
   $ make
   $ sudo make install
   ```

3. Install sysrepo (v0.7.8):

   ```bash
   $ git clone https://github.com/sysrepo/sysrepo.git
   $ cd sysrepo
   ```
$ git checkout v0.7.8 -b v0.7.8
$ mkdir build; cd build
$ cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX:PATH=/usr ..
$ make
$ sudo make install

4. Install libnetconf2:

$ git clone https://github.com/CESNET/libnetconf2.git
$ cd libnetconf2
$ git checkout v0.12-r2 -b v0.12-r2
$ mkdir build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ..
$ make
$ sudo make install

5. Install protobuf:

$ git clone https://github.com/protocolbuffers/protobuf.git
$ cd protobuf
$ git checkout v3.18.0 -b v3.18.0
$ git submodule update --init --recursive
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
$ sudo ldconfig # refresh shared library cache.

6. Install Netopeer2-cli(v0.7-r2):

$ git clone https://github.com/CESNET/Netopeer2.git
$ cd Netopeer2
$ git checkout v0.7-r2 -b v0.7-r2
$ cd cli
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr .
$ make
$ sudo make install

7.4.2.3 Sysrepo

Sysrepo is an YANG-based configuration and operational state data store for Unix/Linux applications. Applications can use sysrepo to store their configuration modeled by provided YANG model instead of using e.g. flat configuration files. In Real-time Edge software, version v0.7.8 was used. Sysrepo will ensure data consistency of the data stored in the datastore and enforce data constraints defined by YANG model. Applications can currently use C language API of sysrepo Client Library to access the configuration in the datastore, but the support for other programming languages is planned for later too (since sysrepo uses Google Protocol Buffers as the interface between the datastore and client library, writing of a native client library for any programming language that supports GPB is possible).

For information about sysrepo, see:

7.4.2.4 Netopeer2 server

Netopeer2 software is a collection of utilities and tools to support the main application, Netopeer2 server, which is a NETCONF server implementation. It uses libnetconf2 for all NETCONF communication. Conforming to the relevant RFCs and still being part of the aforementioned library, it supports the mandatory SSH as the
transport protocol. Once a client successfully connects using either of these transport protocols and establishes a NETCONF session, it can send NETCONF RPCs and the Netopeer2 server responds with correct replies.

The following set of tools are a part of the Netopeer server:

- Netopeer2-keystored as a tool for the storage and process of keys.
- Netopeer2-server as the main service daemon integrating the SSH server.

7.4.2.5 Netopeer2 client

Netopeer2-cli is a CLI interface that allows user to connect to a NETCONF-enabled device and obtain and manipulate its configuration data.

This application is a part of the Netopeer2 software bundle, but compiled and installed separately. It is a NETCONF client with a command line interface developed and primarily used for Netopeer2 server testing, but allowing all the standards and even some optional features of a full-fledged NETCONF client.

Netopeer2-cli serves as a generic NETCONF client providing a simple interactive command line interface. It allows user to establish a NETCONF session with a NETCONF-enabled device on the network and to obtain and manipulate its configuration data.

7.4.2.6 Workflow in application practice

In practical application, we use the YANG language to model the device and generate the YANG model. The model is then instantiated to generate configuration files in XML format. The device was then configured using this configuration file as input via netopeer.

![Figure 146. Workflow for netopeer](image)

7.4.3 Configuration

7.4.3.1 Enabling NETCONF feature

This feature is enabled by default in Real-time Edge software.
The below packages are enabled by default in Real-time Edge software:

```shell
netopeer2-keystore netopeer2-server real-time-edge-sysrepo
```

`sysrepo-tns` is a daemon application to implement tns configuration based on `sysrepo`. It was enabled for LS1028ARDB, i.MX 8DXL LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK and i.MX 93 EVK.

**Note:**
- For LS1028ARDB board, Qbv, Qbu, Qci, stream identification in CB, IP, MAC, and VLAN are supported.

### 7.4.3.2 Netopeer2-server

The netopeer2-server is the NETCONF protocol server running as a system daemon. The netopeer2-server is based on `sysrepo` and `libnetconf2` library.

- `-U` listen locally on a unix socket
- `-d` debug mode (do not daemonize and print verbose messages to stderr instead of syslog)
- `-V`: Show program version.
- `-v` level verbose output level (0: errors, 1: errors and warnings, 2: errors, warnings, and verbose messages).

### 7.4.3.3 Netopeer2-cli

The netopeer2-cli is a command line interface similar to the NETCONF client. It serves as a generic NETCONF client providing a simple interactive command line interface. It allows user to establish a NETCONF session with a NETCONF-enabled device on the network and to obtain and manipulate its configuration data. netopeer2-cli is limited to a single NETCONF connection at a time via a forward or a reverse (Call Home) connecting method.

#### 7.4.3.3.1 Netopeer2 CLI commands

Following are the Netopeer2 CLI commands:

1. **help**: Displays a list of commands. The `--help` option is also accepted by all commands to show detailed information about the command.
2. **connect**: Connects to a NETCONF server.
   ```bash
   ```
   The `connect` command has the following arguments:
   - `--login` user name: Specifies the user to log in as on the NETCONF server. If not specified, current local user name is taken.
   - `--port` num
     - Port to connect to on the NETCONF server. By default, port 830 for SSH transport is used.
   - `host`
     - Hostname or ip-address of the target NETCONF server.
3. **disconnect**: Disconnects from a NETCONF server.
4. **commit**
   - Performs the NETCONF `commit` operation. For details, see RFC 6241, section 8.3.4.1.
5. **copy-config**: Performs NETCONF `copy-config` operation. For details, see RFC 6241 section 7.3.
   ```bash
   ```
Where, the arguments are the following:

- **--defaults** mode: Use with the --defaults capability with specified retrieval mode. For details, refer to the RFC 6243 section 3 or WITH-DEFAULTS section of this manual.

- **--target** datastore: Specifies the target datastore for the copy-config operation. For description of the datastore parameter, refer to Section 7.4.3.3.2.

- **--source** datastore: Specifies the source datastore for the copy-config operation. For description of the datastore parameter, refer to Section 7.4.3.3.2.

6. **delete-config** Performs NETCONF delete-config operation. Refer to section 7.4 of the RFC 6241 specification for more details.

```
delete-config [--help] --target startup|url:<url>
```

Where

- **target** datastore: Specifies the target datastore for the delete-config operation.

7. **edit-config**

Performs NETCONF edit-config operation. For details, refer to RFC 6241 section 7.2.

```
edit-config [--help] --target running|candidate (--config=<file> | --url <url>)
[---defop merge|replace|none] [---test set|test-only|test-then-set] [--error stop|continue|rollback]
```

Where

- **--defop** operation:
  Specifies default operation for applying configuration data.
  - **merge**: Merges configuration data at the corresponding level. By default, the value is merge.
  - **replace**: Edits configuration data completely replaces the configuration in the target datastore.
  - **none**: The target datastore is unaffected by the edit configuration data, unless and until the edit configuration data contains the operation attribute to request a different operation. For more information, see the EDIT-CONFIG section of RFC 6241.

  **Note**: To delete non-mandatory items, nc:operation="delete" should be added into the end of start tag of the item to be deleted. At the same time, the namespace xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" should also be added to the start tag of the root node. Mandatory items cannot be deleted individually. They can only be deleted with their parent node.

- **--error** action
  Sets reaction to an error.
  - **Stop**: aborts the operation on first error. This is the default value.
  - **Continue**: Continues to process configuration data on error. The error is recorded and negative response is returned.
  - **Rollback**: Stops the operation processing on error and restore the configuration to its complete state at the start of this operation. This action is available only if the server supports rollback-on-error capability (see RFC 6241 section 8.5).

- **--test** option
  Performs validation of the modified configuration data. This option is available only if the server supports :validate:1.1 capability (see RFC 6241 section 8.6).
  - **set**: Does not perform validation test.
  - **test-only**: Does not apply the modified data, only performs the validation test.
  - **test-then-set**: Performs a validation test before attempting to apply modified configuration data. test-then-set is the default value.

- **--config** file
  - Specifies path to a file containing edit configuration data. The content of the file is placed into the config element of the edit-config operation. Therefore, it does not have to be a well-formed XML
document with only a single root element. If neither --config nor --url is specified, user is prompted to write edit configuration data manually. For examples, see the EDIT-CONFIG section of RFC 6241.

- **--url URI**
  - Specifies remote location of the file containing the configuration data hierarchy to be modified, encoded in XML under the element config in the urn:ietf:params:xml:ns:netconf:base:1.0 namespace. Note, that this differs from file parameter, where the config element is not expected.

- **--target**
  - Target datastore to modify. For description of possible values, refer to Section 7.4.3.3.2. Note that the url configuration datastore cannot be modified.

8. **get**: Performs NETCONF get operation. Receives both the status as well as configuration data from the current running datastore. Refer to section 7.7 of the RFC 6241 specification for more details. The command format is as follows:

```bash
```

- **--defaults mode**
  - Use with the -defaults capability with specified retrieval mode. For further details, refer to the Section 3 or 'WITH-DEFAULTS' section of the RFC 6241 specification.

- **--filter [file]**
  - Specifies if the request will contain subtree filter (RFC 6241 section 6). The option is able to accept path to the file containing the filter specification. If the path is not specified, user is prompted to write the filter specification manually.

9. **get-config**: Performs NETCONF get-config operation. Retrieves only configuration data from the specified target datastore. For details, refer to RFC 6241 section 7.1.

```bash
  [--defaults report-all|report-all-tagged|trim|explicit] [--out <file>]
```

- **--defaults mode**
  - Use: with the -defaults capability with specified retrieval mode. For more details see RFC 6243 section 3 or WITH-DEFAULTS section of this manual.

10. **--target**
  - Specifies if the request will contain subtree filter (RFC 6241 section 6). The option is able to accept path to the file containing the filter specification. If the path is not specified, user is prompted to write the filter specification manually.

11. **--target**
  - Target datastore to retrieve. For description of possible values, refer to Section 7.4.3.3.2. Note, that the url configuration datastore cannot be retrieved.

12. **lock**: Performs the NETCONF lock operation to lock the entire configuration datastore of a server. For details, see RFC 6241 section 7.5.

```bash
lock [--help] --target running|startup|candidate
```

Where the

- **--target**: specifies the target datastore to lock. For description of possible values, refer to Section 7.4.3.3.2. Note, that the url configuration datastore cannot be locked.

13. **unlock**: Performs the NETCONF unlock operation to release a configuration lock, previously obtained with the lock operation. Refer to section 7.6 of the RFC 6241 specification for more details.

