REALTIMEEDGEUG

Real-time Edge Software User Guide

Rev. 2.4 — 16 December 2022 User guide

Document information

Information	Content
Keywords	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.MX8DXL, I.MX93 EVK NXP hardware platforms
Abstract	This document describes the features and implementation of Real-time Edge Software on NXP hardware platforms. The key technology components include Real-time Networking, Real-time System, and Protocols.

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 Networking, Real-time System, and Protocols.

- The Real-time Networking includes TSN technology, TSN standards, management, configuration, and applications. Networking and redundancy features are also supported.
- The Real-time System includes PREEMPT_RT Linux, Jailhouse, U-Boot based BareMetal framework, and different combinations of these systems.
- The Protocols component includes support for industry standard protocols such as EtherCAT, CoE, 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 following table lists the NXP hardware SoCs and boards that support the Real-time Edge software.

Table 1. Supported NXP platforms

Platform	Architecture	Boot
i.MX 6ULL EVK	Arm v7	SD
i.MX 8DXL LPDDR4 EVK	Arm v8	SD
i.MX 8M Mini LPDDR4 EVK	Arm v8	SD
i.MX 8M Plus LPDDR4 EVK	Arm v8	SD
i.MX 93 EVK	Arm v8	SD
LS1028ARDB	Arm v8	SD
LS1021AIOT	Arm v7	SD
LS1021ATSN	Arm v7	SD
LS1021ATWR	Arm v7	SD
LS1012ARDB	Arm v8	QSPI
LS1043ARDB	Arm v8	SD
LS1046ARDB	Arm v8	SD
LS1046AFRWY	Arm v8	SD
LX2160ARDB Rev2	Arm v8	SD

1.3.1 Switch setting

The following table lists and describes the switch configuration for the platforms supported by Real-Time Edge software.

Table 2. Switch setting for various NXP platforms

Platform	Boot source	Switch setting
i.MX 6ULL EVK	Internal Boot / MicroSD	SW602 = 0b'10 (internal boot) and SW601[1:4] = 0b'0010 (MicroSD)
i.MX 8DXL LPDDR4 EVK	SD	SW1[1:4] = 0b'1100
i.MX 8M Mini LPDDR4 EVK	MicroSD / uSDHC2	• SW1101[1:10] = 0b' 0110110010 • SW1102[1:10] = 0b' 0001101000
i.MX 8M Plus LPDDR4 EVK	MicroSD / SDHC2	SW4[1:4] = 0b'0011
i.MX 93 EVK	MicroSD / uSDHC2	SW1301[1:4] = 0b'0100
LS1028ARDB	SD	SW2[1:8] = 0b'10001000
LS1021AIOT	SD	SW2[1] = 0b'0
LS1021ATSN	SD	SW2[1:6] = 0b'111111
LS1021ATWR	SD	IFC enabled: • SW2[1:8] = 0b'00101000 • SW3[1:8] = 0b'01100001
		QSPI enabled: • SW2[1:8] = 0b'00100000 • SW3[1:8] = 0b'01100001
LS1012ARDB	QSPI Flash 1	• SW1[1:8] = 0b'10100110 • SW2[1:8] = 0b'00000000
	QSPI Flash 2	• SW1[1:8] = 0b'10100110 • SW2[1:8] = 0b'00000010
LS1043ARDB	SD	SW4[1:8] + SW5[1] = 0b'00100000_0
		UART1 output select • SW3[3] = 0b'0: RJ45 • SW3[3] = 0b'1: CMSIS-DAP (MiniUSB)
LS1046ARDB	SD	SW5[1:8] + SW4[1] = 0b'00100000_0
		UART1 output select • SW4[4] = 0b'0: RJ45 • SW4[4] = 0b'1: CMSIS-DAP (MicroUSB)
LS1046AFRWY	SD	SW1[1:10] = 0b'0010000000
LX2160ARDB Rev2	SD	SW1[1:8] = 0b'10001000

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 (LS1028ARDB-PA as the example for following), then run below command to extract it:

```
$ unzip Real-time_Edge_v2.4_LS1028ARDB.zip
$ cd Real-time_Edge_v2.4_LS1028ARDB/real-time-edge
$ ls
atf                                 fsl-ls1028a-rdb-jailhouse-
without-enetc.dtb
dp                                  Image-ls1028ardb.bin
fsl-ls1028a-rdb-dpdk.dtb            nxp-image-real-time-edge-
ls1028ardb.manifest
fsl-ls1028a-rdb-dsa-swp5-eno3.dtb   nxp-image-real-time-edge-
ls1028ardb.rootfs.tar.bz2
fsl-ls1028a-rdb.dtb                 nxp-image-real-time-edge-
ls1028ardb.wic.zst
fsl-ls1028a-rdb-jailhouse.dtb       rcw
$ zstd -d nxp-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:

```
$ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wic of=/dev/
sdc
```

After flashing this image to SD card, insert this SD card into LS1028ARDB board, connect UART1 port and open it. Then, when you power on LS1028ARDB board, the below message would be displayed:

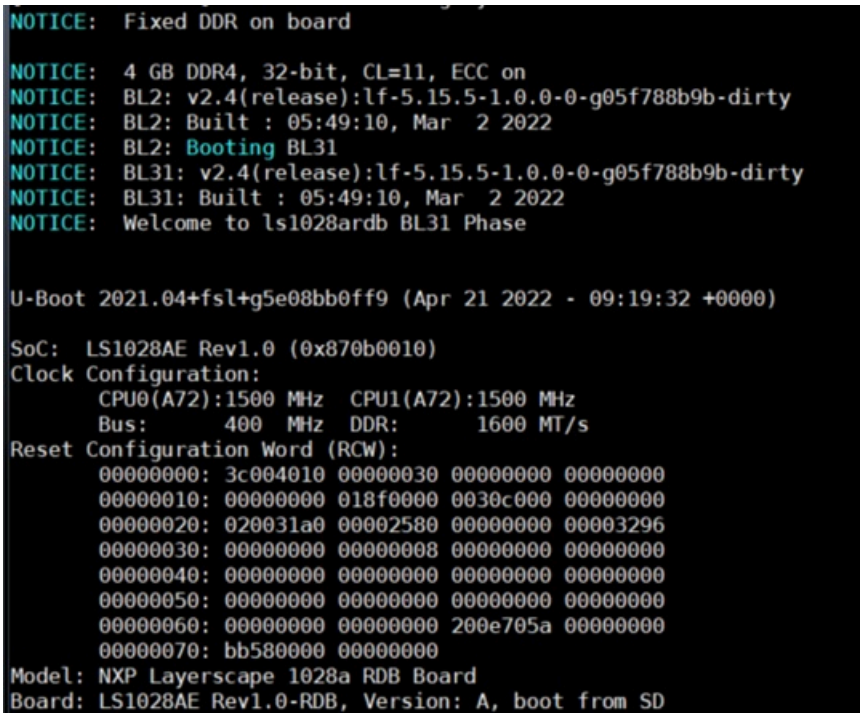


Figure 1. LS1028ARDB boot log

1.4 Related documentation

For all documentation related to Real Time Edge, refer to 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*)

Refer to the following guides for detailed instructions on booting up and setting up the relevant boards.

- *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*
- *LS1028ARDB Quick Start Guide*
- *LS1021AIoT Getting Started Guide*
- *LS1021ATSN Getting Started Guide*
- *LS1021ATWR Getting Started Guide*
- *LS1012ARDB Getting Started Guide*
- *LS1043ARDB Getting Started Guide*
- *LS1046ARDB Getting Started Guide*
- *LS1046AFRWY Getting Started Guide*
- *LX2160A/LX2160A-Rev2 RDB Quick Start Guide*

1.5 Acronyms and abbreviations

The following table lists the acronyms used in this document.

Table 3. Acronyms and abbreviations

Term	Description
AVB	Audio video bridging
AMP	Asymmetric multiprocessing
BC	Boundary clock
BLE	Bluetooth low energy
BMC	Best master clock
CA	Client application
CAN	Controller area network
CBS	Credit-based shaper
CDW	Concurrent Dual Wi-Fi
CMLDS	Common Mean Link Delay Service
DoS	Daniel-of-Service
DEI	Drop eligibility indication
DP	Display port

Table 3. Acronyms and abbreviations...*continued*

Term	Description
EtherCAT	Ethernet for control automation technology
ECU	Electronic control units
FDB	Forwarding database
FQTSS	Forwarding and queuing enhancements for time-sensitive streams
FMan	Frame manager
GPU	General processor unit
ICMP	Internet control message protocol
IEEE	Institute of electrical and electronics engineers
IETF	Internet engineering task force
IPC	Inter-processor communication
KM	Key management
LBT	Latency and bandwidth tester
MAC	Medium access control
MU	MU
NFC	Near field communication
NCI	NFC controller interface
NMT	Network management
OC	Ordinary clock
OpenIL	Open industry Linux
OPC	Open platform communications
OP-TEE	Open portable trusted execution environment
OS	Operating system
OTA	Over-the-air
OTPMK	One-time programmable master key
PCP	Priority code point
PDO	Process data object
PHC	PTP hardware clock
PIT	Packet inter-arrival times
PLC	programmable logic controller
PTP	Precision time protocol
QSPI	Queued serial peripheral interface
RCW	Reset configuration word
REE	Rich execution environment
RPC	Remote procedure call
RPMSG	Remote processor messaging
RTEdge	Real-time edge

Table 3. Acronyms and abbreviations...*continued*

Term	Description
RTC	Real-time clock
RTT	Round-trip times
RX	Receiver
SABRE	Smart application blueprint for rapid engineering
SDO	Service data object
SPI	Serial periphery interface
SRP	Stream reservation protocol
SRTM	Simplified Real-time Messaging
SRK	Single root key
TA	Trusted application
TAS	Time-aware scheduler
TC	Traffic classification
TCP	Transmission control protocol
TEE	Trusted execution environment
TFTP	Trivial file transfer protocol
TSN	Time sensitive networking
TX	Transmitter
TZASC	Trust zone address space controller
UDP	User datagram protocol
VLAN	Virtual local area network

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 v2.4

- **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.MX8M Plus and i.MX8M 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.MX8M Mini and i.MX8M Plus to the Cortex-M from Linux
  - Baremetal extensions on LS1046A
    - Single hardware interrupt routed to multiple cores
    - Newlib math library
  - Integration of Harpoon 2.2.0
- **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.MX8MP Plus
    - HTML5/chromium
  - Modbus
    - Libmodbus package integration
    - Modbus-simulator client and server
  - WIFI enabled on i.MX8DXL
- **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 lf-5.15.52-2.1.0**
  - LTS 5.15.52
  - Yocto Kirkstone 4.0
  - U-boot v2022.04

2.1.2 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 2.1)
    - Zephyr integration on i.MX 8M Plus and i.MX 8M Mini
    - Audio Application
      - sine wave playback
      - playback and recording (loopback)
      - audio pipeline
    - Industrial application:
      - AVB/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
      - RTOS on Cortex-M on i.MX 8M Mini
      - FreeRTOS or without an operating system
- Benchmark
  - Scheduling latency on Preempt_RT and Harpoon RTOS
  - Inter-core communication bandwidth of BareMetal
  - Packet processing time of TSN
  - Packet processing time of EtherCAT
- Based on lf-5.15.5-1.0.0
  - Linux 5.15.5-rt22
  - U-Boot v2021.04
  - Yocto Honister 3.4

2.1.3 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

2.1.4 What's new in Real-time Edge software v2.1

What’s New:

- 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

2.1.5 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

2.1.6 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

2.1.7 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

2.1.8 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

2.1.9 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

2.1.10 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

2.1.11 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

2.1.12 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 Qbv protocol
- Industrial protocols
  - LS1028A - BLE click board

2.1.13 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
    - LS1021AIoT, LS1043ARDB, LS1046ARDB
    - LS1028 BareMetal basic BareMetal support
- Industrial protocols
- – LS1028A - NFC click board
  - QT5.11
- OpenIL framework
  - boards: LS1028ARDB

2.2 Feature Support Matrix

The following tables show the features that are supported in this release.

Table 4. Key features

Feature				i.MX 6ULL 14x14 EVK	i.MX 8DXL LPDDR4 EVK	i.MX 8M Mini LPDDR4 EVK	i.MX 8M Plus LPDDR4 EVK	i.MX 93 EVK	LS1028A RDB	LS1021A TSN	LS1021A IOT	LS1021A TWR	LS10-12 ARDB	LS10-43 ARDB	LS10-46 ARDB	LS10-46 AFRWY	LX2160A RDB
Boot mode	SD			Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y
	QSPI												Y				
Real-Time System	Preempt-RT Linux			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	BareMetal	ICC				Y	Y		Y		Y			Y	Y		Y
		PCIe							Y		Y			Y	Y		
		Ethernet				Y	Y		Y					Y	Y		
		GPIO				Y	Y				Y						
		IPI				Y	Y		Y		Y			Y	Y		Y
		UART				Y	Y		Y		Y			Y	Y		Y
		USB									Y			Y	Y		
		SAI							Y								
		CAN									Y						
		I2C							Y		Y			Y	Y		
		QSPI													Y		
		IFC													Y		
		Linux (communi-cation with BareMetal)	ICC			Y	Y		Y		Y			Y	Y		Y
			IPI			Y	Y		Y		Y			Y	Y		Y
		Single HW Interrupt to multiple cores													Y		
		Newlib Math library													Y		
	Heteroge n-eous Multicore	UART Sharing				Y											
		RPMSG between A-Cores				Y											
		Linux booting M-Core Image				Y	Y										
	Jailhouse					Y	Y		Y					Y	Y		
	Harpoon (RTOS on Cortex-A)	FreeRTOS				Y	Y										
		Zephyr				Y	Y										
Real Time Networking	TSN Standards	Qbv			Y		Y	Y	Y								
		Qbu			Y		Y	Y	Y								
		Qci			N/A		N/A	N/A	Y								
		Qav			Y	Y	Y	Y	Y								
		802.1AS			Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y
		802.1CB			N/A		N/A	N/A	Y								
		VCAP chain mode			N/A		N/A	N/A	Y								
		802.1 Q-in-Q							Y								
	TSN Configurations	Linux tc command			Y		Y	Y	Y								
		TSN tool							Y								
		NETCONF/ YANG	Qbv		Y		Y	Y	Y								
			Qbu		Y		Y	Y	Y								
			Qci		N/A		N/A	N/A	Y								
			IP		Y		Y	Y	Y								
			MAC		Y		Y	Y	Y								
			VLAN config		Y		Y	Y	Y								
		Web based configuration	Qbv				Y		Y								
			Qbu				Y		Y								
			Qci		N/A		N/A	N/A	Y								

Table 4. Key features*...continued*

Feature				i.MX 6ULL 14x14 EVK	i.MX 8DXL LPDDR4 EVK	i.MX 8M Mini LPDDR4 EVK	i.MX 8M Plus LPDDR4 EVK	i.MX 93 EVK	LS1028A RDB	LS1021A TSN	LS1021A IOT	LS1021A TWR	LS10-12 ARDB	LS10-43 ARDB	LS10-46 ARDB	LS10-46 AFRWY	LX2160A RDB
		Dynamic topology discovery					Y		Y								
		Dynamic TSN configuration	Qci		N/A		N/A	N/A	Y								
			CQF						Y								
			Qbv				Y		Y								
	AVB standards	AVTP Talker/ Listener		Y		Y	Y										
		AVDECC		Y		Y	Y										
		MAAP		Y		Y	Y										
		Milan		Y		Y	Y										
		Media clock recovery		Y			Y										
	IEEE 1588/802.1AS				Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y
Industrial Protocol	EtherCAT master	IGH EtherCAT master stack		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
		IGH native Ethernet device driver			Y	Y	Y	Y	Y					Y	Y		
		SOEM				Y	Y										
	FlexCAN								Y		Y						
	CANopen										Y						
	OPC UA	open62541		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
		OPC UA PubSub over TSN			Y		Y	Y	Y								
	BEE (Mikroe Click board)								Y								
	BLE (Mikroe Click board)								Y								
	NFC (Mikroe Click board)								Y								
	Modbus	Modbus slave and master		Y	Y	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	Y	Y
		Modbus-TCP		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
		Modbus-ASCII		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

2.3 Open, fixed, and closed issues

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

- Open issues do not currently have a resolution. Workaround suggestions are provided where possible.
- Fixed issues have a software fix that has been integrated into the '*Fixed In*' Release.
- 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.

Table 5. Open issues in Real-time Edge software v2.4

ID	Description	Opened In	Workarounds
INDLINUX-1980	After setting the swpN link speed to 100M and sending some pre-emptable frames from DUT1 (which are not received at DUT2 Linux), the swpN Tx interface of DUT1 is blocked. It does not even ping. Using the commands "`ifconfig swpN down`" or "`ifconfigswpN up`" does not unblock it.	Real-time Edge software v2.1	

Table 6. Fixed Issues in Real-time Edge Software v2.4

ID	Description	Opened In	Fixed In
INDLINUX-2449	SOEM application does not work correctly on M7 core on i.MX8M Plus EVK when being loaded by u-boot.	Real-time Edge software v2.2	Real-time Edge software v2.4

Table 7. Closed Issues in Real-time Edge Software v2.4

ID	Description	Opened In	Disposition
None	-	-	-

3 Real-time System

Real-time Edge software supports real-time system features: Preempt-RT Linux, BareMetal, Jailhouse, and Harpoon (RTOS on Cortex-A).

3.1 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.

3.1.1 System Real-time Latency tests

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

3.1.1.1 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:

```
$ cyclictest -p90 -h50 -D30m
```

Note: *For detailed parameters of Cyclictest, refer to Cyclictest Web Page.*

3.1.2 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**:

```
$ arm-linux-gnueabihf-gcc <filename>.c -o <filename>.out -lrt
 -Wall
```

- **Using the native compiler on a target example**:

```
$ 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.

3.2 BareMetal

3.2.1 Overview

The following sections provide an overview of the Real-time Edge BareMetal framework including:

- Features supported
- Getting started with BareMetal framework using the supported platforms:
  - NXP Layerscape platforms

– i.MX 8M platforms.

It also describes how to run a sample BareMetal framework on the host environment and develop customer-specific applications based on BareMetal framework.

3.2.1.1 BareMetal framework

The BareMetal framework targets to support 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 that directly. The figure below depicts the BareMetal framework architecture.

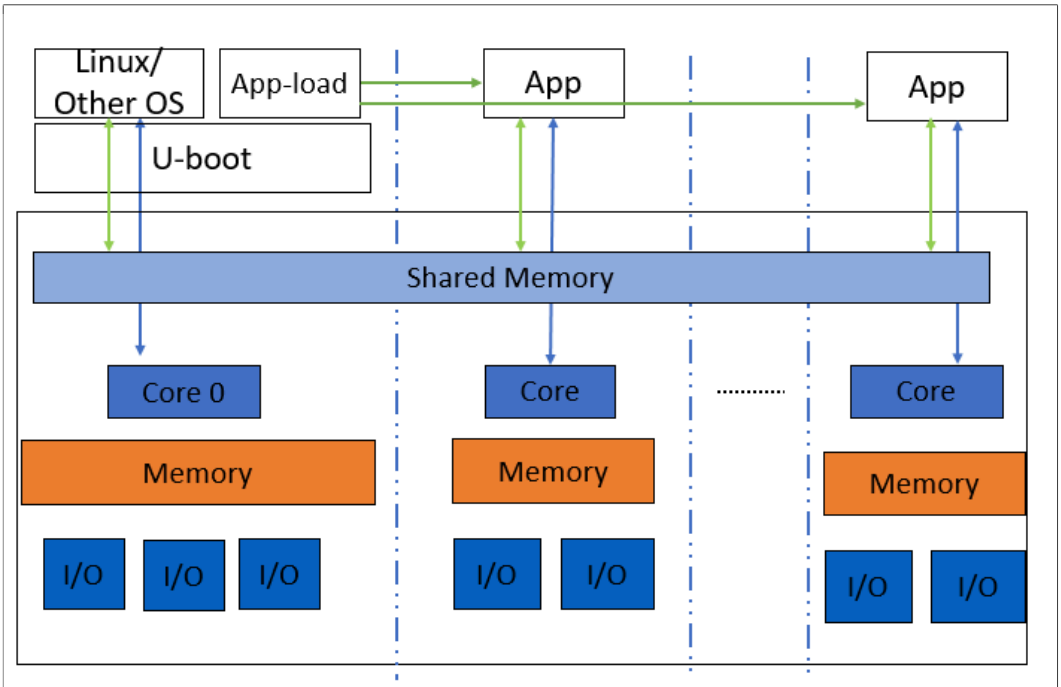


Figure 2. BareMetal framework architecture

The main features of the BareMetal framework are as follows:

- Core0 runs as master, which runs 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 U-Boot, it then loads the BareMetal application to the slave cores and starts the BareMetal application. The following figure depicts the boot flow diagram:

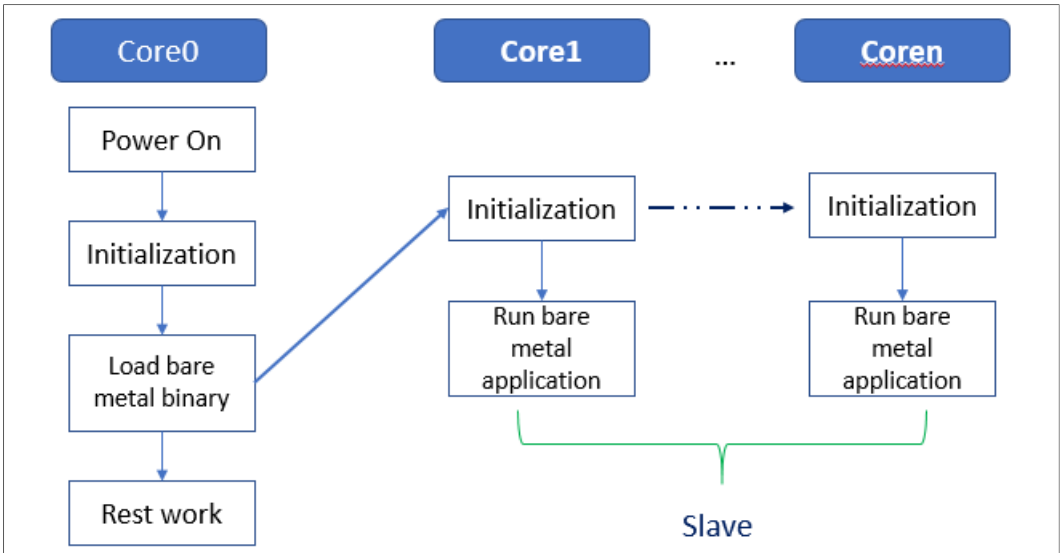


Figure 3. BareMetal framework boot flow diagram

3.2.1.2 Supported platforms

The table below lists the industrial IoT features supported by various NXP processors and boards.

Table 8. Industrial IoT features supported by NXP processors

Processor	Board	Main features supported
i.MX 8M Mini	i.MX 8M Mini LPDDR4 EVK	UART, IPI, data transfer, Ethernet, GPIO
i.MX 8M Plus	i.MX 8M Plus LPDDR4 EVK	UART, IPI, data transfer, Ethernet, GPIO
LS1028A	LS1028ARDB	I2C, UART, ENETC, IPI, data transfer, SAI
LS1043A	LS1043ARDB	IRQ, IPI, data transfer, Ethernet, IFC, I2C, UART, FMan, USB, PCIe
LS1046A	LS1046ARDB	IRQ, IPI, data transfer, Ethernet, IFC, I2C, UART, FMan, QSPI, USB, PCIe
LS1021A	LS1021A-IoT	GPIO, IRQ, IPI, data transfer, IFC, I2C, UART, QSPI, USB, PCIe, FlexCAN
LX2160A/Rev2	LX2160ARDB	UART, IPI, data transfer

3.2.2 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).

3.2.2.1 Hardware and software requirements

The following are required for running BareMetal framework scenarios:

- Hardware: i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, LS1028ARDB, LS1043ARDB, LS1046ARDB, LS1021A-IoT, LX2160ARDB, and serial cables.
- Software: Real-time Edge Software v2.0 release or later.

3.2.2.2 Hardware setup

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

3.2.2.2.1 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
```

   /dev/ttyUSB2 is used for core0 (master core), /dev/ttyUSB3 is used 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
```

   /dev/ttyUSB1 is used for core0 (master core), /dev/ttyUSB0 is used for core1, core2, and core3 (slave cores).
3. GPIO setup
   For GPIO test on i.MX 8M Plus LPDDR4 EVK, pin 7 and pin 8 of J21 should be connected by a jumper.

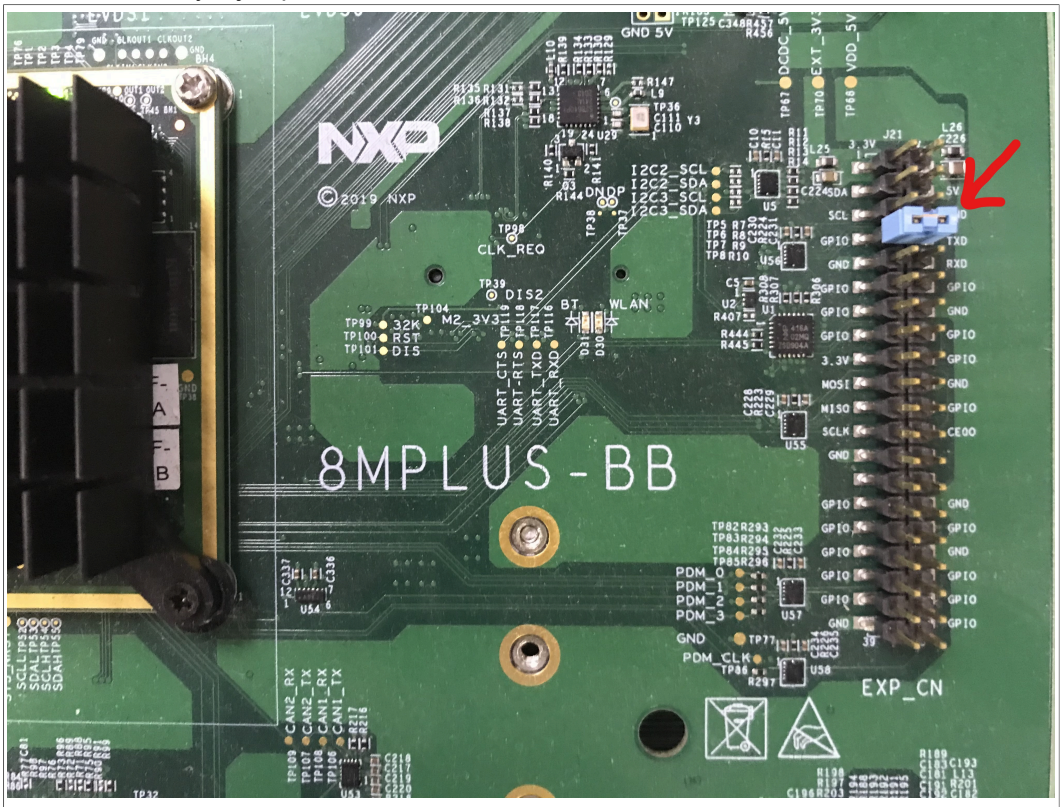


Figure 4. Connections for GPIO test on i.MX 8M Plus LPDDR4 EVK board

4. For GPIO test on i.MX 8M Mini LPDDR4 EVK, pin7 and pin8 of J1003 should be connected by a jumper.

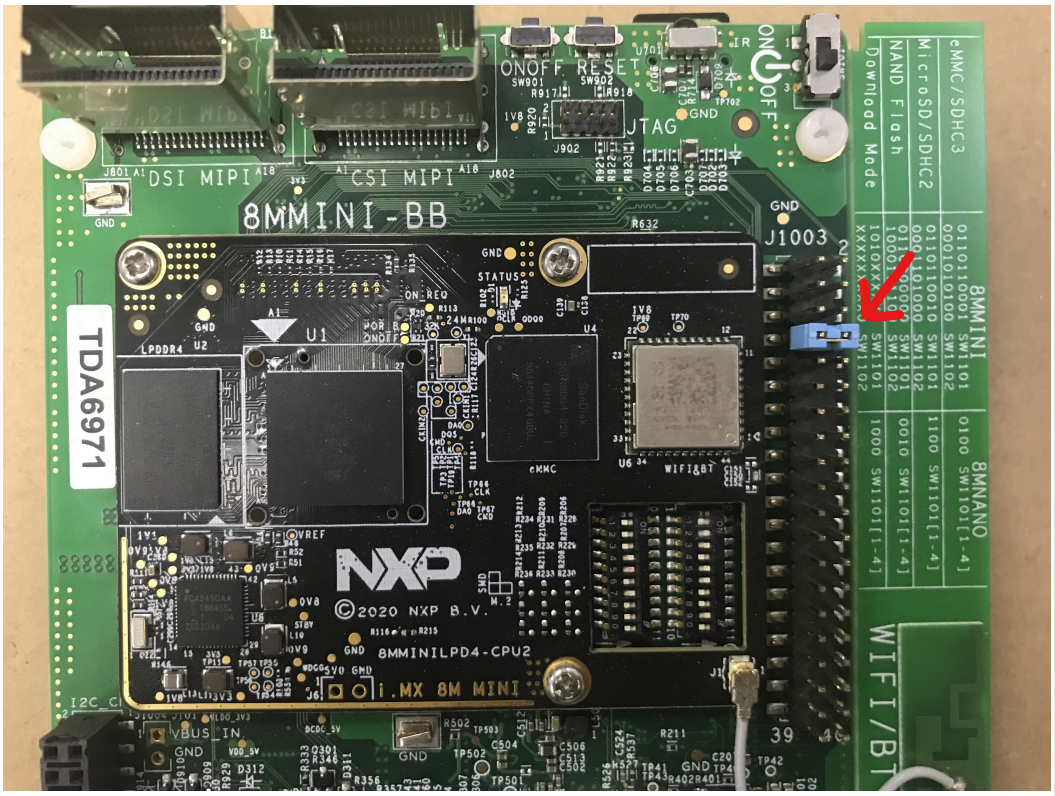


Figure 5. Connections for GPIO test on i.MX 8M Mini LPDDR4 EVK board

3.2.2.2.2 LS1028ARDB, LX2160ARDB, LS1043ARDB, or LS1046ARDB

In case, either the LS1028ARDB, LX2160ARDB, LS1043ARDB, or LS1046ARDB hardware boards are used for developing the Real-time Edge BareMetal framework, two serial cables are needed. One serial cable is used for core0, to connect to UART1 port, and the other one is used for slave cores, and connects to the UART2 port.