```bash
unlock [--help] --target running|startup|candidate
```
where
  • --target: specifies the target datastore to unlock. For description of possible values, refer to Section 7.4.3.3.2. Note, that the url configuration datastore cannot be unlocked.

15. verb
  • Enables/disables verbose messages.

16. quit
  • Quits the program.

7.4.3.3.2 Netopeer2 CLI datastore

Following are the netopeer2 CLI datastores:

• running
  – This is the base NETCONF configuration datastore holding the complete configuration currently active on the device. This datastore always exists.

• startup
  – The configuration datastore holding the configuration loaded by the device when it boots. This datastore is available only on servers that implement the :startup capability.

• candidate
  – The configuration datastore that can be manipulated without impacting the device's current configuration and that can be committed to the running configuration datastore. This datastore is available only on servers that implement :candidate capability.

• url:URI
  – Refers to a remote configuration datastore located at URI. The file that the URI refers to contains the configuration data hierarchy to be modified, encoded in XML under the element config in the urn:ietf:params:xml:ns:netconf:base:1.0 namespace. This datastore is available only on servers that implement the :url capability.

7.4.3.4 Sysrepod

Sysrepo deamon provides the functionality of the datastore on the system (executes the system-wide Sysrepo Engine). In normal circumstances, it gets automatically started when the system starts up. However, auto-start is not configured by cmake install operation and user should configure it manually, according to the guidelines of user's system.

Usage:

sysrepod [-h] [-v] [-d] [-l <level>]

Options:

• -h Prints usage help.
• -v Prints version.
• -d Debug mode - daemon runs in the foreground and print logs to stderr instead of syslog.
• -l <level> Sets verbosity level of logging:
  – 0 = all logging turned off
  – 1 = log only error messages
  – 2 = (default) log error and warning messages
  – 3 = log error, warning and informational messages
  – 4 = log everything, including development debug messages
### 7.4.3.5 Sysrepocfg

`sysrepocfg` is a command-line tool for editing, importing, and exporting configuration stored in Sysrepo datastore. It allows users to edit startup or running configuration of specified module in a preferred text editor. It also propagates the performed changes into the datastore transparently for all subscribed applications. Moreover, the user can export the current configuration into a file or get it printed to the standard output. Similarly, an already prepared configuration can be imported from a file or read from the standard input.

In the background, `sysrepocfg` uses Sysrepo client library for any data manipulation rather than directly accessing configuration data files. Thus, it effectively inherits all main features of Sysrepo, such as YANG-based data validation, full transaction and concurrency support. Most importantly, subscribed applications are notified about the changes made using `sysrepocfg` and can immediately take the new configuration into account.

### 7.4.3.6 Sysrepoctl

The `sysrepoctl` provides functions to manage modules. It can be configured using the options and commands described below.

**operation-operations**

- `--help`: Prints the generic description and a list of commands. The detailed description and list of arguments for the specific command are displayed by using `--help` argument of the command.
- `--install`: Installs specified schema into sysrepo (`--yang` or `--yin` must be specified).
- `--uninstall`: Uninstalls specified schema from sysrepo (`--module` must be specified).
- `--list`: Lists YANG modules installed in sysrepo (note that Conformance Installed implies also Implemented).
- `--change`: Changes specified module in sysrepo (`--module` must be specified).
- `--feature-enable`: Enables a feature within a module in sysrepo (feature name is the argument, `--module` must be specified).
- `--feature-disable`: Disables a feature within a module in sysrepo (feature name is the argument, `--module` must be specified).

**Other-options**

- `--yang`: Path to the file with schema in YANG format (`--install` operation).
- `--yin`: Path to the file with schema in YIN format (`--install` operation).
- `--module`: Name of the module to be operated on (`--change`, `--feature-enable`, `--feature-disable` operations, `--uninstall` several modules can be delimited with `,`).
- `--permissions`: Access permissions of the module’s data in chmod format (`--install`, `--change` operations).

**Examples**

- Installs a new module by specifying YANG file, ownership and access permissions:
  ```
  sysrepoctl --install --yang=/home/user/ietf-interfaces.yang --owner=admin:admin --permissions=644
  ```
- Changes the ownership and permissions of an existing YANG module:
  ```
  sysrepoctl --change --module=ietf-interfaces --owner=admin:admin --permissions=644
  ```
- Enables a feature within a YANG module:
  ```
  sysrepoctl --feature-enable=if-mib --module=ietf-interfaces
  ```
- Uninstalls 2 modules, second one is without revision:
  ```
  sysrepoctl --uninstall --module=mod-a,mod-b --revision=2035-05-05
  ```
7.4.3.7 List of yang models

Table 87. Revision of yang models

<table>
<thead>
<tr>
<th>YANG models name</th>
<th>Official revision</th>
<th>List of the developer changes</th>
<th>Draft or Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>ieee802-types</td>
<td>2020-10-23</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>ieee802-dot1q-types</td>
<td>2020-10-23</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>ietf/interfaces</td>
<td>2018-02-20</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>iana-if-type</td>
<td>2020-01-10</td>
<td>- Delete duplicate revision</td>
<td>PUBLISHED</td>
</tr>
<tr>
<td>ietf-yang-types</td>
<td>2013-07-15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ieee802-dot1q-bridge</td>
<td>2020-11-24</td>
<td>- Add prefix of bridge-type and bridge-component</td>
<td>DRAFT</td>
</tr>
<tr>
<td>ieee802-dot1q-sched</td>
<td>2020-07-07</td>
<td>- Add prefix of gate operation-name</td>
<td>DRAFT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Allow gate-control-entry to be empty</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Delete cycle-time limited</td>
<td></td>
</tr>
<tr>
<td>ieee802-dot1q-preemption</td>
<td>2020-07-07</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>ieee802-dot1cb-stream-identification-types</td>
<td>2021-06-14</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>ieee802-dot1cb-stream-identification</td>
<td>2021-05-06</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>ieee802-dot1q-stream-filters-gates</td>
<td>2020-11-06</td>
<td></td>
<td>PUBLISHED</td>
</tr>
<tr>
<td>ieee802-dot1q-psfp</td>
<td>2020-07-07</td>
<td>- Add prefix of set-gate-and-ipv</td>
<td>DRAFT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Delete cycle-time limited</td>
<td></td>
</tr>
<tr>
<td>ietf-ip</td>
<td>2018-02-22</td>
<td></td>
<td>RFC</td>
</tr>
<tr>
<td>ieee802-dot1q-pb</td>
<td>2020-11-24</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>ieee802-dot1q-qci-augment</td>
<td>2019-05-20</td>
<td>- Add by NXP</td>
<td>PROPRIETARY</td>
</tr>
<tr>
<td>ieee802-dot1cb-frer-types</td>
<td>2021-05-06</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>ieee802-dot1cb-frer</td>
<td>2021-05-06</td>
<td></td>
<td>DRAFT</td>
</tr>
<tr>
<td>nxp-bridge-vlan-tc-flower</td>
<td>2020-04-02</td>
<td>- Add by NXP</td>
<td>PROPRIETARY</td>
</tr>
</tbody>
</table>

7.4.3.8 Operation examples

The following figure describes the steps to configure device via netopeer2:
In *sysrepo-tsn*, there are some instance files to configure TSN features on LS1028ARDB board:

- **Instance files for TSN configuration**

Users can configure TSN functions of LS1028ARDB board using these instance files. Before starting, make sure that *sysrepo, sysrepo-plugind, sysrepo-tsn* and *netopeer2-server* are running on the board. Use the following steps to configure TSN feature on LS1028ARDB board.

1. Start *netopeer2-cli* on the computer with *netopeer2-cli* installed:
   ```
   $ netopeer2-cli
   ```

2. Connect to *netopeer2-server* on LS1028ARDB board (use the IP on LS1028ARDB, here 10.193.20.53 is example):
   ```
   > connect --login root --host 10.193.20.53
   ```
3. Get status data of server:
   ```
   > get
   ```

4. Get configuration data in running datastore:
   ```
   > get-config --source running
   ```

5. Configure QBV feature of LS1028ARDB with `qbv-eno0-enable.xml`
   ```
   > edit-config --target running --config=qbv-eno0-enable.xml
   ```

6. Check configuration data of QBV:
   ```
   > get-config --source running --filter-xpath /ietf-interfaces:interfaces/
     interface[name='eno0']/ieee802-dot1q-sched:gate-parameters
   ```

7. Copy configuration data in `running` datastore to `startup` datastore:
   ```
   > copy-config --source running --target startup
   ```

8. Disconnect with netopeer2-server:
   ```
   > disconnect
   ```

### 7.4.3.9 Application scenarios

**Note:** The related xml file in the following cases can be obtained from the link: [https://github.com/nxp-real-time-edge-sw/real-time-edge-sysrepo/blob/master/Instances](https://github.com/nxp-real-time-edge-sw/real-time-edge-sysrepo/blob/master/Instances).

**Note:** The interface name in the xml file should match with the actual interface name used on the board.

1. Prerequisites:
   a. Start netopeer2-cli on the computer with netopeer2-cli installed:
      ```
      $ netopeer2-cli
      ```
   b. Connect to the netopeer2-server using the command below:
      ```
      > connect --login root --host 10.193.20.53
      ```

2. Configure the IP address:
   a. Edit configuration file, change Ethernet interface name and IP:
      ```
      $ vim ietf-ip-cfg.xml
      ```
   b. Send the configuration file:
      ```
      > edit-config --target running --config=ietf-ip-cfg.xml
      ```

3. Configure the MAC address for the bridge:
   a. Create a bridge named `br1`:
      ```
      $ ip link add name br1 type bridge
      ```
   b. Edit the configuration file, change bridge name and MAC:
      ```
      $ vim ietf-mac-cfg.xml
      ```
c. Send the configuration file:

```
$ edit-config --target running --config=ietf-mac-cfg.xml
```

4. Add VLAN for Ethernet interface:
   a. Create bridge named "br1" if not existing:

```
$ ip link add name br1 type bridge
```

   b. Edit the configuration file to change the interface name and VLAN ID:

```
$ vim ietf-vlan-cfg.xml
```

   c. Send the configuration file:

```
> edit-config --target running --config=ietf-vlan-cfg.xml
```

5. Configure LS1028ARDB Qbv via tc.
   a. Edit the configuration file to change the interface name and VLAN ID:

```
$ vim qbv-swp0-enable.xml
```

   b. Send the configuration file:

```
> edit-config --target running --config=qbv-swp0-enable.xml
```

   c. Show the result.

```
# tc qdisc show dev swp0
```

*Note: If using tc or ethtool commands instead of libtsn, enable "real-time-edge-sysrepo-tc" in conf/distro/include/real-time-edge-base.inc as shown below:

```
REAL_TIME_EDGE_SYSREPO_ls1028ardb = "real-time-edge-sysrepo-tc"
```

Otherwise, disable "real-time-edge-sysrepo-tc":

```
REAL_TIME_EDGE_SYSREPO_ls1028ardb = ""
```

• For LS1028ARDB board, if real-time-edge-sysrepo-tc is enabled, you should set prerequisite for swpx (swp0 swp1 or swp2 ...) port using the following commands:

```
# tc qdisc add dev swpx ingress
# tc filter add dev swpx ingress chain 0 pref 49152 flower skip_sw action goto chain 10000
# tc filter add dev swpx ingress chain 10000 pref 49152 flower skip_sw # action goto chain 11000
# tc filter add dev swpx ingress chain 11000 pref 49152 flower skip_sw action goto chain 12000
# tc filter add dev swpx ingress chain 12000 pref 49152 flower skip_sw action goto chain 20000
# tc filter add dev swpx ingress chain 20000 pref 49152 flower skip_sw action goto chain 21000
# tc filter add dev swpx ingress chain 21000 pref 49152 flower skip_sw action goto chain 30000
```

6. Configure LS1028ARDB Qci via tc using the steps below.
   a. Create a bridge named "switch" if not existing:

```
# ip link add name switch type bridge
```

   b. Edit and send configuration file:

```
edit-config --target running --config=switch-qci-fm-gate-enable.xml
```
c. Show the result.

```
# tc filter show dev swp0 ingress
```

d. Disable the configuration.

```
> edit-config --target running --config=switch-qci-fm-gate-disable.xml
```

**Note:**
- The destination-address in instance file should be learned by switch.
- Users should send switch-qci-fm-gate-disable.xml after switch-qci-fm-gate-enable.xml

7. Configure LS1028ARDB Qbu via ethtool using the steps below.

**Note:** Disable cut through on the target board first by executing the command below:

```
# tsntool ctset --device swp0 --queue_stat 0x0
```

a. Edit and send configuration file:

```
> edit-config --target running --config=qbu-swp0.xml
```

b. Show the result:

```
# ethtool --show-frame-preemption swp0
```

8. Configure LS1028ARDB VLAN ID and priority filter via tc:

a. Edit configuration file, change the interface name and action_spec:

```
$ vim ietf-br-vlan-cfg.xml
```

b. Send the configuration file:

```
> edit-config --target running --config=ietf-br-vlan-cfg.xml
```

9. Configure i.MX 8DXL / i.MX 8M Plus / i.MX 93 Qbv via tc.

a. Edit and send configuration file:

```
> edit-config --target running --config=qbv-eth1-enable.xml
```

b. Display the result using the command below:

```
# tc qdisc show dev eth1
```

10. Configure i.MX 8DXL / i.MX 8M Plus / i.MX 93 Qbu via ethtool.

a. Edit and send configuration file:

```
> edit-config --target running --config=qbu-eth1.xml
```

b. Display the result using the command below:

```
# ethtool --show-frame-preemption eth1
```

7.4.4 Troubleshooting

1. Connection fails at client side:

```
nc ERROR: Remote host key changed, the connection will be terminated!
nc ERROR: Checking the host key failed.
cmd_connect: Connecting to the 10.193.20.4:830 as user "root" failed.
```

**Fixing:**
The reason is that the SSHD key changed at the server.
- First, users should get host list using the command knownhosts.
• Then, remove the related item. For example `knownhosts --del 19`.

2. Request could not be completed because the relevant data model content does not exist.

| type:     | application          |
| tag:      | data-missing         |
| severity: | error                |
| path:     | /ietf-interfaces:interfaces/interface[name='eno0']/ieee802-dot1q-sched:gate-parameters/admin-gate-states |
| message:  | Request could not be completed because the relevant data model content does not exist. |

**Fixing:**
The reason is that the configuration data in xpath does not exist in the datastore. Such as deleting a node that does not exist.

When encountering such an error, user can get configuration data in the board with `get-config` command, and check whether the operation type (add/delete/modify) of the node in the path is reasonable or not.

7.5 Graphics on LS1028A

This chapter is applicable to LS1028A. For i.MX 8M Plus and i.MX 8M Mini, refer to [i.MX Graphics User's Guide](#) for verification of the features.

7.5.1 GPU

The GPU consists of a 3D graphics core and a 2D graphics core.

3D graphics core features are the following:
- Supports 166 million triangles/sec
- Supports 1 Giga pixel/sec fill rate
- Supports 16 GFLOPs
- Supports OpenGL ES 1.1, 2.0, 3.0, 3.1
- Supports OpenCL 1.2
- Vulkan

2D graphics core features are:
- Supports multi-source composition
- Supports one-pass filter
- Supports tile format from 3D graphics core
- Supports tile format from VPU

**Step1: Software setting and configuration**

GPU is enabled by default when compiling the image for i.MX 8M Plus, i.MX 8M Mini, and LS1028A.

**Step 2: Hardware setup**
- For LS1028ARDB, connect the monitor and LS1028ARDB with DP cable.
- For i.MX 8M Mini LPDDR4 EVK, connect MIPI-DSI to HDMI module, then connect to the monitor.
- For i.MX 8M Plus LPDDR4 EVK, connect the monitor and i.MX 8M Plus LPDDR4 EVK using HDMI cable.

Insert the USB mouse into USB port in the keyboard.

**Step 3: Running GPU demo**
OpenCL demo (example A and B) is supported on LS1028ARDB and i.MX 8M Plus LPDDR4 EVK. i.MX 8M Mini LPDDR4 EVK do not support this feature.

**Note:** Weston is running by default, before doing below demo, need to exit weston by command “killall weston”.

### A. OpenCL information