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

3.2.2.2.3 LS1021A-IoT board

Two serial cables are needed. One is used for core0, which connects to USB0/K22 port for UART0. The other cable is used for slave cores to connect J8 and J17 together for UART1. The table below describes the pins for UART1.

Table 9. UART pins

Pin name	Function
J8 pin7	Ground
J17 pin1	Uart1_SIN
J17 pin2	Uart1_SOUT

The figure below depicts the UART1 hardware connections.

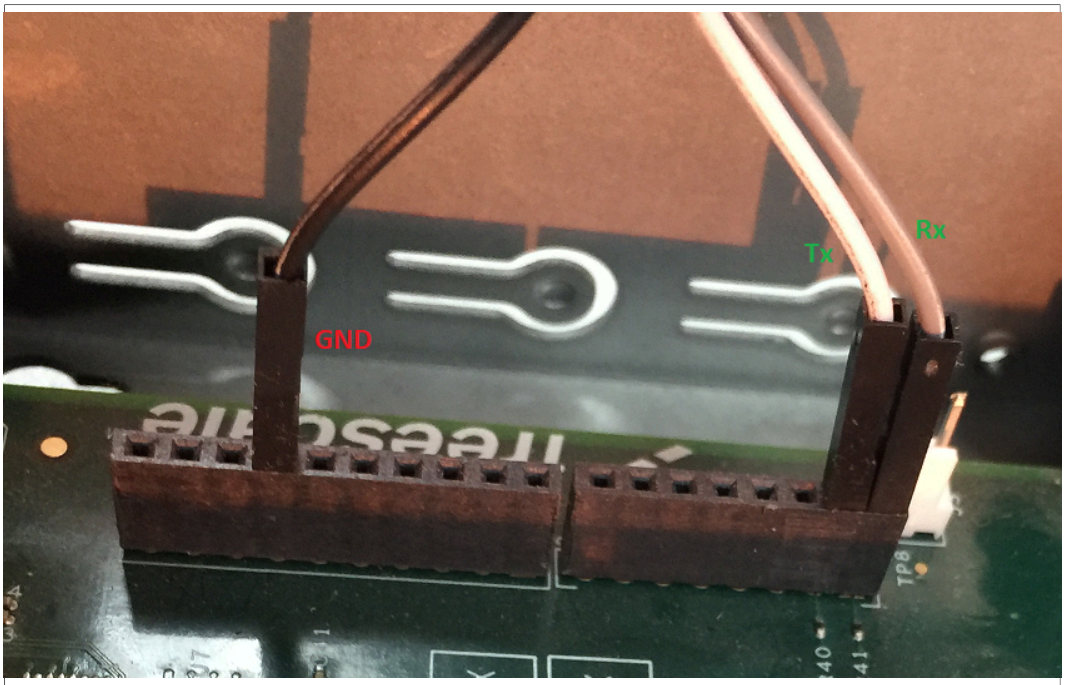


Figure 6. UART1 hardware connection

In order to test GPIO, connect the two pins, GPIO24 and GPIO25 together, which are through J502.3 and J502.5.

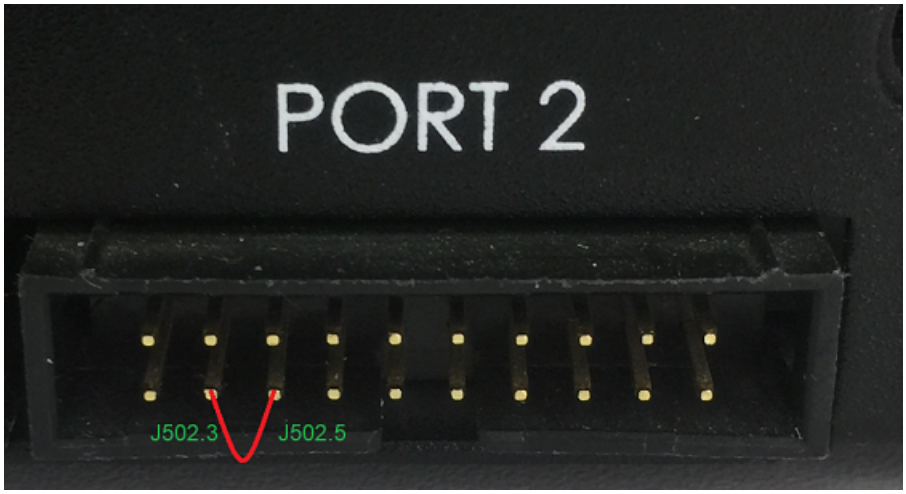


Figure 7. GPIO24 and GPIO25 connection on LS1021A-IoT

3.2.2.3 Building the BareMetal images from U-Boot source code

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".

3.2.2.3.1 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-v2.4-baremetal-202212`
3. Configure cross-toolchain on your host environment.
4. Then, run the following commands:

```
/* build BareMetal image for i.MX 8M Mini LPDDR4 EVK Rev.C
 board */
 $ make imx8mm_evk_bm_defconfig
 $ make
/* build BareMetal image for i.MX 8M Plus LPDDR4 EVK board */
 $ make imx8mp_evk_bm_defconfig
 $ make
/* build BareMetal image for LS1021A-IoT board */
 $ make ls1021aiot_bm_defconfig
 $ make
/* build BareMetal image for LS1028ARDB board */
 $ make ls1028ardb_bm_defconfig
 $ make
/* build BareMetal image with SAI for LS1028ARDB board */
 $ make ls1028ardb_bm_sai_defconfig
 $ make
/* build BareMetal image for LS1043ARDB board */
 $ make ls1043ardb_bm_defconfig
 $ make
/* build BareMetal image for LS1046ARDB board */
 $ make ls1046ardb_bm_defconfig
 $ make
/* build BareMetal image for LX2160ARDB board */
 $ make lx2160ardb_bm_defconfig
 $ make
```

5. Finally, the file u-boot.bin (or `u-boot-dtb.bin`, only for lx2160ardb) used for BareMetal is generated.

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

3.2.2.4 Building the image through Yocto

There are two methods to build the BareMetal images. One method is to compile the images in a standalone way which is described in Section 3.2.2.3. The second method is to build the BareMetal images using Real-time Edge software framework, which 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.

3.2.2.4.1 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.

3.2.2.4.2 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 LS1021A-IoT, LS1043ARDB, LS1046ARDB, LX2160ARDB, i.MX 8M Plus LPDDR4 EVK, and i.MX 8M Mini LPDDR4 EVK board.

3.2.2.4.2.1 Building the BareMetal images

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

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

For LS1021A-IOT BareMetal image:

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1021aiot source
 real-time-edge-setup-env.sh -b build-ls1021aiot-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 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-imx8mpevk-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-ix8mmevk-bm
```

Then, use:

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

3.2.2.4.3 Booting up the Linux with BareMetal

Use the following steps to bootup 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
# or in some other host machine:
$ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wic of=/
dev/mmcblkx
# 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.

3.2.2.5 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 to output `0` and `1` to simulate an external hardware. The other one is 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. On LS1046ardb, GPIO2_01 and GPIO2_02 are selected. GPIO2_01 is used to simulate an external hardware, GPIO2_02 is used to trigger an interrupt. The board needs to rework to connect these two pins. In LS1046ARDB, TP14 and TP13 are connected to GPIO2_01 and GPIO2_02 separately. We need to connect TP14 with TP13. The figure below shows how to connect TP14 and TP13.

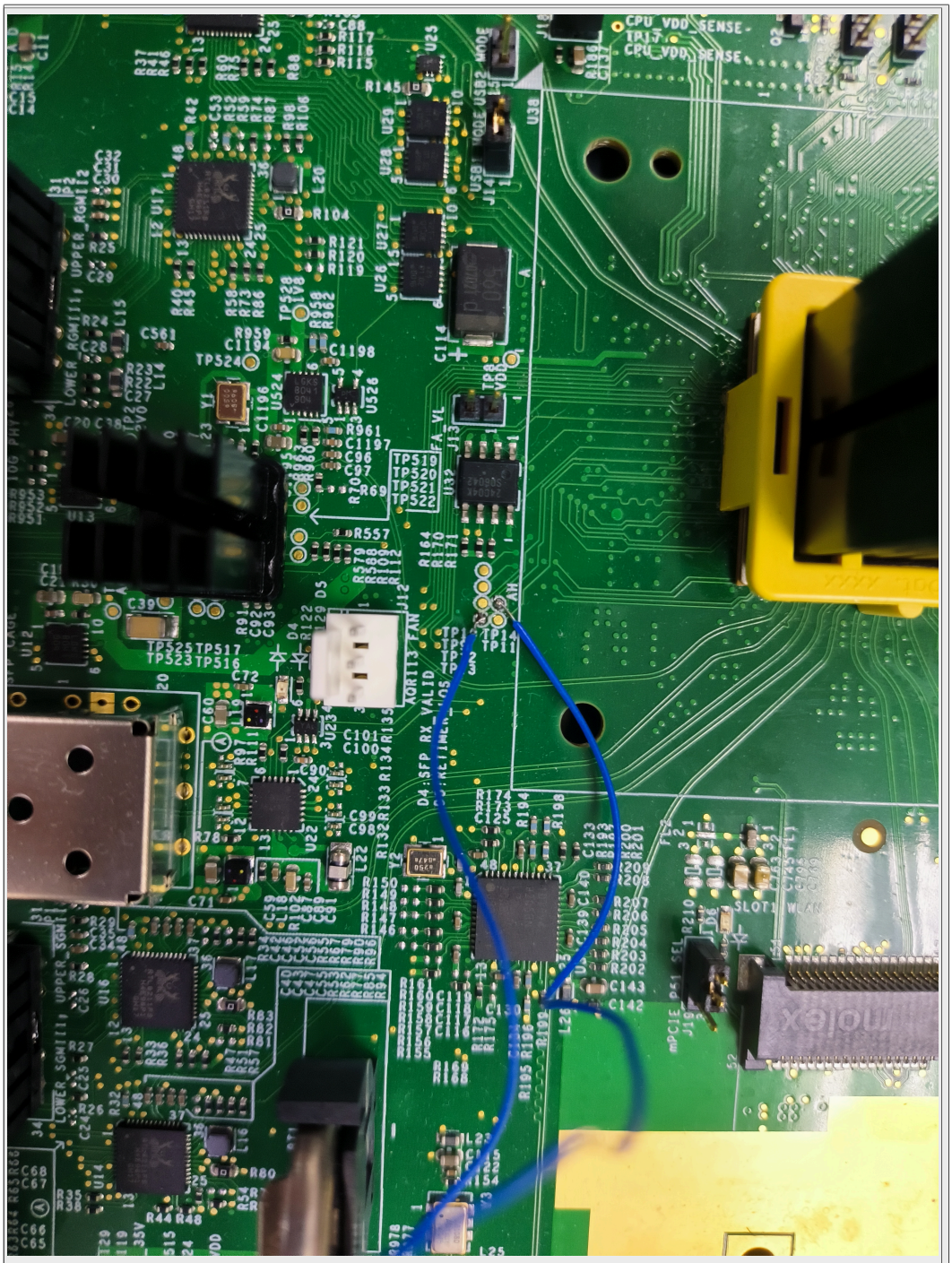


Figure 8. LS1046ARDB hardware interrupt routing to multiple cores

GPIO2_01/02 are multiplexed with `SPI_CS_B[0]` and `SDHC_DAT[4]` signals. `RCW[382 ~ 383]` needs to change 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. The below code could be added in Linux kernel file `fsl-ls1046a-rdb-sdk-bm.dts` to disable GPIO2.

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

Under BareMetal, gpio_int command provides "*enable*", "*start*" and "*stop*" commands to control these two pins.

- **gpio_int** enable - Configure GPIO2_01 and GPIO2_02 and enable interrupt
- **gpio_int** start - Set GPIO2_01 to high
- **gpio_int** stop - Set GPIO2_01 to low

gpio_int could run under BareMetal console.

"`gpio_int enable`" command should be run first to initialize the GPIO2_01/02. Then, use command pair "`gpio_int start`" and "`gpio_int stop`" to pull high and pull down GPIO2_01. After the command pair, GPIO2_02 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_int enable
=> gpio_int start
1:data: 0x60000000
1:event: 0x40000000
=> gpio_int stop
1:Time(us): 0xb8f5b8c8, GPIO event: 0x60000000
3:Time(us): 0xb8f5b8c9, Get the SGI from CoreID: 1
1:data: 0x0
2:Time(us): 0xb8f5b8c9, Get the SGI from CoreID: 1
1:event: 0x0
=>
```

3.2.3 Running examples

The following sections describe how to run the BareMetal examples on the host environment for LS1021A-IoT board. Similar steps can be followed for LS1028ARDB, LS1043ARDB, LS1046ARDB, i.MX 8M Mini LPDDR4 EVK, and i.MX 8M Plus LPDDR4 EVK board.

3.2.3.1 Preparing the console

Prepare a USB to TTL serial line, connect the pins to LPUART of Arduino pins to use it as UART1, refer to Section 3.2.2.2.

In order to change LPUART to UART pins, modify 44th byte of RCW to 0x00.

In current BareMetal framework design, two UART ports are used as console. UART1 is used for master core and UART2 is used for slave cores.

- For LS1021A, UART2 is pin-muxed with LPUART, so users should change the RCW to enable UART2 on master core. This modification has been implemented in the code for LS1021A-IoT board, so users need not modify the code for this board.

- For customer-specific boards, apply the change to the RCW file:

```
bit[354~356] = 000
bit[366~368] = 111
```

3.2.3.2 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 3.2.2.3 and Section 3.2.2.4 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.bin on 0x84000000 using the command below:

     ```
     => tftp 0x84000000 xxxx/u-boot.bin
     ```

     Where
     - xxxx is your tftp server directory.
     - 0x84000000 is the address of CONFIG_SYS_TEXT_BASE on bare metal for Layerscape platforms.

     Note: *The address is 0x50200000 for i.MX 8M Plus LPDDR4 EVK and i.MX 8M Mini LPDDR4 EVK boards.*
  2. Then, start the BareMetal cores using the command below:

     ```
     => cpu 1 release 0x84000000
     ```

     Note: *in command "cpu <num> start 0x84000000", the 'num' can be 1, 2, 3, ... to the max cpu number.*
     For i.MX 8M Plus LPDDR4 EVK and i.MX 8M Mini LPDDR4 EVK boards, below command will be used:

     ```
     => dcache flush;cpu 1 release 50200000;sleep 6;cpu 2
      release 50200000;sleep 2;cpu 3 release 50200000;
     ```
  3. Last, the UART1 port displays the logs, and the bare metal application runs on slave cores successfully.

  The figure below displays a sample output log.