```
root@ls1028ardb:~# cd /opt/viv_samples/cl11/UnitTest
root@ls1028ardb:/opt/viv_samples/cl11/UnitTest# ./clinfo
>>>>>>>> ./clinfo Starting...
Available platforms: 1
Platform ID: 0
  CL_PLATFORM_NAME:       Vivante OpenCL Platform
  CL_PLATFORM_PROFILE:    FULL_PROFILE
  CL_PLATFORM_VERSION:    OpenCL 1.2 V6.4.0.p2.234062
  CL_PLATFORM_VENDOR:     Vivante Corporation
  CL_PLATFORM_EXTENSIONS: cl_khr_icd
Available devices:      1
Device ID:     0
Device Ptr:    0xd04742f0
  CL_DEVICE_NAME: Vivante OpenCL Device GC7000UL.6202.0000
  CL_DEVICE_VENDOR: Vivante Corporation
  CL_DEVICE_TYPE:                                GPU
  CL_DEVICE_OPENCL_C_VERSION:                    OpenCL C 1.2
  CL_DEVICE_VENDOR_ID:                           0x00564956
  CL_DEVICE_PLATFORM:                            0x9e272728
  CL_DEVICE_VERSION:                             OpenCL 1.2
  CL_DRIVER_VERSION:                             OpenCL 1.2 V6.4.0.p2.234062
  CL_DEVICE_MAX_COMPUTE_UNITS:                   1
  CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:            3
    CL_DEVICE_MAX_WORK_ITEM_SIZES[0]:      512
    CL_DEVICE_MAX_WORK_ITEM_SIZES[1]:      512
    CL_DEVICE_MAX_WORK_ITEM_SIZES[2]:      512
  CL_DEVICE_MAX_WORK_GROUP_SIZE:                 512
  CL_DEVICE_MAX_CLOCK_FREQUENCY:                 650 MHz
  CL_DEVICE_IMAGE_SUPPORT:                       Yes
    CL_DEVICE_MAX_READ_IMAGE_ARGS:         128
    CL_DEVICE_MAX_WRITE_IMAGE_ARGS:        8
    CL_DEVICE_IMAGE2D_MAX_WIDTH:           8192
    CL_DEVICE_IMAGE2D_MAX_HEIGHT:          8192
    CLDEVICE_IMAGE3D_MAX_WIDTH:            8192
    CL_DEVICE_IMAGE3D_MAX_HEIGHT:          8192
    CL_DEVICE_IMAGE3D_MAX_DEPTH:           8192
  CL_DEVICE_MAX_SAMPLERS:                16
```

### B. Fourier transform based on GPU

```
root@ls1028ardb:~# cd /opt/viv_samples/cl11/fft/
root@ls1028ardb:/opt/viv_samples/cl11/fft# ./fft 16
Block size: 16
Print result: yes
Initializing device(s)...
Get the Device info and select Device...
# of Devices Available = 1
# of Compute Units = 1
# compute units = 1
Creating Command Queue...
```
\[
\log_2(\text{fft size}) = \log_2(16) = 4
\]
Compiling radix-2 FFT Program for GPU...
Creating radix-2 kernels...
Creating kernel fft_radix2 0 (p=1)...
Creating kernel fft_radix2 1 (p=2)...
Creating kernel fft_radix2 2 (p=4)...
Creating kernel fft_radix2 3 (p=8)...
Setting kernel args for kernel 0 (p=1)...
Setting kernel args for kernel 1 (p=2)...
Setting kernel args for kernel 2 (p=4)...
Setting kernel args for kernel 3 (p=8)...
running kernel 0 (p=1)...
running kernel 1 (p=2)...
running kernel 2 (p=4)...
running kernel 3 (p=8)...

Kernel execution time on GPU (kernel 0) : 0.000118 seconds
Kernel execution time on GPU (kernel 1) : 0.000122 seconds
Kernel execution time on GPU (kernel 2) : 0.000102 seconds
Kernel execution time on GPU (kernel 3) : 0.000076 seconds
Total Kernel execution time on GPU : 0.000418 seconds
Successful.

C. OpenGL ES demo

Note: Kill the Weston process before running kmscube.

kmscube is used to test OpenGL ES, as it supports HDMI and eDP interface.

For eDP interface, 4K resolution is not supported due to firmware limitation.

```bash
root@LS1028ARDB:~# kmscube
Using display 0x3107b6d0 with EGL version 1.5
===================================
EGL information:
  version: "1.5"
  vendor: "Vivante Corporation"
  client extensions: "EGL_EXT_client_extensions EGL_EXT_platform_base
  EGL_KHR_platform_wayland EGL_EXT_platform_wayland EGL_KHR_platform_gbm"
  display extensions: "EGL_KHR_fence_sync EGL_KHR_reusable_sync
  EGL_KHR_wait_sync EGL_KHR_image EGL_KHR_image_base EGL_KHR_image_pixmap
  EGL_KHR_gl_texture_2D_image EGL_KHR_gl_texture_cubemap_image
  EGL_KHR_gl_renderbuffer_image EGL_EXT_image dma_buf_import
  EGL_EXT_image_dma buf import modifiers EGL_KHR_Lock_surface
  EGL_KHR_create_context EGL_KHR_no_config_context EGL_KHR_surfaceless_context
  EGL_KHR_get_all_proc_addresses EGL_EXT_create_context_robustness
  EGL_EXT_protected_surface EGL_EXT_protected_content EGL_EXT_buffer_age
  EGL_ANDROID_native_fence_sync EGL_WL_bind_wayland_display
  EGL_WL_create_wayland_buffer_from_image EGL_KHR_partial_update
  EGL_EXT_swap_buffers_with_damage EGL_KHR_swap_buffers_with_damage"
===================================
OpenGL ES 2.x information:
  version: "OpenGL ES 3.1 V6.4.0.p2.234062"
  shading language version: "OpenGL ES GLSL ES 3.10"
  renderer: "Vivante Corporation"
  extensions: "GL_OES_vertex_type_10_10_10_2 GL_OES_vertex_half_float
  GL_OES_element_index_uint GL_OES_mapbuffer GL_OES_vertex_array_object
  GL_OES_compressed_ETC1_RGB8_texture GL_OES_compressed_paletted_texture
  GL_OES_texture_npot GL_OES_rgb8_rgba8 GL_OES_depth_texture
  GL_OES_depth_texture_cube_map GL_OES_depth24 GL_OES_depth32
  GL_OES_packed_depth_stencil GL_OES_fbo_render_mipmap GL_OES_get_program_binary
```
GL_OES_fragment_precision_high GL_OES_standard_derivatives GL_OES_EGL_image
GL_OES_EGL_sync GL_OES_texture_stencil8 GL_OES_shader_image_atomic
GL_OES_texture_storage_multisample_2d_array GL_OES_required_internalformat
GL_OES_surfaceless_context GL_OES_draw_buffers_indexed
GL_OES_texture_border_clamp GL_OES_texture_buffer GL_OES_texture_cube_map_array
GL_OES_draw_elements_base_vertex GL_OES_texture_half_float GL_OES_texture_float
GL_KHR_blend_equation_advanced GL_KHR_debug GL_KHR_robustness
GL_KHR_robust_buffer_access_behavior GL_EXT_texture_type_2_10_10_10_REV
GL_EXT_texture_compression_dxt1 GL_EXT_texture_format_BGRA8888
GL_EXT_texture_compression_s3tc GL_EXT_read_format_bgra
GL_EXT_multi_draw_arrays GL_EXT_frag_depth GL_EXT_discard_framebuffer
GL_EXT_blend_minmax GL_EXT_multisampled_render_to_texture
GL_EXT_color_buffer_half_float GL_EXT_color_buffer_float
GL_EXT_robustness GL_EXT_texture_sRGB_decode GL_EXT_draw_buffers_indexed
GL_EXT_texture_border_clamp GL_EXT_texture_buffer GL_EXT_texture_cube_map_array
GL_EXT_multi_draw_indirect GL_EXT_draw_elements_base_vertex GL_EXT_texture_rg
GL_EXT_protected_textures GL_EXT_sRGB GL_VIV_direct_texture

===================================
Rendered 120 frames in 2.000008 sec (59.999758 fps)
Rendered 241 frames in 4.016689 sec (59.999663 fps)
Rendered 361 frames in 6.016730 sec (59.999368 fps)

Below is the snapshot on screen.
7.5.2 Wayland and Weston

Weston is the reference implementation of a Wayland, this section describes how to enable Weston on NXP platforms. Weston is supported by LS1028ARDB, i.MX 8M Plus LPDDR4 EVK, and i.MX 8M Mini LPDDR4 EVK platforms.

1. **Software setting and configuration**
   It is enabled by default when compiling the image for i.MX 8M Plus, i.MX 8M Mini, and LS1028A platforms.

2. **Hardware setup**
   - For LS1028ARDB, connect the monitor and LS1028ARDB using DP cable.
   - For i.MX 8M Plus LPDDR4 EVK, connect the monitor and i.MX 8M Plus LPDDR4 EVK using HDMI cable.
   - For i.MX 8M Mini LPDDR4 EVK, connect MIPI-DSI to HDMI module, then connect to the monitor. Then, insert USB mouse and keyboard into USB port.

3. Run the lightweight desktop
   ```bash
   root@ls1028ardb:~# mkdir -p /run/user/0/
   root@ls1028ardb:~# export XDG_RUNTIME_DIR="/run/user/0/
   root@ls1028ardb:~# weston --tty=1
   # With parameter "-i" or "--idle-time" to set the time to enter idle state, default is 300s.
   # "0" means weston will not enter idle state.
   ```
root@ls1028ardb:~# weston --tty=1 -i 0 # or
root@ls1028ardb:~# weston --tty=1 --idle-time=0

Date: 2020-08-20 UTC
[14:38:00.002] weston 8.0.0
https://wayland.freedesktop.org
Bug reports to: https://gitlab.freedesktop.org/wayland/weston/issues/

[14:38:00.002] Command line: weston --tty=1
[14:38:00.002] OS: Linux, 5.4.3-rt1, #1 SMP PREEMPT_RT Tue Aug 18 14:49:14 CST 2020, aarch64
[14:38:00.002] Starting with no config file.
[14:38:00.005] Output repaint window is 16 ms maximum.
[14:38:00.007] Loading module '/usr/lib/libweston-8/drm-backend.so'
[14:38:00.050] initializing drm backend
[14:38:00.054] using /dev/dri/card0
[14:38:00.054] DRM: supports universal planes
[14:38:00.054] DRM: supports atomic modesetting
[14:38:00.054] DRM: supports picture aspect ratio
[14:38:00.056] Loading module '/usr/lib/libweston-8/gl-renderer.so'
[14:38:00.208] EGL client extensions: EGL_EXT_client_extensions
EGL_EXT_platform_base EGL_KHR_platform_wayland
EGL_EXT_platform_wayland EGL_KHR_platform_gbm
[14:38:00.224] EGL version: 1.5
[14:38:00.224] EGL vendor: Vivante Corporation
[14:38:00.224] EGL client APIs: OpenGL_ES OpenGL OpenVG
[14:38:00.224] EGL extensions: EGL_KHR_fence_sync EGL_KHR_reusable_sync
EGL_KHR_wait_sync EGL_KHR_image EGL_KHR_image_base
EGL_KHR_imagePixmap EGL_KHR_gl_texture_2D_image
EGL_KHR_gl_texture_cubemap_image EGL_KHR_gl_renderbuffer_image
EGL_EXT_image_dma_buf_import
EGL_EXT_image_dma_buf_import_modifiers EGL_KHR_lock_surface
EGL_KHR_create_context EGL_KHR_no_config_context
EGL_KHR_surfaceless_context EGL_KHR_get_all_proc_addresses
EGL_EXT_create_context_robustness EGL_EXT_protected_surface
EGL_KHR_surfaceless_context EGL_KHR_surfaceless_context available
EGL_KHR_surfaceless_context_available
[14:38:00.224] GL version: OpenGL ES 3.1 V6.4.0.p2.234062
GLSL version: OpenGL ES GLSL ES 3.10
[14:38:00.311] GL vendor: Vivante Corporation
[14:38:00.311] GL renderer: Vivante GC7000UL
[14:38:00.311] GL extensions: GL_OES_vertex_type_10_10_10_2
GL_OES_vertex_half_float GL_OES_element_index_uint
GL_OES_mapbuffer GL_OES_vertex_array_object
GL_OES_compressed_ETC1_RGB8_texture
GL_OES_compressed_paletted_texture GL_OES_texture_npot
GL_OES_rgb8_rgba8 GL_OES_depth_texture
GL_OES_depth_texture_cube_map GL_OES_depth24 GL_OES_depth32
GL_OES_packed_depth_stencil GL_OES_fbo_render_mipmap
GL_OES_get_program_binary GL_OES_fragment_precision_high
GL_OES_standard_derivatives GL_OES_EGL_image GL_OES_EGL_sync
GL_OES_texture_stencil18 GL_OES_shader_image_atomic
GL_OES_texture_storage_multisample_2d_array
GL_OES_required_internalformat GL_OES_surfaceless_context
GL_OES_draw_buffers_indexed GL_OES_texture_border_clamp
GL_OES_texture_buffer GL_OES_texture_cube_map_array
GL_OES_draw_elements_base_vertex GL_OES_texture_half_float
GL_OES_texture_float GL_KHR_blend_equation_advanced
GL_KHR_debug GL_KHR_robustness
GL_KHR_robust_buffer_access_behavior
GL_EXT_texture_type_2_10_10_10_REV
GL_EXT_texture_compression_dxt1 GL_EXT_texture_format_BGRA8888
GL_EXT_texture_compression_s3tc GL_EXT_read_format_bgra
GL_EXT_multi_draw_arrays GL_EXT_frag_depth
GL_EXT_discard_framebuffer GL_EXT_blend_minmax
GL_EXT_multisampled_render_to_texture
GL_EXT_color_buffer_half_float GL_EXT_color_buffer_float
GL_EXT_robustness GL_EXT_texture_sRGB_decode
GL_EXT_draw_buffers_indexed GL_EXT_texture_border_clamp
GL_EXT_texture_buffer GL_EXT_texture_cube_map_array
GL_EXT_multi_draw_indirect GL_EXT_draw_elements_base_vertex
GL_EXT_texture_rg GL_EXT_protected_textures GL_EXT_sRGB
GL_VIV_direct_texture

[14:38:00.311] GL ES 2 renderer features:
read-back format: BGRA
wl_shm sub-image to texture: yes
EGL Wayland extension: yes
[14:38:00.343] warning: no input devices on entering Weston. Possible causes:
- no permissions to read /dev/input/event*
- seats misconfigured (Weston backend option 'seat', udev device
  property ID Seat)
[14:38:00.343] failed to create input devices
[14:38:00.349] DRM: head 'DP-1' found, connector 56 is connected, EDID make
 'DEL', model 'P2417H', serial 'C9G5D7561ECB'
[14:38:00.349] Registered plugin API 'weston drm output api v1' of size 24
[14:38:00.357] Chosen EGL config details: id: 41 rgba: 8 8 8 0 buf: 24
dep: 0 stcl: 0 int: 1-60 type: win|pix|pbf|swap_preserved vis_id: XRGB8888
(0x34325258)
[14:38:00.357] Output DP-1 (crtc 48) video modes:
1920x1080@60.0, preferred, current, 148.5 MHz
1600x900@60.0, 108.0 MHz
1280x1024@75.0, 135.0 MHz
1280x1024@60.0, 108.0 MHz
1152x864@75.0, 108.0 MHz
1024x768@75.0, 78.8 MHz
1024x768@60.0, 65.0 MHz
800x600@75.0, 49.5 MHz
800x600@60.0, 40.0 MHz
640x480@75.0, 31.5 MHz
640x480@59.9, 25.2 MHz
720x400@70.1, 28.3 MHz
[14:38:00.357] Output 'DP-1' enabled with head(s) DP-1
[14:38:00.357] Compositor capabilities:
arbitrary surface rotation: yes
screen capture uses y-flip: yes
presentation clock: CLOCK_MONOTONIC, id 1
presentation clock resolution: 0.000000001 s
[14:38:00.359] Loading module '/usr/lib/weston/desktop-shell.so'
[14:38:00.367] launching '/usr/libexec/weston-keyboard'
[14:38:00.373] launching '/usr/libexec/weston-desktop-shell'
[14:39:23.341] event0 - Logitech USB Optical Mouse: is tagged by udev as:
Mouse
[14:39:23.341] event0 - Logitech USB Optical Mouse: device is a pointer
[14:39:23.342] associating input device event0 with output DP-1 (none by udev)
Below is the snapshot.

could not load cursor 'dnd-move'
could not load cursor 'dnd-copy'
could not load cursor 'dnd-none'
[14:39:51.794] event0  - Dell Dell USB Keyboard: is tagged by udev as:
  Keyboard
[14:39:51.794] event0  - Dell Dell USB Keyboard: device is a keyboard
[14:39:51.859] associating input device event0 with output DP-1 (none by udev)
[14:40:03.937] event0  - Dell Dell USB Keyboard: device removed
[14:40:11.758] event0  - Logitech USB Optical Mouse: is tagged by udev as:
  Mouse
[14:40:11.758] event0  - Logitech USB Optical Mouse: device is a pointer
[14:40:11.758] associating input device event0 with output DP-1 (none by udev)
[14:40:29.454] event0  - Dell Dell USB Keyboard: is tagged by udev as:
  Keyboard
[14:40:29.454] event0  - Dell Dell USB Keyboard: device is a keyboard
[14:40:29.454] libinput: configuring device "Dell Dell USB Keyboard".
[14:40:29.454] associating input device event0 with output DP-1 (none by udev)
[14:41:00.156] event0  - Dell Dell USB Keyboard: device removed
[14:42:29.287] event0  - Logitech USB Optical Mouse: is tagged by udev as:
  Mouse
[14:42:29.287] event0  - Logitech USB Optical Mouse: device is a pointer
[14:42:29.287] associating input device event0 with output DP-1 (none by udev)
[14:42:35.418] event1  - Dell Dell USB Keyboard: is tagged by udev as:
  Keyboard
[14:42:35.419] event1  - Dell Dell USB Keyboard: device is a keyboard
[14:42:35.419] libinput: configuring device "Dell Dell USB Keyboard".
[14:42:35.419] associating input device event1 with output DP-1 (none by udev)
7.5.3 CSI Camera

NXP provides i.MX 8M Mini LPDDR4 EVK board that have CSI MIPI and DSI MIPI interfaces. The gstreamer
video stream captures video frame from CSI camera and displays it to screen via DSI MIPI interface. The
element waylandsink is based on wayland library and Weston desktop. Refer to Section 7.5.2 to enable them.
Users should follow the steps below to enable gstreamer on a target board.

1. Software setting and configuration
   Gstreamer is enabled by default.

2. Hardware setup
   - For i.MX 8M Mini LPDDR4 EVK boards, DSI MIPI, and CSI MIPI modules are connected to the board.
   - MIPI-DSI to HDMI interface:
3. MIPI-CSI camera module: The following figure shows the MIPI-CSI camera module:

![MIPI-CSI camera module](image1)

**Figure 151. MIPI-CSI camera module**

4. Run `gstreamer` for camera.
   After entering Linux prompt, run `gstreamer` command as shown in the below steps:

   ```bash
   [root@imx8mmevk ~] # gst-launch-1.0 v4l2src device=/dev/video0 ! 'video/x-
raw,width=640,height=480,framerate=(fraction)30/1' ! videoconvert ! fbdevsink
   ```

   Below is the snapshot with `fbdevsink`:
5. Run `wayland` and `weston` for `waylandsink` using the commands shown below:

```
[root@imx8mmevk ~] # mkdir -p /run/user/0/
[root@imx8mmevk ~] # export XDG_RUNTIME_DIR="/run/user/0/
[root@imx8mmevk ~] # weston --tty=1 &
[root@imx8mmevk ~] # gst-launch-1.0 v4l2src device= /dev/video0 ! 'video/x-raw,width=640,height=480,framerate=(fraction)30/1' ! videoconvert ! waylandsink
```

The below snapshot shows `waylandsink` implementation:
7.5.4 OpenCV on LS1028ARDB

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

In Real-time Edge software v2.1 or later, OpenCV-4.5.2 and Python3 are enabled.

In order to get the FPS wanted during developing OpenCV application, below code is necessary to set the FPS.

```python
camera.set(cv2.CAP_PROP_FPS, 30)
```

7.6 Wireless on LS1028A

7.6.1 NFC

NFC click board is a mikroBUS add-on board with a versatile near field communications controller from NXP — the PN7120 NFC devices are used in contactless payment systems, electronic ticketing, smartcards. In retail and advertising, inexpensive NFC tags can be embedded into packaging labels, flyers, or posters.

This board is fully compliant with NFC Forum specifications. This implies that users can use the full potential of NFC and its three distinct operating modes listed below:

1. Card emulation
2. Read/Write
3. P2P

7.6.1.1 Introduction

The NXP’s PN7120 IC integrates an Arm™ Cortex-M0 MCU, which enables easier integration into designs, because it requires fewer resources from the host MCU. The integrated firmware provides all NFC protocols for performing the contactless communication in charge of the modulation, data processing, and error detection.

The board communicates with the target board MCU through the mikroBUS™ I2C interface, in compliance with NCI (NFC controller interface) 1.0 host protocols. RST and INT pins provide additional functionality. The board uses a 3.3 V power supply.

7.6.1.2 PN7120 features

PN7120 IC embeds a new generation RF contactless front-end, supporting various transmission modes according to NFCIP-1 and NFCIP-2, ISO/IEC14443, ISO/IEC 15693, ISO/IEC 18000-3, MIFARE, and FeliCa specifications. It embeds an Arm Cortex-M0 microcontroller core loaded with the integrated firmware supporting the NCI 1.0 host communication.

7.6.1.3 Hardware preparation

Use the following hardware items for the NFC clickboard demo setup:

1. LS1028ARDB
2. NFC click board
3. NFC sample card (tag)

Note: Users should insert the NFC click board into the LS1028ARDB mikroBUS1 slot.

7.6.1.4 Software preparation

In order to support NFC click board, use the following steps:

1. In Real-time Edge, libnfc-nci is enabled by default.
2. In Linux kernel configuration, make sure the below options are enabled:

   [*] Networking support
   <M> NFC subsystem support
   Near Field Communication (NFC) devices
   <M> NXP PN5XX based driver

   Note: The NXP PN5XX based driver only supports the Module mode.
3. Use the `make` command to create the images.

7.6.1.5 Testing the NFC click board

Use the following steps for testing the NFC Clickboard:

1. Install NFC driver module

   [root]# modprobe pn5xx_i2c.ko

2. The following log appears at the console after the above command is successful. The error information can be ignored in this case.
3. Run the `nfcDemoApp` application:

```
[root]# nfcDemoApp poll
```

4. Put the NFC Sample Card (tag) on top of the NFC click board:
Display of the above information indicates successful card reading.

7.6.2 Bluetooth Low Energy

This chapter introduces the features of the Bluetooth Low Energy P click board and how to use it on NXP’s LS1028A reference design board (RDB).

7.6.2.1 Introduction

The **BLE P Click** board carries the nRF8001 IC that allows user to add Bluetooth 4.0 to user's device. The click communicates with the target board MCU through mikroBUS SPI (CS, SCK, MISO, MOSI), RDY and ACT lines, and runs on 3.3 V power supply.

The **BLE P Click** board features a PCB trace antenna, designed for the 2400 MHz to 2483.5 MHz frequency band. The maximum device range is up to 40 meters in open space.

7.6.2.2 Bluetooth Low Energy

LS1028ARDB support Bluetooth Low Energy click board, Bluetooth Low Energy P click carries the nRF8001 IC that allows user to add Bluetooth 4.0 to the device.

7.6.2.3 Features

Following are the features provided by BLE P Click board:

- nRF8001 Bluetooth low energy RF transceiver
  - 16 MHz crystal oscillator
  - Ultra-low peak current consumption <14 mA
  - Low current for connection-oriented profiles, typically 2 μA
- PCB trace antenna (2400-2483.5 MHz, up to 40 meters)
- BLE Android app
- Interface: SPI (CS, SCK, MISO, MOSI), RDY, and ACT lines
- 3.3 V power supply

7.6.2.4 Hardware preparation

Use the following hardware items for the BLE P Click board demo setup:

1. LS1028ARDB
2. BLE P Click board
3. Android phone (option)

   The figure below depicts the hardware setup required for the demo:
7.6.2.5 Software preparation

Use these steps for the BLE P click board demo software setup:

- Download the JUMA UART (Android app) by using the link: https://apkpure.com/juma_uart/com.juma_UART

- Then, run the steps below in order to support BLE P click board:
  1. In Real-time Edge, libblep is enabled by default.
  2. In Linux kernel configuration, make sure the below options are enabled:

<table>
<thead>
<tr>
<th>Device Drivers</th>
<th>SPI support</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;*&gt; Freescale DSPI controller</td>
<td>&lt;*&gt; User mode SPI device driver support</td>
</tr>
</tbody>
</table>

- 3. Use the `make` command to create the images.

7.6.2.6 Testing the BLE P click board

Use the following steps for testing the BLE P click board:
1. **Running the blep_demo application:**
   The following log is displayed to indicate that the BLE P click board is initialized. After this, users can scan from their mobile phone or computer's Bluetooth device for the BLE P click board. The name of the BLE P click board used is "MikroE".

   ![Initialization log](image1)

2. **Connection log**
   Connect the BLE P click board via mobile app. On successful connection, the following log is displayed. Thereafter, the application can communicate with the BLE P click board.

   ![Connection log](image2)

3. **Disconnection log**
   Click the **Disconnect** button of the Android APP to disconnect from the BLE P click board. The following log displays that the disconnection is successful.

   ![Disconnection log](image3)

4. **Command line introduction**
   The **blep_demo** application supports four command lines: **devaddr**, **name=**, **version**, and **echo**.
   a. **devaddr**
      This command is used to obtain the MAC address of the BLE P click board. User can run this command at any time.

   ![devaddr command log](image4)
   b. **name=**
      This command is used to set the Bluetooth name of the BLE P click board while broadcasting. No spaces are required after the equal sign "="; and the content after the 'equal to' sign is the set name. The maximum length is 16 characters.
This command is used to send a string to the Android app. This command should be executed after the connection is established. The maximum length is 20 characters.

The below log displays the message displayed after user tries to send a string when no connection is established:

```
version
Please input a command!
Unknow event:0x00
Device version
Configuration ID:0x41
ACI protocol version:2
Current setup format:3
Setup ID:0x00
Configuration status:open(VM)
```

The below log is displayed when user sends a string after a connection is established:

```
echo hi
Please input a command!
Unknow event:0x00
ACI Lvt Pipe Error: Pipe #9
Pipe Error Code: 0xA3
Pipe Error Data: 0x00
Please connect the device before sending data
```

The below is displayed when user sends a string after a connection is established:

```
echo hello, world!
Please input a command!
Unknow event:0x00
The number of data command buffer is 1
```

5. Receiving data
When the Android app sends a string:

```
DataReceivedEvent: hi.yugxdr
```

7.6.3 BEE
This chapter introduces the features of the BEE Click Board and how to use it on LS1028ARDB.
7.6.3.1 BEE/ZigBEE

LS1028ARDB supports BEE click board, which can implement the MRF24J40MA 2.4 GHz IEEE 802.15.4 radio transceiver module from Microchip.

7.6.3.2 Introduction

Heterogeneous Multi-SoC Framework enables the usage of a combination of MPU and i.MX RT1180 as an Industrial Switch. This extends the MPU hardware capability with the i.MX RT1180 hardware capability, thereby providing switch functionality, TSN functionality, and the capability of supporting different Industrial Protocols. The i.MX RT1180 can be used to run real-time tasks such as industrial protocols in the real-time domain, whereas the MPU can process compute-heavy tasks, in the non-real-time domain.

The external Ethernet ports of i.MX RT1180 can be exposed to the MPU side as standard Ethernet interfaces as data path. Different interfaces such as SPI or I2C can be used as the management interface between MPU and i.MX RT1180.

7.6.3.3 Features

The features of the BEE Click Board are listed below:

- PCB antenna
- MRF24J40MA module
- Low current consumption (TX 23 mA, RX 19 mA, Sleep 2 μA)
- ZigBee stack
- MiWi™ stack
- SPI Interface
- 3.3 V power supply

7.6.3.4 Hardware preparation

Use the following hardware items for the BEE Click Board demo setup:

- Two LS1028ARDB boards
- Two BEE Click Boards

The Figure 164 describes the hardware setup for the BEE Click Board.
7.6.3.5 Software preparation

In order to support BEE click board, use the following steps:

1. In Real-time Edge, `libbee` is enabled by default.
2. In Linux kernel configuration, make sure the below options are enabled:

<table>
<thead>
<tr>
<th>Device Drivers</th>
<th>SPI support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freescale DSPI controller</td>
<td></td>
</tr>
<tr>
<td>User mode SPI device driver support</td>
<td></td>
</tr>
<tr>
<td>GPIO Support</td>
<td>sysfs interface</td>
</tr>
<tr>
<td>MPC512x/MPC8xxx/QorIQ GPIO support</td>
<td></td>
</tr>
</tbody>
</table>

3. Use the `make` command to create the images.

Note: The WA pin of BEE Click Board connects with the NC pin.
7.6.3.6 Testing the BEE click board

The test application bee_demo is created by using the BEE Click Board library. This application can transfer the file between two BEE Click Boards.

1. Users should create a file in any path. For example, ./samples/test.txt.
2. First, start a server node by running the command below:

   ```
   bee_demo -s -f=XXX
   ```

   The command parameters are as below:
   - **-s**: This device node acts as a server.
   - **-f=XXX**: This parameter is valid only on the server node. XXX is the file path (relative or absolute) to be transferred.

   ```
   [root]# ls
   samples
   [root]# bee_demo -s -f=./samples/test.txt
   spi mode: 0x0
   bits per word: 8
   max speed: 500000 Hz (500 KHz)
   BEE Click Board Demo.
   This node is a server node.
   Waiting for a client
   Reading the content of the file
   ```

3. Start a client node on another LS1028ARDB by running the command bee_demo -c. In the above command, the parameter -c implies that this device node acts as a client. After receiving the file, the client node automatically exits. The received file is saved in the current path.

   ```
   [root]# ls
   samples
   [root]# bee_demo -c
   spi mode: 0x0
   bits per word: 8
   max speed: 500000 Hz (500 KHz)
   BEE Click Board Demo.
   This node is a client node.
   Starting to get a file
   Send the SEQ_REQ command.
   Send the SEQ_START command.
   ```

   ```
   [root]# ls
   samples  test.txt
   [root]#
   ```

4. The following log indicates that the server node has finished sending a file.

   ```
   Send the SEQ_INFO command.
   Start to send the file
   It's completed to send a file.
   ```

7.7 SAI on LS1028ARDB

SAI on LS1028ARDB is enabled. Due to the pins are shared between IEEE 1588 and SAI, therefore the default setting is to enable IEEE 1588 and disable SAI.

Following below steps to enable SAI in Real-time Edge Linux.
1. Enable SAI support in Real-time Edge software

   $ cd yocto-real-time-edge/sources/meta-real-time-edge
   # Open file "conf/distro/include/real-time-edge-base.inc", add "sai" to
   "DISTRO_FEATURES:append:ls1028ardb" like this:
   DISTRO_FEATURES:append:ls1028ardb = " jailhouse real-time-edge-libbee real-
   time-edge-libblep libnfc-nci \
   wayland-protocols weston imx-gpu-viv libdrm kmscube \
   real-time-edge-sysrepo tsn-scripts wayland alsa sai"

2. Build the image

   $ cd yocto-real-time-edge
   $ DISTRO=nxp-real-time-edge MACHINE=ls1028ardb source real-time-edge-setup-
   env.sh -b build-1s1028ardb
   $ bitbake nxp-image-real-time-edge

3. Turn on "Lineout Playback Switch" and
   boot up LS1028ARDB with new image, enter Linux prompt:

   $ amixer -c 0 cset name='Lineout Playback Switch' on
   numid=11,iface=MIXER,name='Lineout Playback Switch'
   ; type=BOOLEAN,access=rw----,values=1
   : values=on

4. Run audio test (insert 3.5 mm headphone in AUDIO port on LS1028ARDB, some voice can be heard from
   left and right).

   $ speaker-test -c 2 -l 10 -t wav
   speaker-test 1.2.5.1
   Playback device is default
   Stream parameters are 48000Hz, S16_LE, 2 channels
   WAV file(s)
   Rate set to 48000Hz (requested 48000Hz)
   Buffer size range from 96 to 1048576
   Period size range from 32 to 349526
   Using max buffer size 1048576
   Periods = 4
   was set period_size = 262144
   was set buffer_size = 1048576
   0 - Front Left
   1 - Front Right
   Time per period = 7.964220
   0 - Front Left
   1 - Front Right
   Time per period = 2.976679
   0 - Front Left
   1 - Front Right
   ...

7.8 Wi-Fi on i.MX 8DXL EVK

7.8.1 Wi-Fi card information

The Wi-Fi card shipped with i.MX 8DXL EVK is called ‘1XL M.2 Module’ which is co-developed by Embedded Artists and Murata. The table below describes this Wi-Fi card.
The NXP 88W9098 wireless SoC is the industry's first Wi-Fi 6 solution based on the latest IEEE 802.11ax standard with an innovative Concurrent Dual Wi-Fi (CDW) architecture. CDW supports separate Wi-Fi networks simultaneously using both 2.4 GHz and 5 GHz frequency bands.

## 7.8.2 Hardware Setup

Install the Wi-Fi card ‘1XL M.2 Module’ to the M.2 connector (J17) on i.MX 8DXL EVK.

## 7.8.3 Software Enablement

The following instructions show how to bring up the Wi-Fi module on i.MX 8DXL EVK and connect to Access Point (AP) in Station Mode.

1. **Boot up Linux using DTB** `imx8dxl-evk-rpmsg.dtb`.

   ```
   => setenv fdt_file imx8dxl-evk-rpmsg.dtb
   => boot
   ```

2. **Load the Wi-Fi kernel driver module** `moal.ko`.

   The module parameters are in configuration file `/lib/firmware/nxp/wifi_mod_para.conf`. The NXP Wi-Fi SoCs require a firmware image to be loaded on power-up/reset. The firmware image for NXP 88W9098 with PCIe interface is `/lib/firmware/nxp/pcieuart9098_combo_v1.bin`.

   ```
   # modprobe moal mod_para=nxp/wifi_mod_para.conf
   ```

   • Verify the kernel debug messages printed in serial console. Notice the bold texts.

   ```
   [ 44.893129] wlan: Loading MWLAN driver
   [ 44.897479] wlan: Register to Bus Driver...
   [ 44.903192] wlan_pcie 0000:01:00.0: enabling device (0000 -> 0002)
   [ 44.909628] Attach moal handle ops, card interface type: 0x206
   [ 44.916925] PCIE9098: init module param from usr cfg
   [ 44.922056] card_type: PCIE9098, config block: 0
   [ 44.926746] cfg80211_wext=0xf
   [ 44.929821] max_vir_bss=1
   [ 44.932503] cal_data_cfg=none
   [ 44.935612] ps_mode = 1
   [ 44.938076] auto_ds = 1
   [ 44.940610] host_mlme=enable
   [ 44.947387] fw_name=nxp/pcieuart9098_combo_v1.bin
   ```
3. Verify the Wi-Fi interface.

```
# ifconfig -a
```

Notice the bold texts in the output.

```
# ifconfig -a
mlan0: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether 00:50:43:20:12:34  txqueuelen 1000  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
mmlan0: flags=4098<BROADCAST,MULTICAST> mtu 1500
    ether 00:50:43:20:52:56  txqueuelen 1000  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
```

RX errors 0  dropped 0  overruns 0  frame 0
TX packets 0  bytes 0 (0.0 B)
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

muap0:  
flags=4098<BROADCAST,MULTICAST>  mtu 1500
ether 00:50:43:20:53:56  txqueuelen 1000  (Ethernet)
RX packets 0  bytes 0 (0.0 B)
TX packets 0  bytes 0 (0.0 B)
TX errors 0  dropped 0  overruns 0  frame 0
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

mwfd0:  
flags=4098<BROADCAST,MULTICAST>  mtu 1500
ether 02:50:43:20:52:56  txqueuelen 1000  (Ethernet)
RX packets 0  bytes 0 (0.0 B)
TX packets 0  bytes 0 (0.0 B)
TX errors 0  dropped 0  overruns 0  frame 0
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

uap0:  
flags=4098<BROADCAST,MULTICAST>  mtu 1500
ether 00:50:43:20:13:34  txqueuelen 1000  (Ethernet)
RX packets 0  bytes 0 (0.0 B)
TX packets 0  bytes 0 (0.0 B)
TX errors 0  dropped 0  overruns 0  frame 0
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

wfd0:  
flags=4098<BROADCAST,MULTICAST>  mtu 1500
ether 02:50:43:20:12:34  txqueuelen 1000  (Ethernet)
RX packets 0  bytes 0 (0.0 B)
TX packets 0  bytes 0 (0.0 B)
TX errors 0  dropped 0  overruns 0  frame 0
TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

4. Enable Wi-Fi interface. Either `mlan0` or `mmlan0` can be used in the following steps.

    # ifconfig mlan0 up

5. Scan the visible Wi-Fi access points.

    # iw dev mlan0 scan

An sample output is below. Output for only one AP is shown for simplicity. Notice the bold text for the frequency and SSID.

    # iw dev mlan0 scan
    [ 88.895756] wlan: mlan0 START SCAN
    [ 90.941865] wlan: SCAN COMPLETED: scanned AP count=11

    BSS 00:fe:c8:d3:00:6e(on mlan0)
    freq: 5320
    beacon interval: 102 TUs
    capability: ESS Privacy SpectrumMgmt RadioMeasure (0x1111)
    signal: -73.00 dBm
    last seen: 4 ms ago
    SSID: NXPOPEN
    Supported rates: 18.0 24.0* 36.0 48.0 54.0
    TIM: DTIM Count 0 DTIM Period 1 Bitmap Control 0x0 Bitmap[0] 0x0
    Country: CN  Environment: Indoor/Outdoor
        Channels [36 - 48] @ 23 dBm
        Channels [52 - 64] @ 23 dBm
        Channels [149 - 165] @ 30 dBm
    BSS Load:
        * station count: 8
* channel utilisation: 34/255
* available admission capacity: 23437 [*32us]

Power constraint: 0 dB

HT capabilities:

Capabilities: 0x19ee
  HT20/HT40
  SM Power Save disabled
  RX HT20 SGI
  RX HT40 SGI
  TX STBC
  RX STBC 1-stream
  Max AMSDU length: 7935 bytes
  DSSS/CCK HT40

Maximum RX AMPDU length 65535 bytes (exponent: 0x003)
Minimum RX AMPDU time spacing: 8 usec (0x06)
HT RX MCS rate indexes supported: 0-23
HT TX MCS rate indexes are undefined

RSN:
  * Version: 1
  * Group cipher: CCMP
  * Pairwise ciphers: CCMP
  * Authentication suites: PSK
  * Capabilities: 4-PTKSA-RC 4-GTKSA-RC (0x0028)

HT operation:
  * primary channel: 64
  * secondary channel offset: below
  * STA channel width: any
  * RIFS: 0
  * HT protection: no
  * non-GF present: 1
  * OBSS non-GF present: 0
  * dual beacon: 0
  * dual CTS protection: 0
  * STBC beacon: 0
  * L-SIG TXOP Prot: 0
  * PCO active: 0
  * PCO phase: 0

RM enabled capabilities:

Capabilities: 0x73 0xc0 0x00 0x00 0x00
  Link Measurement
  Neighbor Report
  Beacon Passive Measurement
  Beacon Active Measurement
  Beacon Table Measurement
  Transmit Stream/Category Measurement
  Triggered Transmit Stream/Category
  Nonoperating Channel Max Measurement Duration: 0
  Measurement Pilot Capability: 0

Extended capabilities:
  * Proxy ARP Service
  * BSS Transition
  * DMS
  * QoS Map
  * WNM-Notification
  * Operating Mode Notification
  * Max Number Of MSDUs In A-MSDU is unlimited

VHT capabilities:

VHT Capabilities (0x0f8379b2):
  Max MFDU length: 11454
  Supported Channel Width: neither 160 nor 80+80
  RX LDPC
short GI (80 MHz)
TX STBC
SU Beamformer
SU Beamformee

VHT RX MCS set:
1 streams: MCS 0-9
2 streams: MCS 0-9
3 streams: MCS 0-9
4 streams: not supported
5 streams: not supported
6 streams: not supported
7 streams: not supported
8 streams: not supported

VHT RX highest supported: 0 Mbps

VHT TX MCS set:
1 streams: MCS 0-9
2 streams: MCS 0-9
3 streams: MCS 0-9
4 streams: not supported
5 streams: not supported
6 streams: not supported
7 streams: not supported
8 streams: not supported

VHT TX highest supported: 0 Mbps

VHT operation:
* channel width: 0 (20 or 40 MHz)
* center freq segment 1: 0
* center freq segment 2: 0
* VHT basic MCS set: 0xffc0

Transmit Power Envelope:
* Local Maximum Transmit Power For 20 MHz: 1 dBm
* Local Maximum Transmit Power For 40 MHz: 1 dBm

WMM:
* Parameter version 1
* u-APSD
* BE: CW 15-1023, AIFSN 3
* BK: CW 15-1023, AIFSN 7
* VI: CW 7-15, AIFSN 2, TXOP 3008 usec
* VO: CW 3-7, AIFSN 2, TXOP 1504 usec

DS Parameter set: channel 64

6. Check the current link status.