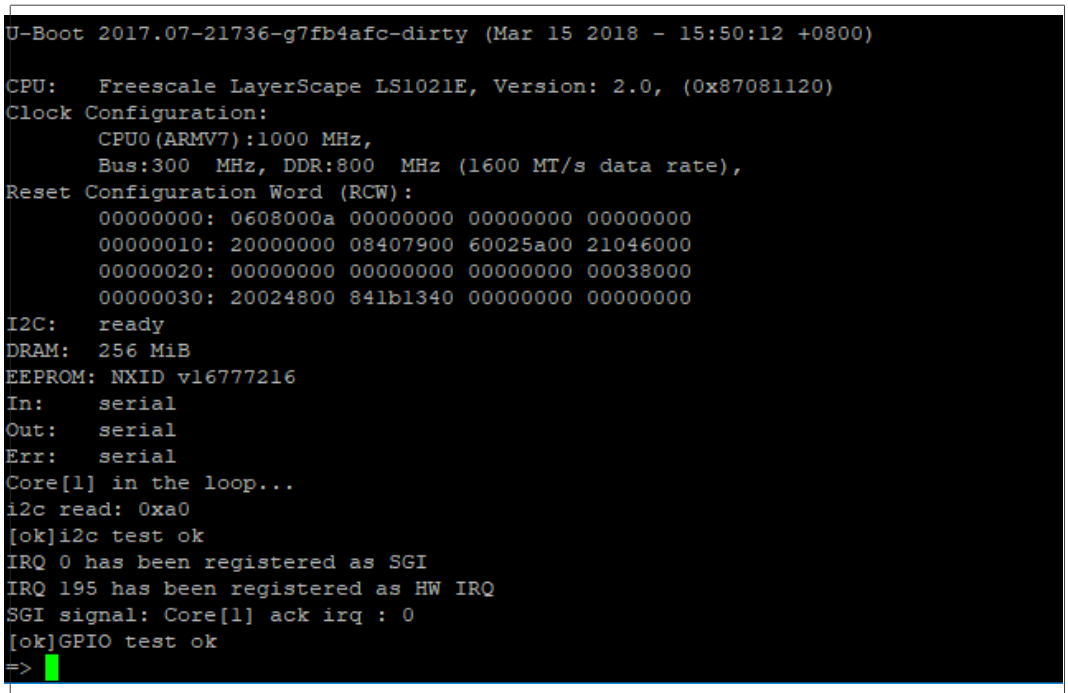


Figure 9. BareMetal output logs

3.2.4 Development based on BareMetal framework

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

3.2.4.1 Developing the BareMetal application

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

3.2.4.2 Example software

3.2.4.2.1 Main file app.c

The file `<industry-Uboot path>/app/app.c`, is the main entrance for all applications. Users can modify the app.c file to add their applications.

- If using standalone method to build the BareMetal image as described in Section 3.2.2.3, just change the directory to industry-Uboot path to check the `app.c` file.
- If using Real-time Edge software to compile the BareMetal binary, you should change to the building directory to check 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.

```
void core1_main(void)
{
  test_i2c();
  test_irq_init();
```

```
    test_gpio();
    return;
}
```

3.2.4.2.2 Common header files

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

Table 10. Common header file description

Header file	Description
asm/io.h	Read/Write IO APIs. For example, `__raw_readb`, `__raw_writeb`, `out_be32`, and `in_be32`.
linux/string.h	APIs for manipulating strings. For example, `strlen`, `strcpy`, and `strcmp`.
linux/delay.h	APIs used for small pauses. For example, `udelay` and `mdelay`.
linux/types.h	APIs specifying common types. For example, `__u32` and `__u64`.
common.h	Common APIs. For example, `printf` and `puts`.

3.2.4.2.3 GPIO file

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

Here is an example for the ls1021aiot board:

On ls1021aiot board, first you need the GPIO header file, asm-generic/gpio.h, which includes all interfaces for the GPIO. Then, configure GPIO25 to OUT direction, and configure GPIO24 to IN direction. Last, by writing the value `1` or `0` to GPIO25, you can receive the same value from GPIO24.

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

Table 11. GPIO APIs and their description

Function declaration	Description
`gpio_request (ngpio, label)`	Requests GPIO. • `ngpio` - The GPIO number, such as 25, that is for GPIO25. • `label` – the name of GPIOI request. Returns **0** if **OK**, **-1** on **Error**.
`gpio_direction_output (ngpio, value)`	Configures the direction of GPIO to OUT and writes the value to it. • `ngpio` - The GPIO number, such as 25,that is, for GPIO25. • `value` – the value written to this GPIO. Returns **0** if **Low**, **1** if **High**, **-1** on **Error**.
`gpio_direction_input (ngpio);`	Configures the direction of GPIO to IN. `ngpio` - The GPIO number, such as 24, that is for GPIO24; Returns **0** if **OK**, **-1** on **Error**.

Table 11. GPIO APIs and their description*...continued*

Function declaration	Description
`gpio_get_value (ngpio)`	Reads the value. • `ngpio` - The GPIO number, such as 24, that is for GPIO24; Returns **0** if **Low**, **1** if **High**, **-1** on **Error**.
`gpio_free (ngpio)`	Frees the GPIO just requested. • `ngpio` - The GPIO number, such as 24, that is for GPIO24; Returns **0** if **OK**, -1 on **error**.

3.2.4.2.4 I2C file

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 ls1021aiot board, include the I2C header file, i2c.h, which contains all interfaces for I2C. Then, read a fixed data from offset 0 of Audio codec device (0x2A). If the data is 0xa0, the message, `[ok]I2C test ok`, is displayed on the console.

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

Function declaration	Description
`int i2c_set_bus_num (unsigned int bus)`	Sets the I2C bus. `bus`- bus index, zero based Returns **0** if **OK**, **-1** on error.
`int i2c_read (uint8_t chip, unsigned int addr, int alen, uint8_t *buffer, int len)`	Read data from I2C device chip. • `chip` - I2C chip address, range 0..127 • `addr` - Memory (register) address within the chip • `alen` - Number of bytes to use for address (typically 1, 2 for larger memories, 0 for register type devices with only one register) • `buffer` - Where to read/write the data • `len` - How many bytes to read/write Returns **0** if **OK**, **not 0** on error.
`int i2c_write (uint8_t chip, unsigned int addr, int alen, uint8_t *buffer, int len)`	Writes data to I2C device chip. • `chip` - I2C chip address, range 0..127 • `addr` - Memory (register) address within the chip • `alen` - Number of bytes to use for address (typically 1, 2 for larger memories, 0 for register type devices with only one register) • `buffer` - Where to read/write the data • `len` - How many bytes to read/write Returns **0** if **OK**, **not 0** on error.

3.2.4.2.5 IRQ file

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 3.2.4.4. SGI IRQ id is 0-15. The SGI IRQ id '8' is reserved for ICC.

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

The table below shows the APIs used in the sample file, *test_irq_init.c*.

Table 13. IRQ APIs and their description

Return type API name (parameter list)	Description
`void gic_irq_register (int irq_num, void (*irq_handle)(int))`	Registers an IRQ function. • `irq_num`- IRQ id, 0-15 for SGI, 16-31 for PPI, 32-1019 for SPI • `irq_handle` – IRQ function
`void gic_set_sgi (int core_mask, u32 hw_irq)`	Sets a SGI IRQ signal. • `core_mask` – target core mask • `hw_irq` – IRQ id
`void gic_set_target (u32 core_mask, unsigned long hw_irq)`	Sets the target core for hw IRQ. • `core_mask` – target core mask • `hw_irq` – IRQ id
`void gic_set_type (unsigned long hw_irq)`	Sets the type for hardware IRQ to identify whether the corresponding interrupt is edge-triggered or level-sensitive. • `hw_irq` – IRQ id

3.2.4.2.6 QSPI file

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 below shows the APIs used in the file test_qsip.c example.

Table 14. QSPI APIs

API name (type)	Description
`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. • "`bus_dev`" - If the bus is found. • "`new`" – If the device is found. Returns `0` if OK, `-ENODEV` on error.

Table 14. QSPI APIs*...continued*

API name (type)	Description
`spi_flash_probe_bus_cs(bus, cs, speed, mode, &new)`	Initializes the SPI flash device. • "`bus`" - bus index, zero based. • "`cs`" – the value to Chip Select mode. • "`speed`" – SPI flash speed, can use 0 or CONFIG_SF_DEFAULT_SPEED. • "`mode`" –SPI flash mode, can use 0 or CONFIG_SF_DEFAULT_MODE. • "`new`" – If the device is initialized. Returns `0` if OK, `-ENODEV` on error.
`dev_get_uclass_priv(new)`	Gets the SPI flash. • "`new`" - The device being initialized. Returns `flash` if OK , `NULL` on error.
`spi_flash_erase(flash, offset, size)`	Erases the specified location and length of the flash content, erases the content of all. • "`flash`" - Flash is being initialized. • "`offset`" – Flash offset address. • "`size`" - Erase the length of the data. Returns `0` if OK, `!0` on error.
`spi_flash_read(flash, offset, len, vbuf)`	Reads flash data to memory. • "`flash`" - The flash being initialized. • "`offset`" – Flash offset address. • "`len`" - Read the length of the data. • "`vbug`" - the buffer to store the data read Returns `0` if OK, `!0` on error.
`spi_flash_write(flash, offset, len, buf)`	Writes memory data to flash. • "`flash`" - The flash being initialized. • "`offset`" – Flash offset address. • "`len`" - Write the length of the data. • "`buf`" - the buffer to store the data write Returns `0` if OK, `!0` on error.

3.2.4.2.7 IFC

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:

```
1:NAND:  512 MiB
1:Flash: 128 MiB
```

or （LS1046ARDB）

```
1:NAND:  512 MiB
```

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), 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
  ................
1:60000040: 96000000 01000000 78015709 10e00000
  .........W.x....
1:60000050: 00001809 08000000 18045709 9e000000
  .........W......
1:60000060: 1c045709 9e000000 20045709 9e000000
  .W.......W. ....
1:60000070: 00065709 00000000 04065709 00001060
  .W.......W..`...
1:60000080: c000ee09 00440000 58015709 00220000
  ......D..W.X..".
1:60000090: 40800089 01000000 40006108 f56b710a
  ...@.....a.@.qk.
1:600000a0: ffffffff ffffffff ffffffff ffffffff
  ................
```

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]
nand write.raw[.noverify] - addr off|partition [count]
    Use 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
```

3.2.4.2.8 Ethernet

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.

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 15. Net APIs and their description

API name (type)	Description
`void net_init (void)`	Initializes the network
`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.

3.2.4.2.9 USB file

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 16. USB APIs and their description

API name (type)	Description
`int usb_init(void)`	Initializes the USB controller.
`int usb_stop(void)`	Stops the USB controller.
`int usb_stor_scan(int mode)`	Scans the USB and reports device information to the user if mode = 1 • `Mode` – if mode = 1, the information is returned to user. Returns • the current device, or • -1 (if device not found).
`struct blk_desc *blk_get_devnum_by_type(enum if_type if_type, int devnum)`	Get a block device by type and number. • `If_type` – Block device type • `devnum` - device number Returns • Points to block device descriptor, or • NULL (if not found).
`unsigned long blk_dread(struct blk_desc *block_dev, lbaint_t start, lbaint_t blkcnt, void *buffer);`	Reads data from USB device. • `block_dev` – block device descripter • `start` – start block • `blkcnt` – block number • `buffer` – buffer to store the data Returns the block number from which, data is read.
`unsigned long blk_dwrite(struct blk_desc *block_dev, lbaint_t start, lbaint_t blkcnt, const void *buffer);`	Writes data to USB device. • `block_dev` – block device descripter • `start` – start block • `blkcnt` – block number • `buffer` – buffer to store the data Returns the block number to which data is written.

3.2.4.2.10 PCIe file

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 17. PCIe APIs and their description

API name (type)	Description
`void pci_init(void)`	Initializes the PCIe controller. Does not return a value.
`int uclass_get_device_by_seq(enum uclass_id id, int seq, struct udevice **devp)`	Gets the uclass device based on an ID and sequence: • `id` – uclass ID • `seq` – sequence • `devp` – Pointer to device Returns: • 0 if Ok. • Negative value on error.
`static inline int pci_eth_init(bd_t *bis)`	Initializes network card on the PCIe bus. • `Bis` – struct containing variables accessed by shared code Returns the number of network cards.
`int net_loop (enum proto_t protocol)`	Main network processing loop. • `enum proto_t protocol` - protocol type Returns: • 0 if Ok. • Negative value on error.

3.2.4.2.11 CAN file

The file `app/test_flexcan.c` provides a sample test case to test flexCAN and CANopen features. The following steps show the design process:

1. Register the receive interruption function for flexCAN.
2. Register an overflow interruption function for flextimer.
3. Initialize a list of callback functions.
4. Set the node ID of this node.

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

Table 18. CAN APIs and their description

API name (type)	Description
`void test_flexcan(void)`	It is the test code entry for flexCAN.
`void flexcan_rx_irq(struct can_module *canx)`	FlexCAN receive interruption function. • `canx` – flexCAN interface.
`void flexcan_receive(struct can_module *canx, struct can_frame *cf)`	FlexCAN receives CAN data frame. • `canx` – FlexCAN interface. • `cf` – CAN message.
`UNS8 setState(CO_Data* d, e_nodeState newState)`	Sets node state • `d` – object dictionary • `newState` – The state that requires to be set. Returns: • 0 if Ok, • > 0 on error.
`void canDispatch(CO_Data* d, Message *m)`	CANopen handles data frames that CAN receives: • `d` – object dictionary. • `m` – Received CAN message.

Table 18. CAN APIs and their description*...continued*

`int flexcan_send(struct can_module *canx, struct can_frame *cf)`	FlexCAN interface sends CAN message: • `canx` – FlexCAN interface. • `cf` – CAN message.
`void flextimer_overflow_irq(void)`	Flextimer overflow interruption handler function.
`void timerForCan(void)`	CANopen virtual clock. Call this function per 100 µs.

- The following log shows the CANopen slave node state:

```
=> flexcan error: 0x42242!
Note: slave node entry into the stop mode!
Note: slave node initialization is complete!
Note: slave node entry into the preOperation mode!
Note: slave node entry into the operation mode!
Note: slave node initialization is complete!
Note: slave node entry into the preOperation mode!
Note: slave node entry into the operation mode!
```

3.2.4.2.12 ENETC file

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 4.2.7 for other Net APIs.

Table 19. ENETC APIs and their description

API name (type)	Description
`void pci_init(void)`	Initializes the PCIe controller. Does not return a value.
`void eth_initialize(void)`	Initializes the Ethernet.

3.2.4.2.13 SAI file

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.

3.2.4.2.13.1 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

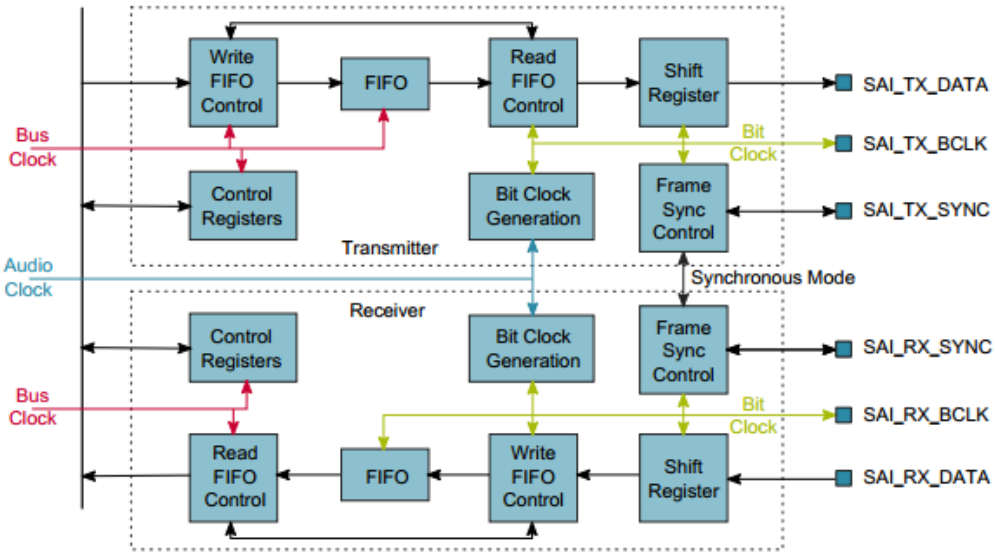


Figure 10. SAI block diagram

3.2.4.2.13.2 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, 32 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.

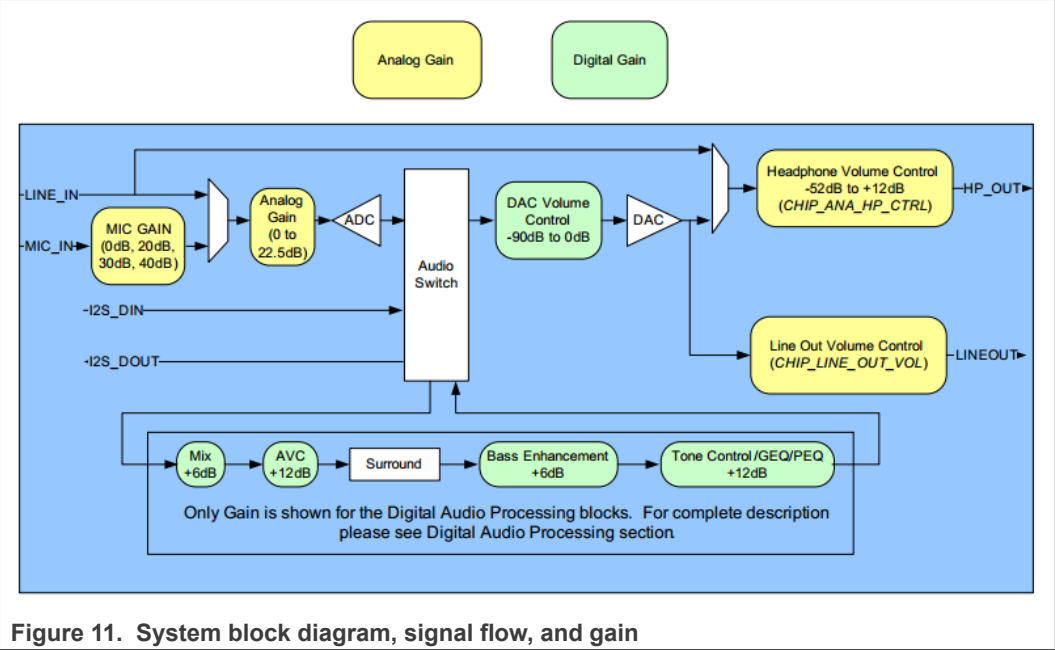
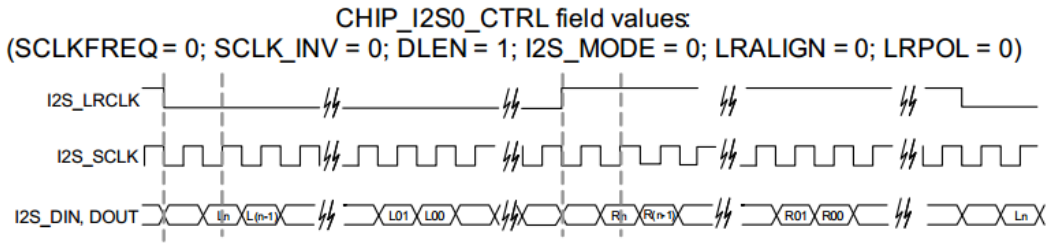


Figure 11. System block diagram, signal flow, and gain

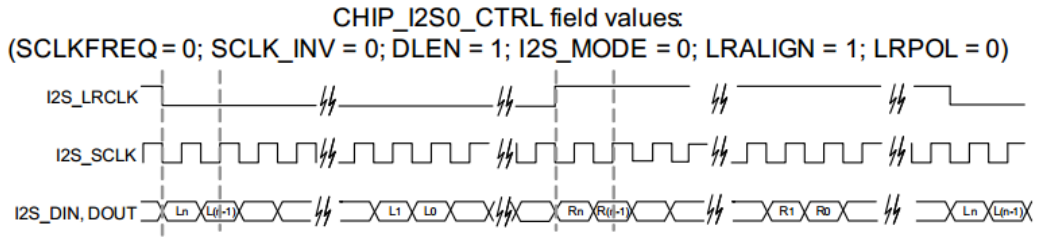
3.2.4.2.13.3 Digital interface formats

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

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

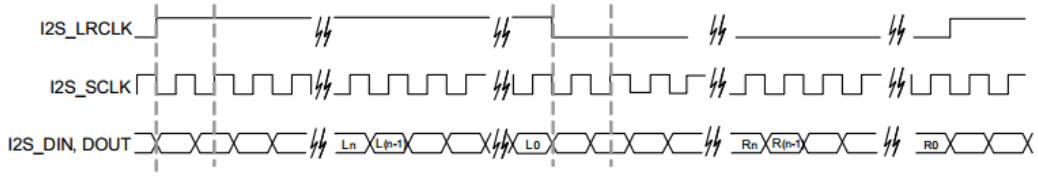


- **Left Justified Format (n = bit length)**



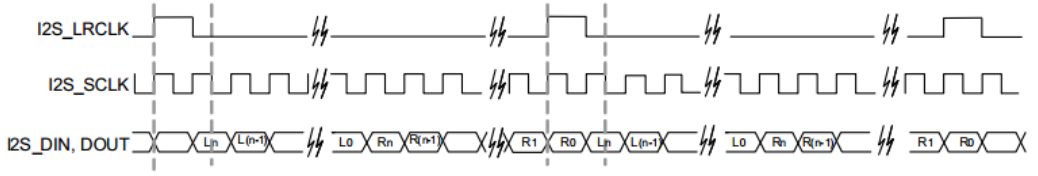
- **Right Justified Format (n = bit length)**

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



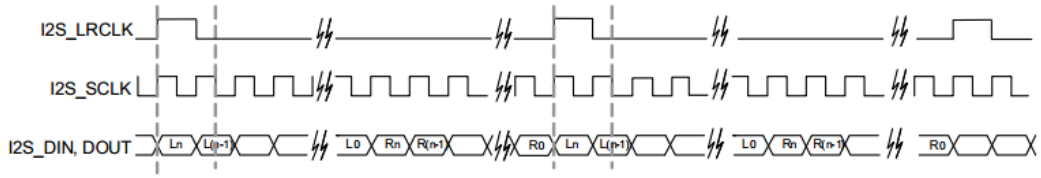
- **PCM Format A**

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



- **PCM Format B**

CHIP_I2S0_CTRL = 0x01F6
(SCLKFREQ = 1; MS = 1; SCLK_INV = 1; DLEN = 3; I2S_MODE = 2; LRALIGN = 1)



3.2.4.2.13.4 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
 ************************************************************
 sgtl5000 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!
 ************************************************************
```

3.2.4.3 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.

```
#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:

```
 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 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
```

3.2.4.4 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.

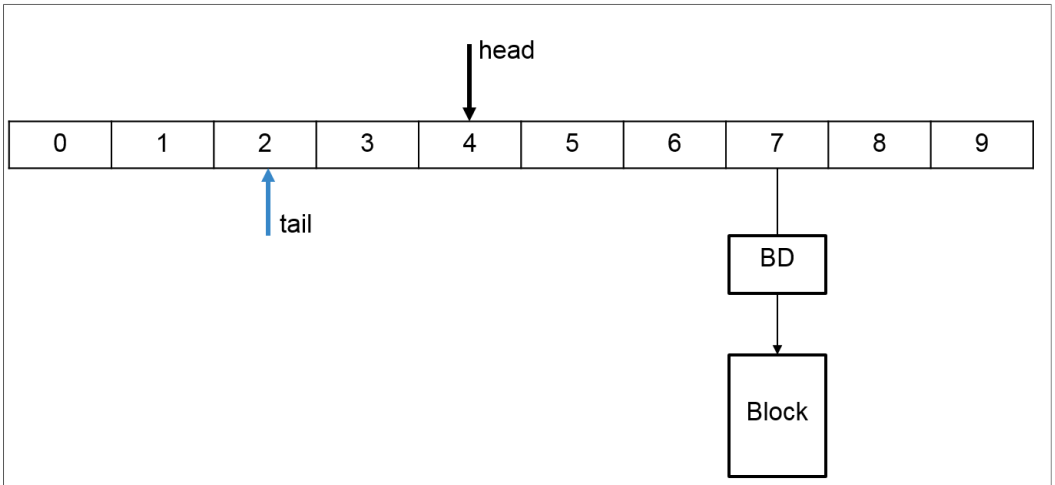


Figure 12. BD rings for inter-core communication

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.)

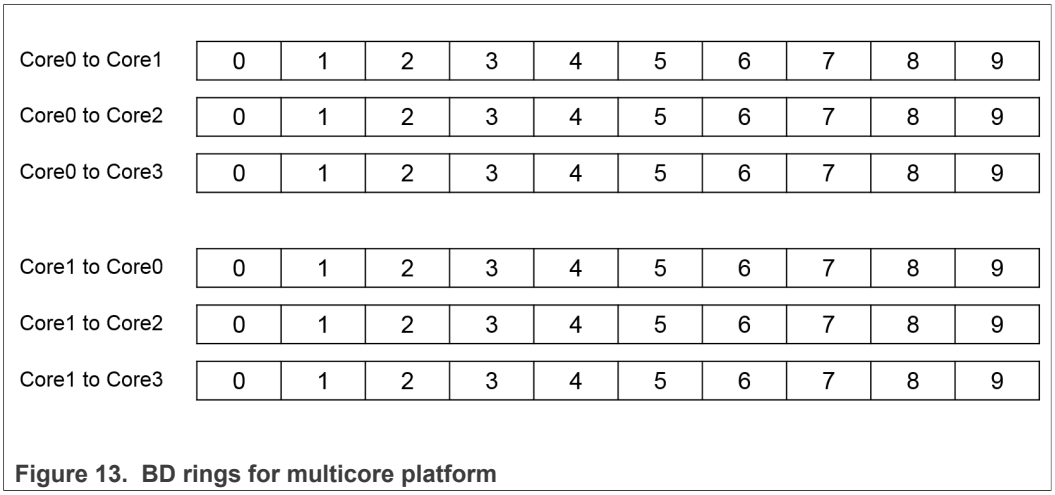


Figure 13. BD rings for multicore platform

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.

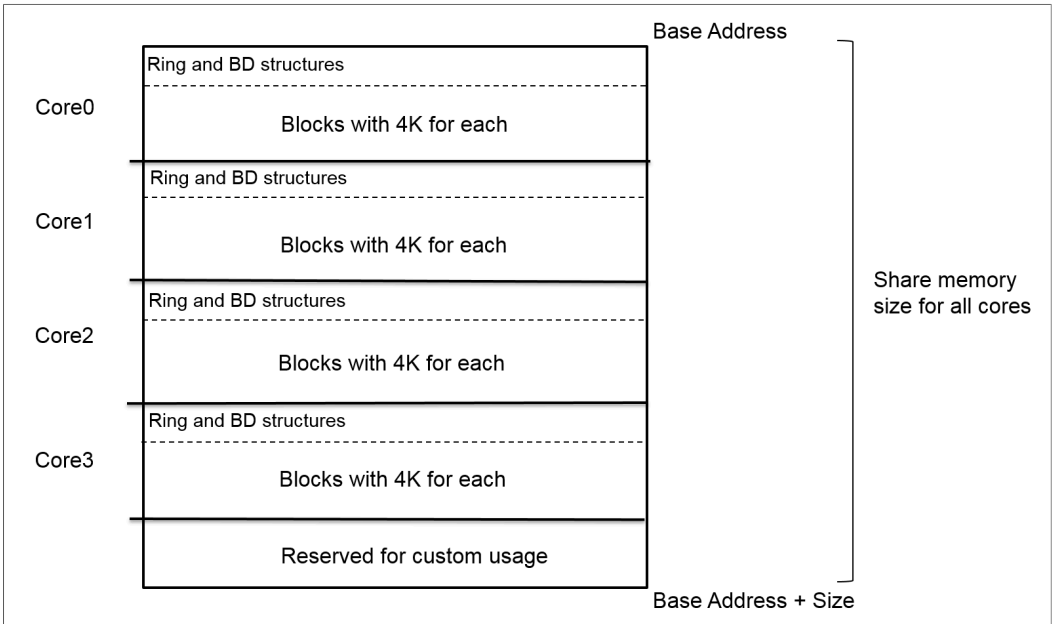


Figure 14. ICC shared memory map for the four-core platform

Generally, Core 0 runs Linux as master core while other cores run BareMetal as slaves. They obtain the same size of share 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 share memory is out of the ICC module and for the custom usage.

For LS1021A 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 share 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.

The ICC code for Linux user space in the repository: https://github.com/nxp-real-time-edge-sw/real-time-edge-icc.git.

```
├── 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 20. ICC APIs

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

3.2.4.4.1 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>          - ICC performance to
 cores <core_mask> with <counts> bytes
icc send <core_mask> <data> <counts>    - Send <counts>
 <data> to cores <core_mask>
icc irq <core_mask> <irq>              - Send SGI <irq> ID[0
 - 15] to <core_mask>
```

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: 0xd0200000
ICC send testing ...
Target cores: 0x2, bytes: 128
ICC send: 128 bytes to 0x2 cores success
all cores: reserved_share_memory_base: 0xdf000000; 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: 0xd00000c0; head: 0; tail: 0
busy_counts: 0; interrupt_counts: 0
#ring 1 base: 0xffff9133f030; dest_core: 1; SGI: 8
desc_num: 128; desc_base: 0xd00008c0; head: 1; tail: 1
busy_counts: 0; interrupt_counts: 1
#ring 2 base: 0xffff9133f060; dest_core: 2; SGI: 8
desc_num: 128; desc_base: 0xd00010c0; head: 0; tail: 0
busy_counts: 0; interrupt_counts: 0
#ring 3 base: 0xffff9133f090; dest_core: 3; SGI: 8
desc_num: 128; desc_base: 0xd00018c0; 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: 0xdf000000; 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: 00000000d3c008c0; 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: 00000000d3c010c0; head: 0; tail:
 0
1:busy_counts: 0; interrupt_counts: 0
1:#ring 3 base: 00000000d3c00090; dest_core: 3; SGI: 8
1:desc_num: 128; desc_base: 00000000d3c018c0; 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
```

3.2.4.5 Hardware resource allocation

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

3.2.4.5.1 LS1021A-IoT board

3.2.4.5.1.1 Linux DTS

Remove cpu1 node on DTS, and remove all the devices that baremetal has used.

3.2.4.5.1.2 Memory configuration

LS1021A-IoT board has a 1 GB size DDR. The DDR memory can be configured into three partitions: 512M for core0 (Linux), 256M for core1 (baremetal), and 256M for shared memory.

The configuration is in the path: include/configs/ls1021aiot_config.h.