```
# iw dev mlan0 link
```

The output should be as below:

```
# iw dev mlan0 link
Not connected.
```

7. Configure the Wi-Fi network SSID and password in /etc/wpa_supplicant.conf. First delete lines from line 5 to end. Then run `wpa_passphrase <ssid> <password>` and append the output to /etc/wpa_supplicant.conf. This step only needs to be done once and is saved across reboots.

```
# sed -i '5,$ d' /etc/wpa_supplicant.conf
# wpa_passphrase <ssid> <password> >> /etc/wpa_supplicant.conf
```
A sample `/etc/wpa_supplicant.conf` file is shown below:

```bash
# cat /etc/wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
    ssid="NXPOPEN"
    psk=c5397a26ced00bcb3545de1e0b421f76c9b6ed2c72a36150b87d5951b755cf7c
}
```

8. Connect to the WLAN with the given SSID in `/etc/wpa_supplicant.conf`.

```bash
# wpa_supplicant -B -i mlan0 -c /etc/wpa_supplicant.conf -D nl80211
```

9. Check the link status again for the WLAN it is connected to.

```bash
# iw dev mlan0 link
```

A sample output is below:

```
# iw dev mlan0 link
Connected to fc:5b:39:5f:5d:ce (on mlan0)
    SSID: NXPOPEN
    freq: 5745
    RX: 52226 bytes (109 packets)
    TX: 2454 bytes (20 packets)
    signal: -75 dBm
    rx bitrate: 24.0 MBit/s
    tx bitrate: 6.5 MBit/s VHT-MCS 0 VHT-NSS 1
    bss flags: dtim period: 1
                beacon int: 102
```

10. Run DHCP client to get IP address.

```bash
# udhcpc -i mlan0
```

11. Check the IP address of AP and ping the AP to check connectivity.

```bash
# ip route
# ping <IP address of AP>
```

The sample output is shown below:

```
# ip route
default via 192.168.0.1 dev mlan0 metric 10
192.168.0.0/23 dev mlan0 proto kernel scope link src 192.168.0.158
# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255 time=5.39 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=255 time=5.37 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=255 time=6.13 ms
64 bytes from 192.168.0.1: icmp_seq=4 ttl=255 time=7.02 ms
......
```
12. Check the connectivity to the public website.

```
# echo nameserver "8.8.8.8" >> /etc/resolv.conf
# ping www.nxp.com
PING e6860.h.akamaiedge.net (23.7.170.207) 56(84) bytes of data.
64 bytes from a23-7-170-207.deploy.static.akamaitechnologies.com (23.7.170.207): icmp_seq=1 ttl=50 time=311 ms
64 bytes from a23-7-170-207.deploy.static.akamaitechnologies.com (23.7.170.207): icmp_seq=2 ttl=50 time=356 ms
64 bytes from a23-7-170-207.deploy.static.akamaitechnologies.com (23.7.170.207): icmp_seq=3 ttl=50 time=299 ms
64 bytes from a23-7-170-207.deploy.static.akamaitechnologies.com (23.7.170.207): icmp_seq=4 ttl=50 time=340 ms
64 bytes from a23-7-170-207.deploy.static.akamaitechnologies.com (23.7.170.207): icmp_seq=5 ttl=50 time=282 ms
.....
```

7.9 MODBUS

MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model, that provides client/server communication between devices connected on different types of buses or network.

7.9.1 Libmodbus introduction

Real-time Edge integrates libmodbus library. Libmodbus is a free software library used to send or receive data with a device that conforms to the Modbus protocol. It contains various backends to communicate over different networks (for example, serial in RTU mode or Ethernet in TCP IPv4/IPv6).

All the i.MX series and Layerscape series of boards support modbus. It can be used to write both:

- Client applications that reads/writes data from various devices.
- Server applications that provide data to several clients.

The official website that contains the latest version of the documentation for libmodbus is https://libmodbus.org/.

7.9.2 Modbus-Simulator introduction

Modbus-Simulator is a Modbus tool based on libmodbus library. And it contains a modbus client and a modbus device simulator.

Modbus-device-Simulator supports both TCP and RTU modes, and each mode supports the following functions.

**Features supported by TCP:**

- Gets CPU temperature of a device
- Gets the status of the LED light of a device
- Modifies the status of the LED light of a device

**Features supported by RTU:**

- Gets CPU temperature of a device
- Gets the status of the LED light of a device
- Modifies the status of the LED light of a device
- Modifies the slave address of the device
- Modifies the baud rate of the device
**Note:** Only the i.MX 8M Mini and i.MX 8M Plus boards support LED light function and CPU temperature function, other boards define a dummy variable. By default Modbus-Simulator is disable except i.MX 8M Mini and i.MX 8M Plus, taking i.MX 93 as an example, add the following line to `meta-real-time-edge/conf/distro/include/real-time-edge-base.inc` to enable it.

```
DISTRO_FEATURES:append:mx93-nxp-bsp = " modbus-simulator"
```

### 7.9.3 Modbus-Simulator usage

#### SYNOPSIS

```
{modbus_device_simulator|modbus_client_simulator} [OPTION]… {RTU-PARAMS|TCP-PARAMS}… {SERIALPORT|HOST}… [WRITE-DATA]…
```

### 7.9.3.1 Parameter description

The parameter of `[Option]` is the general parameter of TCP and RTU startup.

**Table 89. Parameter description**

<table>
<thead>
<tr>
<th>Parameter option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--debug,</td>
<td>show debug information.</td>
</tr>
<tr>
<td>-m,</td>
<td>connection Type TCP/RTU, optional parameter: {tcp</td>
</tr>
<tr>
<td>-a,</td>
<td>slave address.</td>
</tr>
<tr>
<td>-c,</td>
<td>read and write data number.</td>
</tr>
</tbody>
</table>
| -I,              | function codes, the following function codes are available. 
|                  | (0x01) Read Coils, (0x02) Read Discrete Inputs, (0x05) Write Single Coil 
|                  | (0x03) Read Holding Registers, (0x04) Read Input Registers, (0x06) Write Single Register |
| -r,              | register start address. |
| -o,              | response timeout(ms). |

The parameter of `RTU-PARAMS` is the general parameter of RTU startup.

**Table 90. Parameter description**

<table>
<thead>
<tr>
<th>Parameter option</th>
<th>Description</th>
</tr>
</thead>
</table>
| -b,              | baud rate, optional parameter: {4800|9600|19200|115200}. 
|                  | **NOTE**: when running the modbus_device_simulator, directly select the above parameters, when running the modbus_client_simulator, select the index of the parameters: {0|1|2|3}. |
| -d,              | data bits, optional parameter: {7|8}. |
| -s,              | stop bits, optional parameter: {1|2}. |
| -p,              | verify type, optional parameter: {none | even | odd}. |

The parameter of `TCP-PARAMS` is the general parameter of TCP startup.

**Table 91. Parameter description**

<table>
<thead>
<tr>
<th>Parameter option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p,</td>
<td>port.</td>
</tr>
</tbody>
</table>
7.9.3.2 Examples of TCP and RTU

Examples of TCP modbus_device_simulator and modbus_client_simulator startup are as follows:

1. Start modbus_device_simulator locally with port 1502.
   
   ```bash
   # modbus_device_simulator --debug -m tcp -p 1502 0.0.0.0
   ```

2. Start modbus_client_simulator and connect to the modbus_device_simulator with 127.0.0.1 and port 1502, function code is Write Single Coil. Function: Change the status of the LED light to 1.
   
   ```bash
   # modbus_client_simulator --debug -m tcp -t 0x05 -r 0 -p 1502 127.0.0.1 0xFF00
   ```

Examples of RTU modbus_device_simulator and modbus_client_simulator startup are as follows:

1. Start modbus_device_simulator serial connection, slave address is 1, baud rate is 115200, verify type is none, device is /dev/ttymxc2.
   
   ```bash
   # modbus_device_simulator --debug -m rtu -a 1 -b 115200 -p none /dev/ttymxc2
   ```

2. Start modbus_client_simulator serial connection, slave address is 1, function code is Write Single Register, register start address is 1, baud rate is 115200, verify type is none, device is /dev/ttymxc2, write date is 1. Function: Change the baud rate of /dev/ttymxc2 from 9600 to 115200.
   
   ```bash
   # modbus_client_simulator --debug -m rtu -a 1 -t 0x06 -r 1 -b 1 -p none /dev/ttymxc2
   ```

Note: For i.MX 6ULL, do the following:

J1704: pin7 - GND
J1703: pin1 - TX pin2 - RX

```bash
=== setenv fdtfile imx6ull-14x14-evk-lpuart.dtb
=== run bootcmd
```

Device name: /dev/ttymxc1

For i.MX 93, do the following:

J1001: pin25 - GND pin27 - TX pin28 - RX