```
#define CONFIG_SYS_DDR_SDRAM_SLAVE_SIZE (256 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_MASTER_SIZE (512 * 1024 * 1024)
```

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

Modify "`CONFIG_SYS_MALLOC_LEN`" in the *configs/ls1021aiot_bm_defconfig* to change the maximum size of malloc.

You can use functions included in malloc.h to allocate or free memory in your program. These functions are listed in the table below.

Table 21. Description of memory APIs

API name (type)	Description
void_t* 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 memory configuration for baremetal is shown in the figure below.

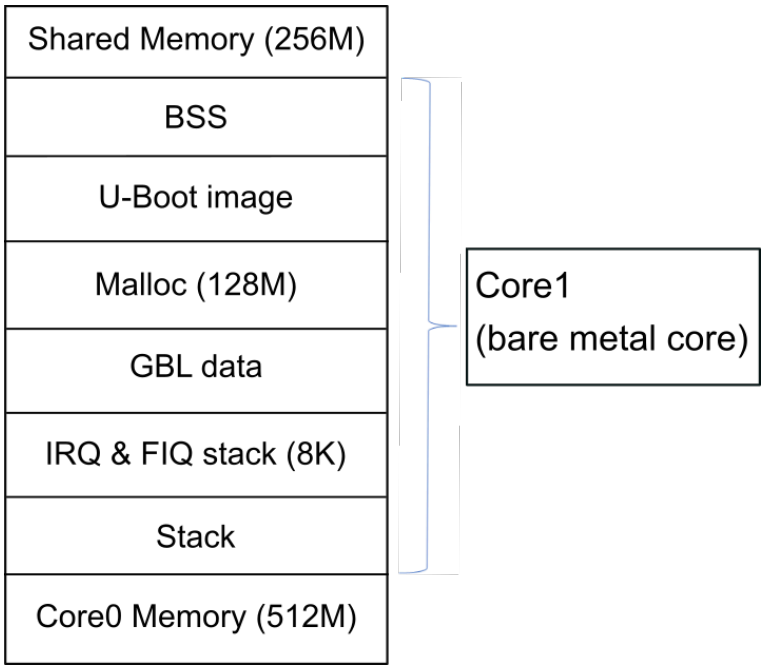


Figure 15. Memory configuration

3.2.4.5.1.3 GPIO

LS1021A has four GPIO controllers. The configuration is defined in the file:*arch/arm/dts/ls1021a-iot.dtsi*. You can add a GPIO node in the file ls1021a-iot.dtsi to assign a GPIO resource to different cores. Following is a sample code for adding a GPIO node.

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

3.2.4.5.1.4 I2C

LS1021A has three I2C controllers. Configure the I2C bus on `ls1021aiot_bm_defconfig` using the commands below:

```
CONFIG_SYS_I2C_MXC_I2C1=y
CONFIG_SYS_I2C_MXC_I2C2=y
CONFIG_SYS_I2C_MXC_I2C3=y
```

3.2.4.5.1.5 Hardware interrupts

LS1021A has six IRQs as external IO signals connected to interrupt the controller. We can use these six IRQs on bare metal cores. The ids for these signals, IRQ0-IRQ5 are: 195, 196, 197, 199, 200, and 201.

GIC interrupt APIs are defined in the file, asm/interrupt-gic.h. The following example shows how to register a hardware interrupt:

```
//register HW interrupt
void gic_irq_register(int irq_num, void (*irq_handle)(int));
```

```
void gic_set_target(u32 core_mask, unsigned long hw_irq);
void gic_set_type(unsigned long hw_irq);
```

3.2.4.5.1.6 IFC

LS1021A-IoT board has no IFC device, but the LS1021A SoC has an IFC interface. Since IFC is multiplexed with QSPI, you should modify the RCW to use the IFC interface, and add a configuration such as shown in the sample code provided below. Users can modify the code to support IFC as per the actual scenario.

```
#define CONFIG_FSL_IFC
#define CONFIG_SYS_CPLD_BASE 0x7fb00000
#define CPLD_BASE_PHYS CONFIG_SYS_CPLD_BASE
#define CONFIG_SYS_FPGA_CSPR_EXT (0x0)
#define CONFIG_SYS_FPGA_CSPR (CSPR_PHYS_ADDR(CPLD_BASE_PHYS) |
 \
CSPR_PORT_SIZE_8 | \
CSPR_MSEL_GPCM | \
CSPR_V)
#define CONFIG_SYS_FPGA_AMASK IFC_AMASK(64 * 1024)
#define CONFIG_SYS_FPGA_CSOR (CSOR_NOR_ADM_SHIFT(4) | \
CSOR_NOR_NOR_MODE_AVD_NOR | \
CSOR_NOR_TRHZ_80)
/* CPLD Timing parameters for IFC GPCM */
#define CONFIG_SYS_FPGA_FTIM0 (FTIM0_GPCM_TACSE(0xf) | \
FTIM0_GPCM_TEADC(0xf) | \
FTIM0_GPCM_TEAHC(0xf))
#define CONFIG_SYS_FPGA_FTIM1 (FTIM1_GPCM_TACO(0xff) | \
FTIM1_GPCM_TRAD(0x3f))
#define CONFIG_SYS_FPGA_FTIM2 (FTIM2_GPCM_TCS(0xf) | \
FTIM2_GPCM_TCH(0xf) | \
FTIM2_GPCM_TWP(0xff))
#define CONFIG_SYS_FPGA_FTIM3 0x0
#define CONFIG_SYS_CSPR1_EXT CONFIG_SYS_FPGA_CSPR_EXT
#define CONFIG_SYS_CSPR1 CONFIG_SYS_FPGA_CSPR
#define CONFIG_SYS_AMASK1 CONFIG_SYS_FPGA_AMASK
#define CONFIG_SYS_CSOR1 CONFIG_SYS_FPGA_CSOR
#define CONFIG_SYS_CS1_FTIM0 CONFIG_SYS_FPGA_FTIM0
#define CONFIG_SYS_CS1_FTIM1 CONFIG_SYS_FPGA_FTIM1
#define CONFIG_SYS_CS1_FTIM2 CONFIG_SYS_FPGA_FTIM2
#define CONFIG_SYS_CS1_FTIM3 CONFIG_SYS_FPGA_FTIM3
```

3.2.4.5.1.7 USB

LS1021AIOT has a single DW3 USB controller, which is assigned to the second core, by default. Use the command `make menuconfig` to reconfigure the U-Boot to assign it to other cores.

```
ARM architecture  --->
[*] Enable baremetal
[*] Enable USB for baremetal
(1) USB0 is assigned to core1
(1) USB Controller numbers
```

3.2.4.5.1.8 PCIe

LS1021AIOT has two PCIe controllers. By default, one is assigned to core0 and the other is assigned to core1. Use the `make menuconfig` command to reconfigure the U-Boot, in order to re-assign the cores.

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

3.2.4.5.1.9 FlexCAN

Assigning CAN3 to BareMetal

In BareMetal, the port is allocated through the `flexcan.c` file. The `flexcan.c` path is `industry-uboot/drivers/flexcan/flexcan.c`. In this file, you should define the following variables:

```
struct can_bittiming_t flexcan3_bittiming =
 CAN_BITTIM_INIT(CAN_500K);
```

Note: You also should set the bit timing and baud rate (500K) of the CAN port.

```
struct can_ctrlmode_t flexcan3_ctrlmode = {
.loopmode = 0, /* Indicates whether the loop mode is enabled */
.listenonly = 0, /* Indicates whether the only-listen mode is
 enabled */
.samples = 0,
.err_report = 1,
};
struct can_init_t flexcan3 = {
.canx = CAN3,   /* Specify CAN port  */
.bt = &flexcan3_bittiming,
.ctrlmode = &flexcan3_ctrlmode,
.reg_ctrl_default = 0,
.reg_esr = 0
};
```

Optional Parameters

- **CAN port**

```
#define CAN3      (struct can_module *)CAN3_BASE)
#define CAN4      (struct can_module *)CAN4_BASE)
```

- **Baud rate**

```
#define CAN_1000K 10
#define CAN_500K 20
#define CAN_250K 40
#define CAN_200K 50
#define CAN_125K 80
```

```
#define CAN_100K 100
#define CAN_50K 200
#define CAN_20K 500
#define CAN_10K 1000
#define CAN_5K 2000
```

Use the command `make menuconfig` for configuring the FlexCAN setting for LS1021A reference design boards, as shown below: .

```
Device Drivers   --->
CAN support   --->
[*] Support for Freescale FLEXCAN based chips
[*]   Support for canfestival
```

3.2.4.5.2 LS1028ARDB board

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

3.2.4.5.2.1 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 --->
[*] Enable baremetal
[*] Enable ENETC for baremetal
   (1)  Enetc1 is assigned to core1
   (1)  ENETC Controller numbers
```

3.2.4.5.2.2 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.

```
#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:

```
=> 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@1
Bus 2:  i2c@2000000->i2c-mux@77->i2c@3
   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..... .......
```

3.2.4.5.2.3 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 --->
   Misc commands --->
      [*] wavplayer
Device Drivers --->
   Sound support --->
      [*] Enable sound support
      [*]   Enable I2S support
      [*] Freescale sound
      [*] Freescale sgtl5000 audio codec
      [*] Freescale SAI module
```

Audio integration in BareMetal

For audio feature, we should add SAI and SGTL5000 drivers to BareMetal.

- Add SAI driver source code to the *drivers/sound* directory
- Add SGTL5000 driver source code to the *drivers/sound* directory
- Add sound device source code to the *drivers/sound* directory
- Add a command that can play wav files to the *cmd* directory
- Add support for SAI and sgtl5000 in LS1028ARDB dts file
  In fsl-ls1028a.dtsi file:

```
sai4: audio-controller@f130000 {
 #sound-dai-cells = <0>;
 compatible = "fsl,vf610-sai";
```

```
reg = <0x0 0xf130000 0x0 0x10000>;
interrupts = <GIC_SPI 83 IRQ_TYPE_LEVEL_HIGH>;
clocks = <&clockgen QORIQ_CLK_PLATFORM_PLL
      QORIQ_CLK_PLL_DIV(2)>,
  <&clockgen QORIQ_CLK_PLATFORM_PLL
      QORIQ_CLK_PLL_DIV(2)>,
  <&clockgen QORIQ_CLK_PLATFORM_PLL
      QORIQ_CLK_PLL_DIV(2)>,
  <&clockgen QORIQ_CLK_PLATFORM_PLL
      QORIQ_CLK_PLL_DIV(2)>;
clock-names = "bus", "mclk1", "mclk2", "mclk3";
dma-names = "tx", "rx";
dmas = <&edma0 1 10>,
   <&edma0 1 9>;
fsl,sai-asynchronous;
status = "disabled";
};
```

In fsl-ls1028a-rdb.dts file:

```
sound {
    compatible = "fsl,audio-sgtl5000";
    model = "ls1028a-sgtl5000";
    audio-cpu = <&sai4>;
    audio-codec = <&sgtl5000>;
    audio-routing =
      "LINE_IN", "Line In Jack",
      "MIC_IN", "Mic Jack",
      "Mic Jack", "Mic Bias",
      "Headphone Jack", "HP_OUT";
};
i2c@1 {
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x1>;
    sgtl5000: codec@a {
        #sound-dai-cells = <0>;
        compatible = "fsl,sgtl5000";
        reg = <0xa>;
        VDDA-supply = <1800>;
        VDDIO-supply = <1800>;
        sys_mclk = <25000000>;
        sclk-strength = <3>;
    };
};
&sai4 {
    status = "okay";
};
```

- Add all source code to the corresponding makefile file.
- Add new default configurations to ls1028ardb_sdcard_baremetal_defconfig file

3.2.4.5.3 LS1043ARDB or LS1046ARDB board

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

3.2.4.5.3.1 Linux DTS

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

3.2.4.5.3.2 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/ls1043ardb.h`.

```
#define CONFIG_SYS_DDR_SDRAM_SLAVE_SIZE (256 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_MASTER_SIZE (512 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_SHARE_RESERVE_SIZE (16 * 1024 *
 1024)
#define CONFIG_SYS_DDR_SDRAM_SHARE_SIZE \ ((256 * 1024 * 1024)
 - CONFIG_SYS_DDR_SDRAM_SHARE_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.

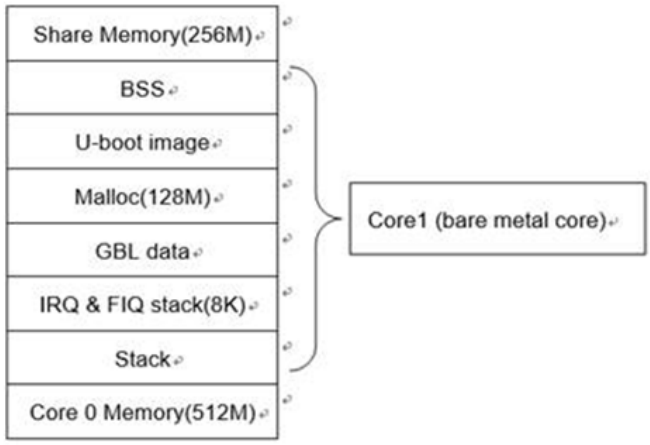


Figure 16. Memory configuration for LS1043ARDB or LS1046ARDB

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/ls1043a_common.h` to change the maximum size of malloc.

Table 22. Memory APIs description

API name (type)	Description
void_t* malloc (size_t n)	Allocates memory • “n” – length of allocated chunk • Returns a pointer to the newly allocated chunk

Table 22. Memory APIs description*...continued*

API name (type)	Description
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`.

3.2.4.5.3.3 GPIO

LS1043/6A has four GPIO controllers. You can add a GPIO node in the file ls1043/6a-rdb.dts to assign a GPIO resource to different cores. The configuration is in arch/arm/dts/fsl-ls1043/6a-rdb.dts. Use the command below to add a GPIO node:

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

3.2.4.5.3.4 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 `ls1043ardb_bm_defconfig` file using the commands below:

```
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.

3.2.4.5.3.5 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:

```
//register HW interrupt
void gic_irq_register(int irq_num, void (*irq_handle)(int));
void gic_set_target(u32 core_mask, unsigned long hw_irq);
void gic_set_type(unsigned long hw_irq);
```

3.2.4.5.3.6 QSPI

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

```
#define CONFIG_FSL_QSPI_COREID 1
```

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

3.2.4.5.3.7 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:

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

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

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

3.2.4.5.3.8 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`.

3.2.4.5.3.9 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
```

3.2.4.5.3.10 PCI 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
```

3.2.4.5.4 LX2160ARDB board

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

3.2.4.5.4.1 Memory configuration

This section describes the memory configuration for LX2160ARDB boards.