```bash
=== setenv fdtfile imx93-11x11-evk-lpuart.dtb
=== run bootcmd
```

Device name: /dev/ttyLP4

7.9.3.3 Commands for all features

The commands for all features of TCP are as follows:

1. Read the status of the LED light:

   ```bash
   modbus_client_simulator --debug -m tcp -t 0x01 -r 0 -p 1502 127.0.0.1
   ```
2. Change the status of the LED light:
   ```bash
   # modbus_client_simulator --debug -m tcp -t 0x05 -r 0 -p 1502 127.0.0.1
   {0xFF00|0x0000}
   ```
3. Read the temperature of CPU
   ```bash
   # modbus_client_simulator --debug -m tcp -t 0x04 -r 0 -p 1502 127.0.0.1
   ```

The commands for all features of RTU are as follows:

1. Read the status of the LED light:
   ```bash
   modbus_client_simulator --debug -m rtu -a 1 -t 0x01 -r 0 -b 3 -p none /dev/ttymxc2
   ```
2. Change the status of the LED light:
   ```bash
   # modbus_client_simulator --debug -m rtu -a 1 -t 0x05 -r 0 -b 3 -p none /dev/ttymxc2 {0xFF00|0x0000}
   ```
3. Read the temperature of CPU:
   ```bash
   # modbus_client_simulator --debug -m rtu -a 1 -t 0x04 -r 0 -b 3 -p none /dev/ttymxc2
   ```
4. Change `modbus_device_simulator` slave address from 1 to 6.
   ```bash
   # modbus_client_simulator --debug -m rtu -a 1 -t 0x06 -r 0 -b 3 -p none /dev/ttymxc2 6
   ```
5. Change `modbus_device_simulator` baud rate from 115200 to 9600.
   ```bash
   # modbus_client_simulator --debug -m rtu -a 1 -t 0x06 -r 1 -b 3 -p none /dev/ttymxc2 1
   ```

7.9.4 Testing Modbus-Simulator

Use two i.MX8MP boards to test the functions of the modbus-simulator.
7.9.4.1 Testing TCP functions

Read the status of the LED light on the board1.

1. Enter board1 and start up `modbus_device_simulator`.
   
   ```bash
   # modbus_device_simulator --debug -m tcp -p 1502 0.0.0.0
   ```

2. Enter board2 and start up `modbus_client_simulator`.
   
   ```bash
   # modbus_client_simulator --debug -m tcp -t 0x01 -r 0 -p 1502 10.193.21.104
   ```

![Figure 165. Testing the functions of the modbus-simulator](image)

Note: Learn about the startup parameters before starting.

7.9.4.2 Testing RTU functions

**Preparation**: Connect the serial ports on the two boards, as shown below.

![Figure 167. Connection of two i.MX8MP boards](image)
The pin connection information of the two boards should be as follows:

- GND <---> GND
- TXD <---> RXD
- RXD <---> TXD

Read the temperature of CPU on the board1

1. Enter board1 and start up `modbus_device_simulator`.

   ```bash
   # modbus_device_simulator --debug -m rtu -a 1 -b 115200 -p none /dev/ttymxc2
   ```

2. Enter board2 and start up `modbus_client_simulator`.

   ```bash
   # modbus_client_simulator --debug -m rtu -a 1 -t 0x04 -r 0 -b 3 -p none /dev/ttymxc2
   ```

Note: The default number of stop bits is 1.

### 7.10 UART 9-bit Multidrop mode (RS-485) support

#### 7.10.1 Overview

The UART provides a 9-bit mode to facilitate multidrop (RS-485) network communication. When 9-bit RS-485 mode is enabled, UART transmitter can transmit the ninth bit (9th bit) set by TXB8. The UART receiver can differentiate between data frames (9th bit = 0) and address frames (9th bit = 1).

Two examples are provided to demo UART RS485 9-bit multidrop support:

- 9bit_iuart_interrupt_transfer for interrupt mode
- 9bit_iuart_polling for polling mode.

These two demos support i.MX 8M Mini LPDDR4 EVK board.

#### 7.10.2 Building and running the demo

##### 7.10.2.1 Building the demo

Refer to `RTEDGEYOCTOUG` to set up Yocto environment and build the `nxp-image-real-time-edge`. All demos are in the `/examples` directory of the rootfs.
The below command compiles the demo separately.

```
bitbake 9bit_uart-interrupt-transfer
```

The demo is located in the below directory:

```
tmp/deploy/images/imx8mm-lpddr4-evk/examples/
```

### 7.10.2.2 Hardware setup

In order to test this feature by using a single i.MX 8M Mini LPDDR4 EVK board, use external loopback of UART3 for testing. Refer to the figure below to connect PIN8 (UART3_TXD) and PIN10 (UART3_RXD) by using a flying wire.

![i.MX 8M Mini LPDDR4 EVK UART Connection](image)

Figure 168. i.MX 8M Mini LPDDR4 EVK UART Connection

### 7.10.2.3 Preparing the demo

1. Connect 12 V power supply, switch SW101 to power on the board.
2. Connect a USB cable between the host PC and the J901 USB port on the target board. Two UART connections appear on the PC, one is used for Linux console, and another is used for FreeRTOS console. Open two serial terminal on host PC with the following settings for each UART connection:
   - 115200 baud rate
   - 8 data bits
   - No parity
   - One stop bit
   - No flow control

### 7.10.2.4 Running the demo

Power up the board and start up the M-core firmware under U-Boot by using the commands listed in the below paragraph. Use `9bit_iuart_interrupt_transfer` or `9bit_iuart_polling` for different examples.
to replace “DEMO_NAME” in the following commands. Use 9-bit-iuart-interrupt-transfer or 9-bit-iuart-polling to replace "DEMO_DIRECTORY" in the following commands..

If you choose to run the binary in DRAM:

```bash
=> ext4load mmc 1:2 0x80000000 /examples/mcux-sdk/DEMO_NAME/ddr_release/DEMO_DIRECTORY.bin
=> dCache flush
=> bootaux 0x80000000
```

If you choose to run the binary in TCM:

```bash
=> ext4load mmc 1:2 0x48000000 /examples/mcux-sdk/DEMO_NAME/release/DEMO_DIRECTORY.bin
=> cp.b 0x48000000 0x7e0000 0x20000
=> bootaux 0x7e0000
```

The demo configures the UART used for testing in 9-bit multidrop mode and sets its slave address to be “0xfe”. Then it sends two pieces of data, the 9-bit of these data is zero, so they are figured out to be “data”. In general, need to send “address” with 9-bit "1" to the physical line of UART before sending “data” with 9-bit “0”. In order to do function verification, the demo does not send “address” before sending “data”, so UART cannot receive this data when it is looped back to itself. The second piece of data is received when it is looped back, because the demo sends “address” with 0xfe firstly before sending this “data”.

Below is a sample log displayed on the FreeRTOS console when the demo runs successfully:

```text
UART will send first piece of data out without addressing itself:
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17
UART will send second piece of data out with addressing itself:
Address: 0xfe : 0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87
RS-485 Slave Address has been detected.
UART received data:
Address: 0xfe : 0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87
All data matches!
```
# Revision history

Table 92 summarizes the revisions to this document.

## Table 92. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision number</th>
<th>Topic cross-reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18/12/2023</td>
<td>2.7</td>
<td>Added Section 5</td>
<td>Updated for Real Time Edge Software 2.7 release.</td>
</tr>
<tr>
<td>28/7/2023</td>
<td>2.6</td>
<td>-</td>
<td>Updated for Real Time Edge Software 2.6 release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.1.5.4</td>
<td>Added the Section.</td>
</tr>
<tr>
<td>30/3/2023</td>
<td>2.5</td>
<td>-</td>
<td>Updated for Real Time Edge Software 2.5 release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 4.6</td>
<td>Added the Section &quot;Heterogeneous Multicore VirtIO and networking sharing&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.1.5, Section 3.4.4.14</td>
<td>Added the Sections &quot;Codesys EtherCAT Master&quot;, &quot;Flextimer module&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 4.5.6</td>
<td>Added the Section &quot;Running the i.MX 93 demo&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Section 4.4.3</td>
<td>Modified the Sections:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Section 3.4.4.4</td>
<td>• RPMSG Cortex-A core and Cortex-M core</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>• GPIO file</td>
</tr>
<tr>
<td>16/12/2022</td>
<td>2.4</td>
<td>-</td>
<td>Added support for i.MX 8DXL and i.MX 93 EVK boards throughout the document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configuring Flow Meter policy on stream</td>
<td>Added the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 6.1.3.4.1</td>
<td>Added the Section for preemptive verify.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 3.4.4.15</td>
<td>Added the Math library support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.8</td>
<td>Added the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.1.3.3.5</td>
<td>Added the Section.</td>
</tr>
<tr>
<td>28/07/2022</td>
<td>2.3</td>
<td>What's new in Real-time Edge software v2.3</td>
<td>Updated the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Heterogeneous AMP Architecture</td>
<td>Added the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 6.2</td>
<td>Updated the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>Other updates throughout the document.</td>
</tr>
<tr>
<td>29/03/2022</td>
<td>2.2</td>
<td>What's new in Real-time Edge software v2.2</td>
<td>Added the Section. Updated for Real-time Edge Software Rev 2.2. Support for TLS protocol removed for NETCONF feature.</td>
</tr>
<tr>
<td>15/12/2021</td>
<td>2.1</td>
<td>Section 2.1.7</td>
<td>Added the Section. Updated for Real-time Edge Software Rev 2.1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 2.3</td>
<td>Updated the Section.</td>
</tr>
<tr>
<td>29/07/2021</td>
<td>2.0</td>
<td>-</td>
<td>• 'Real-time Edge Software' introduced instead of 'Open Industrial Linux', with real-time feature support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Rearranged the document structure to include the Chapters: Real-time System, Real-time Networking, and Protocols.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>Added support for building Real-time Edge image using Yocto Project build environment. Details are provided in the Real-time Edge Yocto Project User Guide.</td>
</tr>
<tr>
<td>Date</td>
<td>Revision number</td>
<td>Topic cross-reference</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>26/04/2021</td>
<td>1.11</td>
<td>Section 2.1</td>
<td>Added the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Integrated Baremetal framework in the document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 2.1</td>
<td>Added the Section that describes the new features of each release.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated the Section 'Getting Open IL'.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Deleted references to Edgescale, OP-TEE, OTA throughout the document and other minor updates.</td>
</tr>
<tr>
<td>22/12/2020</td>
<td>1.10</td>
<td>Section 6.2</td>
<td>Added the Chapter and related contents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added the 'Camera' Section and related details.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added the Host setup for i.MX 8M Plus EVK board details.</td>
</tr>
<tr>
<td>15/09/2020</td>
<td>1.9</td>
<td>Section 6.3</td>
<td>Added the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.5.1</td>
<td>Added the Chapter and related description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.5.2</td>
<td>Added the Chapter and related description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.1.3.3</td>
<td>Made it a part of the Chapter &quot;EtherCAT&quot;</td>
</tr>
<tr>
<td>29/05/2020</td>
<td>1.8</td>
<td>Section 3.3</td>
<td>Added the Section in Section 7.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated this Section Interface naming in Linux for LS1028 ARDB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated the Section Host system requirements for Open IL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated the Section Running Selinux demo.</td>
</tr>
<tr>
<td>20/02/20</td>
<td>1.7.1</td>
<td>Section 7.4.3.8</td>
<td>Updated this Section.</td>
</tr>
<tr>
<td>17/01/20</td>
<td>1.7</td>
<td>Section 7.1.3.3</td>
<td>Added the Chapter (nxp servo).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 6.3</td>
<td>Added the Chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Getting Open IL</td>
<td>Updated the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.4</td>
<td>Other updates.</td>
</tr>
<tr>
<td>31/08/19</td>
<td>1.6</td>
<td>Section 6.1.4</td>
<td>• Information related to pcpmap command removed from the Section Section 6.1.4.1 and Section 6.1.4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Port names &quot;eno/swp0&quot; changed to &quot;swp0&quot; for few tsntool commands.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Note added in Section Section &quot;Stream identification&quot; for usage of nulltagged and streamhandle parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added the Section Section 6.1.4.2.8.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Other minor updates</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated the table &quot;Host system mandatory packages&quot;, Added autogen autoconf libtool and pkg-config packages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.6.3</td>
<td>Added this Chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated Section 7.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.4</td>
<td>• Added the Section Section 7.4.3.1 and other updates.</td>
</tr>
<tr>
<td>01/05/19</td>
<td>1.5</td>
<td></td>
<td>Added the Section to describe interface naming for U-Boot and Linux for LS1028ARDB.</td>
</tr>
</tbody>
</table>

Table 92. Document revision history...continued
Table 92. Document revision history...continued

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision number</th>
<th>Topic cross-reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/02/2019</td>
<td>1.4</td>
<td>Section 1.3</td>
<td>Added support for LS1028ARDB (64-bit and Ubuntu). Updated various Sections accordingly</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Updated the OpenIL version and Git tag in the Section 'Getting Open IL'.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 6.1</td>
<td>Reorganized this Chapter and added separate Section for Section 6.1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.6.1</td>
<td>Added the Chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7.2</td>
<td>Minor updates in this Chapter. Also added the Section, Section 7.2.3.1 and Section 7.2.3.3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Added the Chapter QT.</td>
</tr>
<tr>
<td>15/10/2018</td>
<td>1.3.1</td>
<td>-</td>
<td>Updated the OpenIL version and Git tag in the Section 'Getting Open IL'.</td>
</tr>
<tr>
<td>31/08/2018</td>
<td>1.3</td>
<td>EtherCAT</td>
<td>Added the Chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlexCAN</td>
<td>Added the Chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i.MX6QSabreSD support.</td>
<td>Added the Section in Chapter 'NXP OpenIL platforms'. Updated other Sections for i.MX6Q Sabre support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Getting Open IL</td>
<td>Updated the Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Selinux demo</td>
<td>Added the Section for enabling SELinux and updated Basic setup. Updates in other Sections.</td>
</tr>
<tr>
<td>31/05/2018</td>
<td>1.2</td>
<td>-</td>
<td>Updated the Section, &quot;Hardware requirements&quot; for RTnet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>Updated the Section, &quot;Software requirements&quot; for RTnet.</td>
</tr>
<tr>
<td>18/04/2018</td>
<td>1.1.1</td>
<td>-</td>
<td>Added the Section, &quot;RTnet&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>Added a note for LS1043A switch setting</td>
</tr>
<tr>
<td>30/03/2018</td>
<td>1.1</td>
<td>-</td>
<td>Added support for industrial IoT Baremetal framework in this Section.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>Added a note for steps to be performed before booting up the board.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 1.4</td>
<td>Added the Section.</td>
</tr>
<tr>
<td>22/12/2017</td>
<td>1.0</td>
<td>Section 7.3</td>
<td>Added the Chapter.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 6.1</td>
<td>Chapters for &quot;1-board TSN demo&quot; and &quot;3-board TSN demo&quot; replaced by a single Chapter, &quot;TSN demo&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Section 7</td>
<td>• Updated the Section, 'Industrial Features'.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• -IEEE 1588 -'sja1105-ptp' support removed.</td>
</tr>
<tr>
<td>25/08/2017</td>
<td>0.3</td>
<td>-</td>
<td>Set up the OpenIL website (<a href="http://www.openil.org/%5C">http://www.openil.org/\</a>).</td>
</tr>
</tbody>
</table>
OTA - Xenomai Cobalt 64-bit and SJA1105 support added.

Qbv support added.

SELinux support for LS1043 / LS1046 Ubuntu Userland added.

OP-TEE support for LS1021ATSN platform added.

4G LTE module - 64-bit support for LS1043ARDB, LS1046 ARDB, and LS1012ARDB added.

Ubuntu Userland support for 64-bit LS1043ARDB and 64-bit LS1046ARDB added.

Initial public release.

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision number</th>
<th>Topic cross-reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>26/05/2017</td>
<td>0.2</td>
<td>-</td>
<td>Initial public release.</td>
</tr>
</tbody>
</table>
9 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 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.
 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 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.

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.

Amazon Web Services, AWS, the Powered by AWS logo, and FreeRTOS — are trademarks of Amazon.com, Inc. or its affiliates.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Real-time Edge software</td>
<td>2</td>
</tr>
<tr>
<td>Real-time Edge Software Yocto Project</td>
<td>2</td>
</tr>
<tr>
<td>Supported NXP platforms</td>
<td>2</td>
</tr>
<tr>
<td>Switch settings</td>
<td>3</td>
</tr>
<tr>
<td>Flashing pre-built images</td>
<td>3</td>
</tr>
<tr>
<td>Related documentation</td>
<td>4</td>
</tr>
<tr>
<td>Acronyms and abbreviations</td>
<td>5</td>
</tr>
<tr>
<td>2 Release notes</td>
<td>8</td>
</tr>
<tr>
<td>What's new</td>
<td>8</td>
</tr>
<tr>
<td>What's new in Real-time Edge software</td>
<td>8</td>
</tr>
<tr>
<td>What's new in Real-time Edge software v.2</td>
<td>8</td>
</tr>
<tr>
<td>What's new in Real-time Edge software v.2.1</td>
<td>12</td>
</tr>
<tr>
<td>What's new in Real-time Edge software v.2.1.8</td>
<td>12</td>
</tr>
<tr>
<td>What's new in Real-time Edge software v.2.1.3</td>
<td>12</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.11</td>
<td>14</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.11.1</td>
<td>14</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.14</td>
<td>15</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.15</td>
<td>15</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.5</td>
<td>16</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.6</td>
<td>16</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.7</td>
<td>17</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.14</td>
<td>17</td>
</tr>
<tr>
<td>What's new in OpenIL v.1.4</td>
<td>17</td>
</tr>
<tr>
<td>Feature support matrix</td>
<td>18</td>
</tr>
<tr>
<td>Open, fixed, and closed issues</td>
<td>21</td>
</tr>
<tr>
<td>3 Real-time system</td>
<td>22</td>
</tr>
<tr>
<td>Overview</td>
<td>22</td>
</tr>
<tr>
<td>Building, deploying, and releasing unified software</td>
<td>24</td>
</tr>
<tr>
<td>Yocto layer for Cortex-A core</td>
<td>25</td>
</tr>
<tr>
<td>Yocto layer for Cortex-M core</td>
<td>25</td>
</tr>
<tr>
<td>Preempt-RT Linux</td>
<td>29</td>
</tr>
<tr>
<td>System Real-time Latency tests</td>
<td>29</td>
</tr>
<tr>
<td>Real-time application development</td>
<td>30</td>
</tr>
<tr>
<td>Baremetal on Cortex-A core</td>
<td>30</td>
</tr>
<tr>
<td>Baremetal framework</td>
<td>31</td>
</tr>
<tr>
<td>Getting started</td>
<td>33</td>
</tr>
<tr>
<td>Running examples</td>
<td>39</td>
</tr>
<tr>
<td>Development based on Baremetal framework</td>
<td>40</td>
</tr>
<tr>
<td>Native RTOS on Cortex-A core</td>
<td>71</td>
</tr>
<tr>
<td>Overview</td>
<td>71</td>
</tr>
<tr>
<td>Building native RTOS on Cortex-A core</td>
<td>71</td>
</tr>
<tr>
<td>Booting native RTOS on Cortex-A core</td>
<td>71</td>
</tr>
<tr>
<td>Booting RTOS on Cortex-A core with Jailhouse</td>
<td>73</td>
</tr>
<tr>
<td>Jailhouse</td>
<td>73</td>
</tr>
<tr>
<td>Harpoon (RTOS on Cortex-A)</td>
<td>78</td>
</tr>
<tr>
<td>RTOS and Baremetal on Cortex-M core</td>
<td>79</td>
</tr>
<tr>
<td>Booting Cortex-M Core RTOS Image</td>
<td>79</td>
</tr>
<tr>
<td>Heterogeneous Multicore Framework</td>
<td>81</td>
</tr>
<tr>
<td>Overview</td>
<td>81</td>
</tr>
<tr>
<td>Building Heterogeneous Multicore RTOS Application</td>
<td>84</td>
</tr>
<tr>
<td>Build with Yocto</td>
<td>84</td>
</tr>
<tr>
<td>Build with Standalone Mode</td>
<td>84</td>
</tr>
<tr>
<td>Flexible Real-time System</td>
<td>86</td>
</tr>
<tr>
<td>Heterogeneous Multicore RAM Console</td>
<td>87</td>
</tr>
<tr>
<td>Heterogeneous Multicore hello_world</td>
<td>89</td>
</tr>
<tr>
<td>lwIP Networking Stack</td>
<td>97</td>
</tr>
<tr>
<td>RPMSG data communication</td>
<td>98</td>
</tr>
<tr>
<td>Overview</td>
<td>98</td>
</tr>
<tr>
<td>RPMSG performance evaluation</td>
<td>98</td>
</tr>
<tr>
<td>RPMSG between Cortex-A Core and Cortex-M Core</td>
<td>99</td>
</tr>
<tr>
<td>RPMSG between Cortex-A Core and Cortex-A Core</td>
<td>102</td>
</tr>
<tr>
<td>Complex RPMSG on MPU</td>
<td>108</td>
</tr>
<tr>
<td>RPMSG based resource sharing</td>
<td>115</td>
</tr>
<tr>
<td>Overview</td>
<td>115</td>
</tr>
<tr>
<td>Software architecture and design</td>
<td>115</td>
</tr>
<tr>
<td>Resource sharing based on SRTM</td>
<td>117</td>
</tr>
<tr>
<td>Building and running the demo on i.MX 8M Mini LPDDR4 EVK</td>
<td>120</td>
</tr>
<tr>
<td>Building and running the demo on i.MX 93 EVK</td>
<td>122</td>
</tr>
<tr>
<td>Building and running the demo on i.MX 93 QSB</td>
<td>124</td>
</tr>
<tr>
<td>Heterogeneous Multicore VirtIO and networking sharing</td>
<td>127</td>
</tr>
<tr>
<td>Heterogeneous Multicore VirtIO</td>
<td>127</td>
</tr>
<tr>
<td>Heterogeneous Multicore VirtIO performance evaluation</td>
<td>129</td>
</tr>
<tr>
<td>Heterogeneous Multicore VirtIO network sharing</td>
<td>129</td>
</tr>
<tr>
<td>Building Heterogeneous Multicore VirtIO backend firmware</td>
<td>130</td>
</tr>
<tr>
<td>Building Heterogeneous Multicore VirtIO frontend linux images</td>
<td>130</td>
</tr>
<tr>
<td>Running VirtIO performance testing on i.MX 8M Mini EVK</td>
<td>131</td>
</tr>
<tr>
<td>Running VirtIO network sharing</td>
<td>132</td>
</tr>
<tr>
<td>Unified Life Cycle management</td>
<td>135</td>
</tr>
<tr>
<td>Overview</td>
<td>135</td>
</tr>
<tr>
<td>Booting Native RTOS Cortex-A core image from U-Boot</td>
<td>136</td>
</tr>
<tr>
<td>Booting Native RTOS Cortex-M Core image from U-Boot</td>
<td>136</td>
</tr>
<tr>
<td>Using RemoteProc to boot RTOS Cortex-M Core Image</td>
<td>136</td>
</tr>
</tbody>
</table>

© 2023 NXP B.V. All rights reserved.

User guide
Rev. 2.7 — 18 December 2023

399 / 400
5 Heterogeneous Multi-SoC Framework ........ 138
5.1 Introduction ............................................ 138
5.2 Software architecture ............................... 138
5.2.1 Using one of the i.MX RT1180 switch ports as DSA CPU port ............. 139
5.2.2 Using i.MX RT1180 ENETC port as DSA CPU port ...................... 140
5.3 Running the Heterogeneous Multi-SoC Framework ......................... 140
5.3.1 Building and running on MPU + i.MX RT1180 EVK .................... 140
6 Real-time networking ......................... 150
6.1 Time Sensitive Networking (TSN) on NXP platforms ...................... 150
6.1.1 TSN hardware capability .......................... 150
6.1.2 TSN configuration .................................. 150
6.1.3 TSN on i.MX 8DXL / i.MX 8M Plus / i.MX 93 : 172
6.1.4 TSN on LS1028A .................................... 180
6.2 GenAVB/TSN stack .................................. 212
6.2.1 Introduction ........................................ 212
6.2.2 GenAVB/TSN stack start/stop .................... 215
6.2.3 Use cases description ............................ 216
6.2.4 Configuration files ................................. 238
6.2.5 Log files ............................................ 244
6.3 IEEE 1588/802.1AS .................................. 250
6.3.1 Introduction ...................................... 250
6.3.2 IEEE 1588 device types ......................... 250
6.3.3 IEEE 802.1AS time-aware systems .......... 251
6.3.4 Software stacks .................................. 251
6.3.5 Quick Start for IEEE 1588 ...................... 252
6.3.6 Quick Start for IEEE 802.1AS ................. 256
6.3.7 Long term test .................................... 257
6.3.8 Known issues and limitations ................... 257
6.4 Networking ............................................ 257
6.4.1 Q-in-Q on LS1028A Felix switch .............. 257
6.4.2 VCAP on LS1028A Felix switch ............... 259
7 Protocols .............................................. 263
7.1 EtherCAT master .................................. 263
7.1.1 Introduction .................................... 263
7.1.2 EtherCAT protocol ............................... 263
7.1.3 IGH EtherCAT architecture .................... 264
7.1.4 SOEM EtherCAT Master ......................... 285
7.1.5 CODESYS EtherCAT Master .................... 295
7.2 FlexCAN and CANopen ............................. 312
7.2.1 Introduction .................................... 312
7.2.2 Introducing the function of CAN example code ....................... 316
7.2.3 Running a CAN application ..................... 317
7.3 OPC UA ............................................. 319
7.3.1 OPC introduction ................................ 319
7.3.2 The node model .................................. 320
7.3.3 Node Namespaces ................................ 321
7.3.4 Node classes .................................... 321
7.3.5 Node graph and references .................... 322
7.3.6 Open62541 ....................................... 323
7.3.7 OPC UA Pub/Sub over TSN ..................... 324
7.3.8 OPC UA client installation and usage .......... 337
7.4 NETCONF/YANG .................................. 342
7.4.1 Overview ....................................... 342
7.4.2 Netopeer2 ....................................... 343
7.4.3 Configuration ................................... 345
7.4.4 Troubleshooting ................................. 355
7.5 Graphics on LS1028A ............................. 356
7.5.1 GPU ............................................. 356
7.5.2 Wayland and Weston ......................... 360
7.5.3 CSI Camera ..................................... 364
7.5.4 OpenCV on LS1028ARDB ...................... 367
7.6 Wireless on LS1028A ............................. 367
7.6.1 NFC ............................................ 367
7.6.2 Bluetooth Low Energy ......................... 370
7.6.3 BEE ............................................ 373
7.6.4 SAi on LS1028ARDB ......................... 376
7.6.5 Wi-Fi on i.MX 8DXL EVK ...................... 377
7.6.6 Wi-Fi card information ......................... 377
7.6.7 Hardware Setup ................................ 378
7.6.8 Software Enablement ......................... 378
7.6.9 MODBUS ........................................ 384
7.6.10 Libmodbus introduction ....................... 384
7.6.11 Modbus-Simulator introduction .......... 384
7.6.12 Modbus-Simulator usage ..................... 385
7.6.13 Testing Modbus-Simulator .................... 387
7.6.14 UART 9-bit Multidrop mode (RS-485) ........ 387
8 Revision history ...................................... 392
9 Note about the source code in the document ................................. 396
9.1 Legal information ................................... 397

For more information, please visit: https://www.nxp.com

© 2023 NXP B.V.

All rights reserved.

Document identifier: REALTIMEEDGEUG

Date of release: 18 December 2023