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

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

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

```
#define CONFIG_SYS_DDR_SDRAM_SLAVE_SIZE (64 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_MASTER_SIZE (512 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_SHARE_RESERVE_SIZE (16 * 1024 *
 1024)
#define CONFIG_SYS_DDR_SDRAM_SHARE_SIZE \ ((64 * 1024 * 1024) -
 CONFIG_SYS_DDR_SDRAM_SHARE_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_config.h to change the maximum size of malloc.

Table 23. Memory APIs description

API name (type)	Description
void_t* malloc (size_t n)	Allocates memory • “n” – length of allocated chunk • Returns a pointer to the newly allocated chunk

Table 23. Memory APIs description*...continued*

API name (type)	Description
void free (void *ptr)	Releases the chunk of memory pointed to by ptr (where “ptr” is a pointer to the chunk of memory)

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

3.2.4.5.5.1 Linux DTS

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

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

3.2.4.5.5.2 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_evk.h.` or `include/configs/imx8mp_evk.h.`

```
#define CONFIG_SYS_DDR_SDRAM_SLAVE_RESERVE_SIZE (SZ_32M)
#define CONFIG_SYS_DDR_SDRAM_SHARE_RESERVE_SIZE (SZ_4M)
#define CONFIG_SYS_DDR_SDRAM_SLAVE_SIZE (SZ_32M)
```

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 linux kernel, as shown in the following dts file:

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

3.2.4.5.5.3 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.

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

3.2.4.5.5.4 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:

```
&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.

3.3 Jailhouse

3.3.1 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.

3.3.2 Running PREEMPT_RT Linux in Inmate

3.3.2.1 i.MX 8M Plus LPDDR4 EVK

Perform the following steps;

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 has 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.
[    0.736473] igbvf: Intel(R) Gigabit Virtual Function
 Network Driver
[    0.736475] igbvf: Copyright (c) 2009 - 2012 Intel
 Corporation.
...
NXP Real-time Edge Distro 2.2 imx8mp-lpddr4-evk ttymxc3
```

```
imx8mp-lpddr4-evk login: root
root@imx8mp-lpddr4-evk:~#
root@imx8mp-lpddr4-evk:~# uname -a
Linux imx8mpevk 5.10.72-rt53-lts-5.10.y+g5304e5555731 #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.

3.3.2.2 LS1028ARDB

3.3.2.2.1 Linux in none-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
```

3.3.2.2.2 ENETC in none-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.

3.3.2.2.3 GPIO in none-root cell

GPIO3 controller is assigned to none-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 configure
   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 ouput.

   ```
   # 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 ouput.

   ```
   # echo 0 > /sys/class/gpio/gpio490/value
   # cat /sys/class/gpio/gpio490/value
   # cat /sys/class/gpio/gpio491/value
   ```

5. Exit Jailhouse

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

3.3.2.3 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:

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

4. Exit Jailhouse:

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

3.3.3 Running Jailhouse Examples In Inmate

3.3.3.1 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.

```
# 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:   2039 ns
Timer fired, jitter:    879 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    959 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:   1039 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:   1079 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    959 ns, min:    879 ns, max:   2039 ns
```

5. Execute UART demo:

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

6. Check the result on serial port:

```
Hello 1 from cell!
Hello 2 from cell!
Hello 3 from cell!
Hello 4 from cell!
Hello 5 from cell!
Hello 6 from cell!
Hello 7 from cell!
Hello 8 from cell!
Hello 9 from cell!
Hello 10 from cell!
Hello 11 from cell!
Hello 12 from cell!
Hello 13 from cell!
Hello 14 from cell!
Hello 15 from cell!
Hello 16 from cell!
Hello 17 from cell!
Hello 18 from cell!
Hello 19 from cell!
Hello 20 from cell!
```

7. Exit Jailhouse.

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

3.3.3.2 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:

```
IVSHMEM: Found device at 00:00.0
IVSHMEM: bar0 is at 0x00000000ff000000
IVSHMEM: bar1 is at 0x00000000ff001000
IVSHMEM: ID is 1
IVSHMEM: max. peers is 1
IVSHMEM: state table is at 0x00000000c0500000
IVSHMEM: R/W section is at 0x00000000c0501000
IVSHMEM: input sections start at 0x00000000c050a000
IVSHMEM: output section is at 0x00000000c050c000
IVSHMEM: initialized device
state[0] = 1
state[1] = 2
state[2] = 0
rw[0] = 1
rw[1] = 0
rw[2] = -1001599800
in@0x0000 = 10
in@0x2000 = 0
in@0x4000 = 1758252876

IVSHMEM: got interrupt 0 (#1)
state[0] = 1
state[1] = 2
state[2] = 0
rw[0] = 1
rw[1] = 1
rw[2] = -1001599800
in@0x0000 = 10
in@0x2000 = 10
in@0x4000 = 1758252876
```

6. Exit Jailhouse.

3.3.3.3 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
```

3.4 Harpoon (RTOS on Cortex-A)

3.4.1 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:

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

3.5 Heterogeneous Multicore Framework

3.5.1 Overview

Heterogenous multiprocessor systems on a chip are increasingly being used for real-time processing, achieving high performance, lowering cost, and to increase energy efficiency in industrial applications in recent years. In particular, the use of many different integrated processors running different operating systems poses multiple challenges. The heterogeneous AMP architecture has several drawbacks.

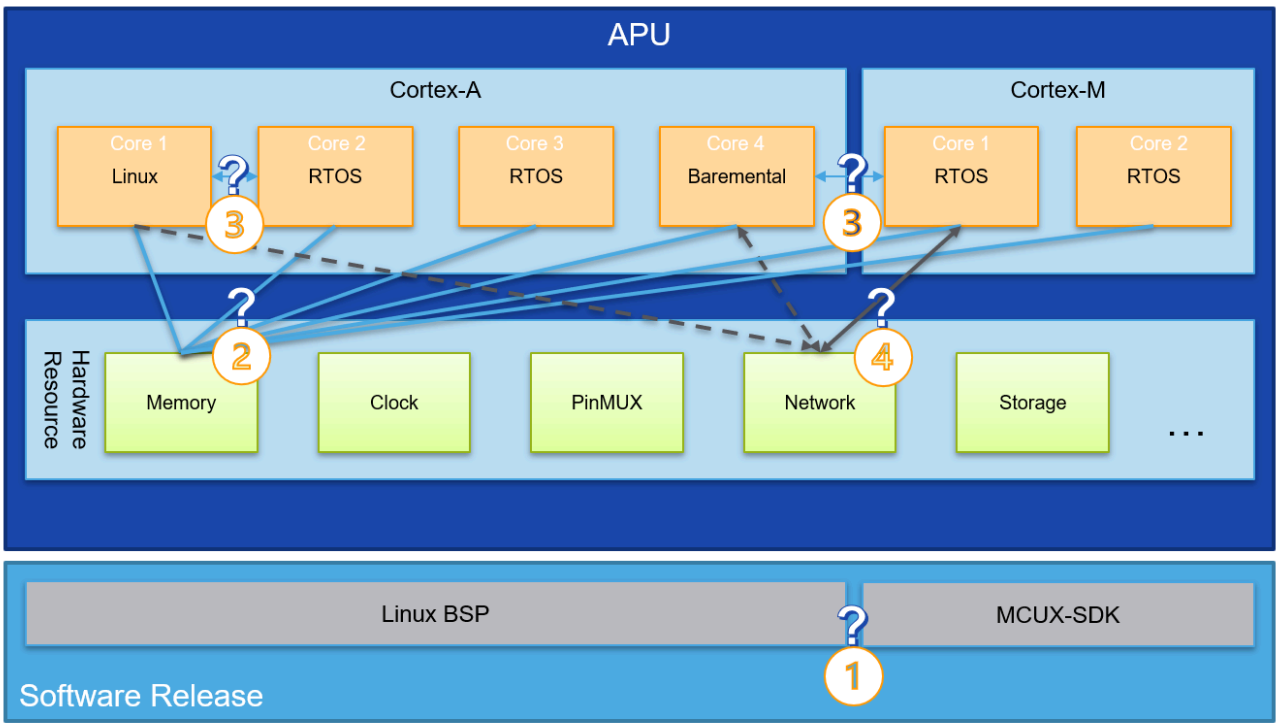


Figure 17. Drawbacks of using Heterogeneous AMP Software

1. No unified SW release
   a. Linux BSP focuses on Cortex-A core.
   b. MCUX-SDK focuses on Cortex-M core.
2. No common way to communicate between OS.
3. No way to share limited hardware resource between OS.
4. How to share network device to cross different OS?
   a. No unified way to assign and manage hardware resource across multiple operating systems.
   b. How to assign memory to a different OS on a different core?
   c. How to assign peripheral devices to a different OS?
   d. How to set clock pin mux to support the assignment?

Real-time Edge software supports that different systems such as Linux, FreeRTOS, Zephyr, and baremetal run on different processors. Real-time Edge software provides a total solution to support Heterogeneous AMP and tries to solve the above pain points.

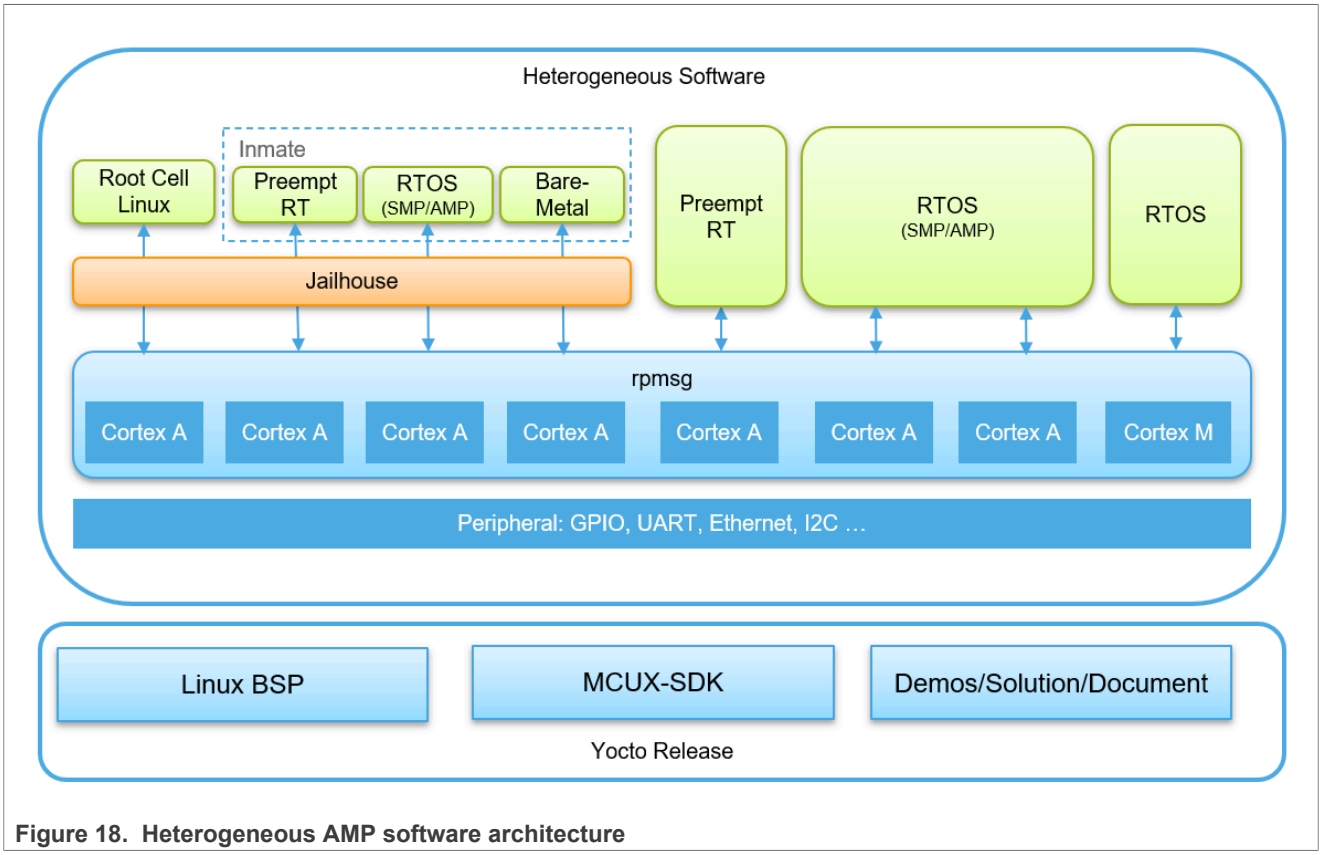


Figure 18. Heterogeneous AMP software architecture

1. Unified SW release
   - All different OS/application running on different cores can be built via Yocto.
   - A bitbake command can be used to create all images on different cores.
2. Communication between different OS
   - RPMSG is used to communicate between different OS
3. Resource sharing between different OS
   - Virtual device drivers
   - Unified APIs with physical device drivers
4. Unified resource allocation and management
   - Pinmux, memory, clock, peripherals

3.5.2 Yocto based unified delivery for Cortex-A and Cortex-M

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 building tool. Real-time Edge also selects Yocto as the unified SW release tool.

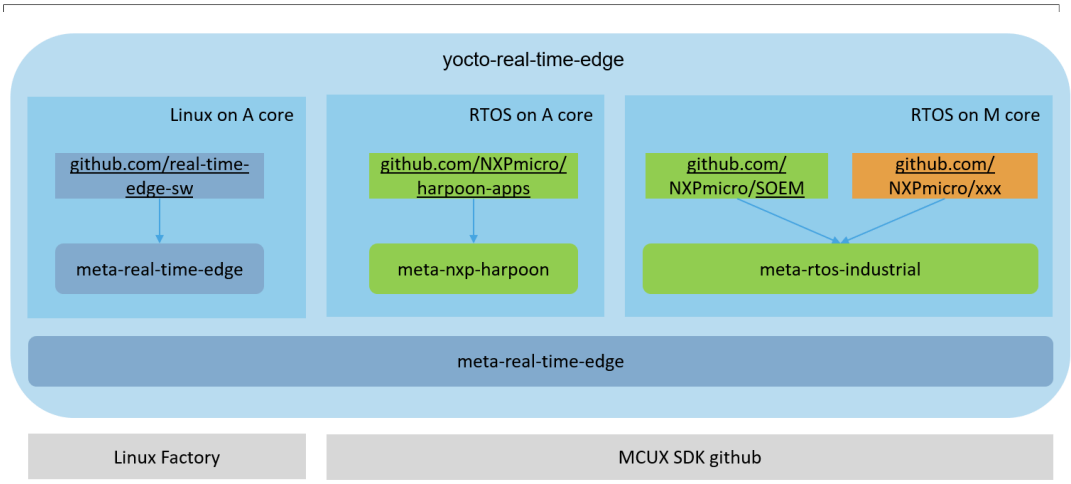


Figure 19. Unified Yocto Structure for Heterogeneous AMP

- 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 full image, including all binaries running on core A and core M.

For example:

```
# 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
```

3.5.3 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.

3.5.3.1 Introduction to meta-rtos-industrial

The following figure shows the `meta-rtos-industrial` file structure.

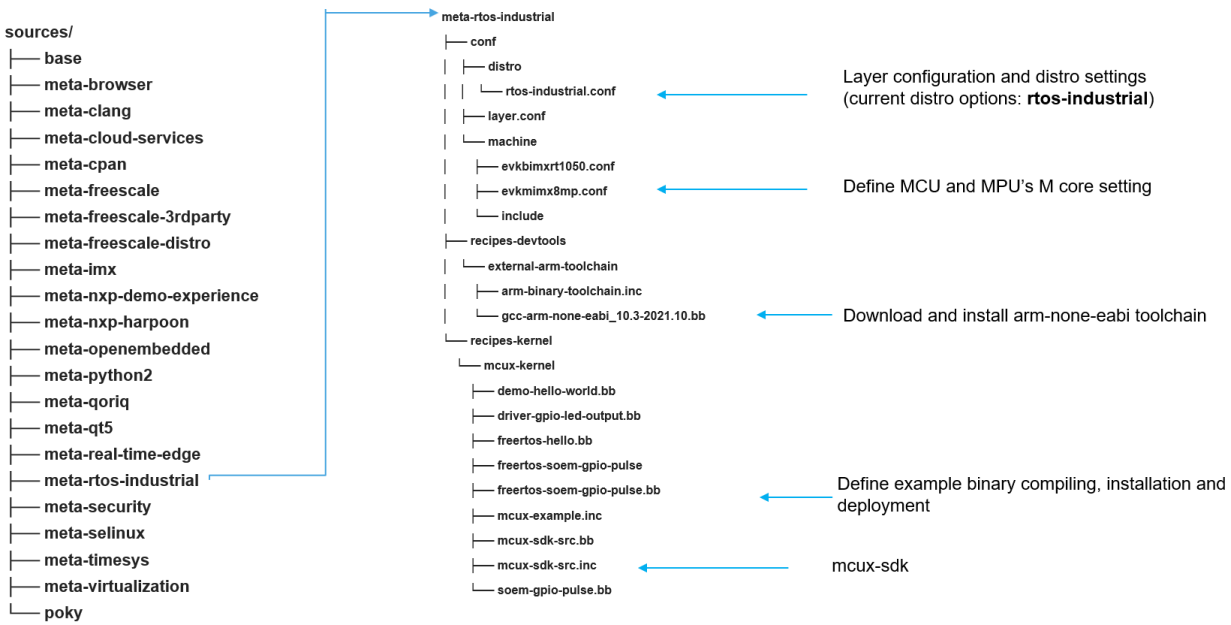


Figure 20. RTOS Industrial Layer Structure

3.5.3.1.1 source code definition

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

mcux-sdk-src.inc defines all repos of (NXPmicro/mcux-sdk: MCUXpresso SDK (github.com)) and the new repos.

If a new repo needs to download, please append the a new line to “SRC_URI” with the repo’s url and location.

For example(Download SOME stack to git/core/components/SOME folder):

“git://${NXPMICRO_BASE}/soem.git;protocol=https;nobranch=1;destsuffix=git/core/components/SOEM;name=SOME \”

mcux-sdk-src-XXX.inc is used to define the MCUX SDK repo commit ID for the release XXX.

For examples: mcux-sdk-src-2.11.0.inc contains all repos commit ID for the release 2.11.0.

The default version is defined by the parameter PREFERRED_VERSION_MCUX-SDK in mcux-sdk-src.inc.

If you want to compile different version, you can overwrite this parameter in local.conf.

For examples:

Add the below line into local.conf

PREFERRED_VERSION_MCUX-SDK = "2.10.0"

3.5.3.1.2 example definition

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. It's easy to add a new example.

For example:

include mcux-example.inc

MCUX_EXAMPLE_DIR = "examples/${RTOS-INDUSTRIAL-BOARD}/demo_apps/hello_world"

3.5.3.1.3 toolchain definition

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 support external toolchain. Parameter "ARMGCC_DIR" could be overwrote to point the external toolchain.

For example:

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

3.5.3.2 Integration of meta-rtos-industrial

To integrate `meta-rtos-industrial` into 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, you should use board name `evkmimx8mm` should be used instead of `imx8mm-lpddr4-evk` to compile Cortex-M application for i.MX 8M Mini EVK with LPDDR4. 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 rootfs.

3.5.3.3 Building meta-rtos-industrial

Since meta-rtos-industrial has been 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.

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

```
$ 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-kirkstone -m real-time-edge-2.4.0.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.

examples/

```
├── demo-hello-world
│   ├── ddr_release
│   │   ├── hello_world.bin
│   │   └── hello_world.elf
│   └── release
│   ├── hello_world.bin
│   └── hello_world.elf
├── driver-gpio-led-output
│   ├── ddr_release
│   │   ├── igpio_led_output.bin
│   │   └── igpio_led_output.elf
│   └── release
│   ├── igpio_led_output.bin
│   └── igpio_led_output.elf
├── freertos-hello
│   ├── ddr_release
│   │   ├── freertos_hello.bin
│   │   └── freertos_hello.elf
│   └── release
│   ├── freertos_hello.bin
│   └── freertos_hello.elf
├── freertos-soem-gpio-pulse
│   ├── ddr_release
│   │   ├── soem_gpio_pulse.bin
│   │   └── soem_gpio_pulse.elf
│   └── release
│   ├── soem_gpio_pulse.bin
│   └── soem_gpio_pulse.elf
├── rpmsg-lite-uart-sharing-rtos
│   └── release
│   ├── rpmsg_lite_uart_sharing_rtos.bin
│   └── rpmsg_lite_uart_sharing_rtos.elf
```

```
└── soem-gpio-pulse
├── ddr_release
│   ├── soem_gpio_pulse.bin
│   └── soem_gpio_pulse.elf
└── release
├── soem_gpio_pulse.bin
└── soem_gpio_pulse.elf
```

If you just want to compile a special example, you can use the following command:

For example:

```
$ DISTRO=nxp-real-time-edge MACHINE=imx8mm-lpddr4-evk bitbake
 rpmsg_lite_uart_sharing_rtos
```

3.5.4 Yocto layer for Cortex-A core

On Cortex-A core, you can run Linux, Jailhouse, Baremetal, and RTOS. Here is the corresponding Yocto layer description.

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 3.2 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 hypervisor 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, please refer to Harpoon User's Guide.

3.5.5 Resource sharing

3.5.5.1 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.

3.5.5.2 Software architecture and design

This chapter describes different software architectures based on different technologies.

3.5.5.3 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.

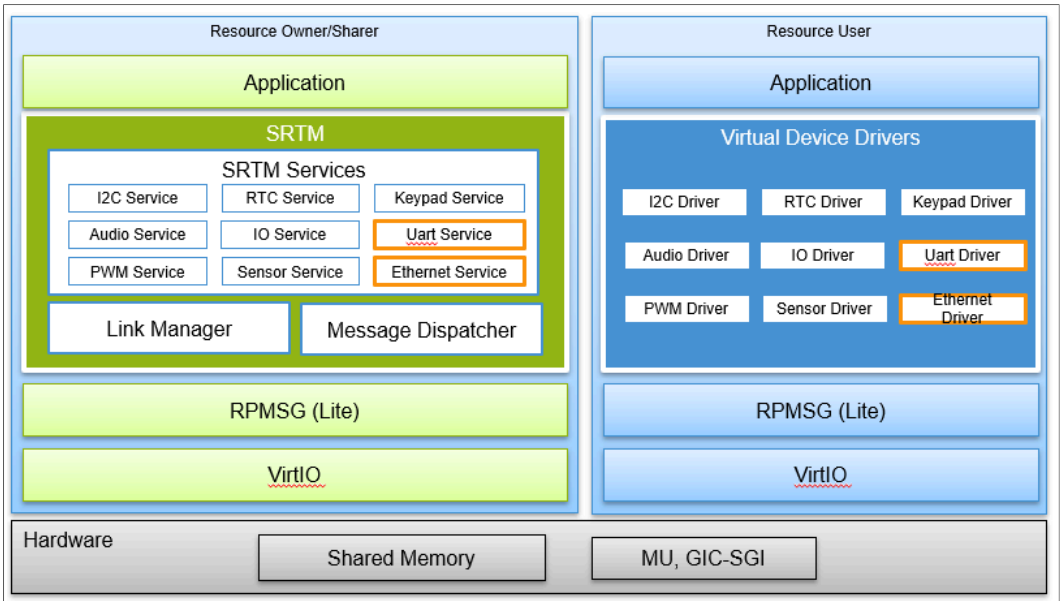


Figure 21. Resource Sharing Software Architecture

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.

3.5.5.3.1 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 also map to one same physical UART.

Supported Platforms: **i.MX 8M Mini.**

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.

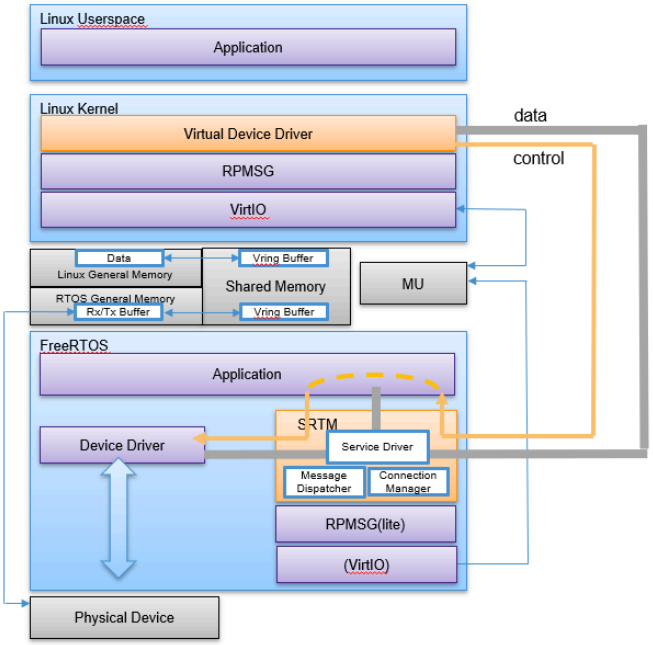


Figure 22. UART Sharing Software Architecture

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 24. Packet Format

Fields	Start Flags (4 bytes)				Address (1 bytes)	Payload Size (1 bytes)	Payload (n bytes)
HEX	24	55	54	2C	x	n	xxxxxx…
ASCII	$	U	T	,			

Packet header includes fields of start flags, address, 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,”.

“Address” field is reused by receive an transmit directions. For receive direction (blue colored path in Figure 7), it is destination address or ID of target virtual device; for transmit direction (orange colored path in Figure 7), it is 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 data are followed packet header.

3.5.5.4 Source code files and configuration

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

Table 25. Software Source Code List

Name	Software Component	Source Files/Directory
FreeRTOS application: `rpmsg_lite_uart_sharing_rtos`	`mcux-sdk-examples`	`evkmimx8mm/multicore_examples/rpmsg_lite_uart_sharing_rtos/`
SRTM Service	`mcu-sdk`	`components/srtm/services/srtm_uart_service.c` `srtm_uart_service.h` `srtm_uart_adapter.c` `srtm_uart_adapter.h`
Virtual UART driver	`real-time-edge-linux`	`drivers/tty/rpmsg_tty.c`

2. **Linux Virtual UART driver**
   By default, Real-time Edge kernel will build virtual UART driver as module by enable 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 RPMSG Endpoint.

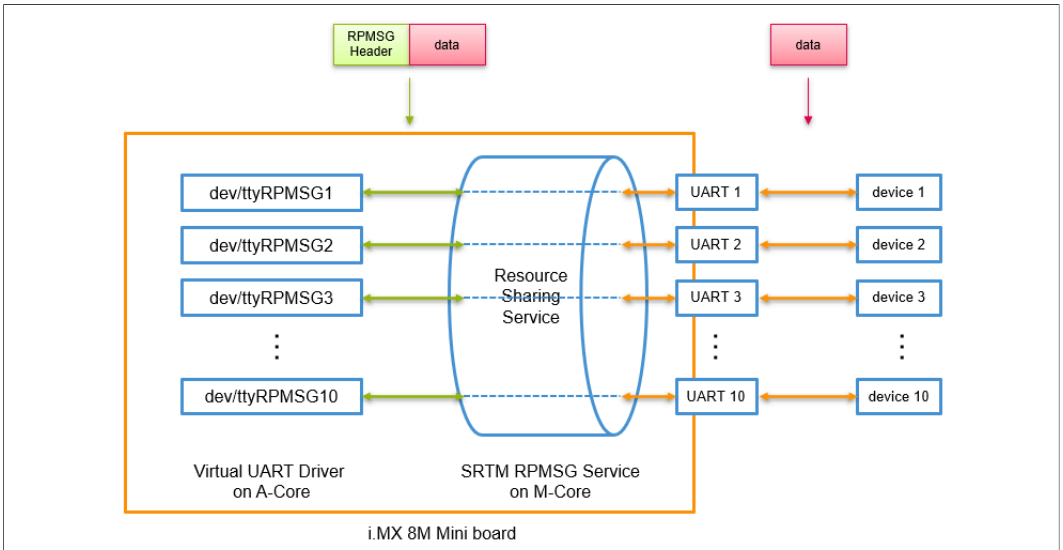


Figure 23. Virtual UART to physical UART 1:1 mapping

   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 (find more details about multiple UART Header in chapter “UART Sharing Design Details”)

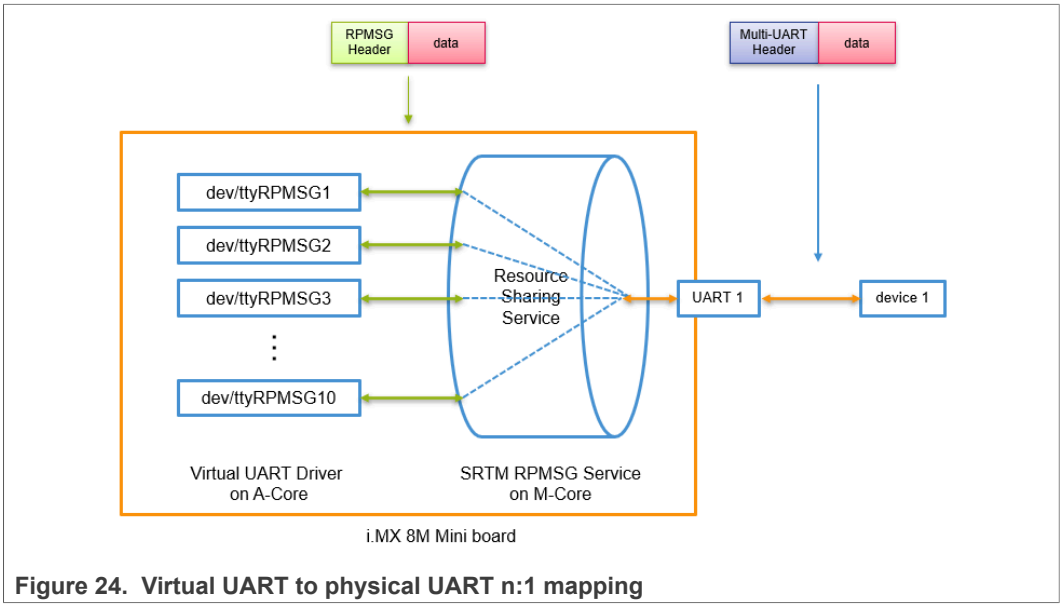


Figure 24. Virtual UART to physical UART n:1 mapping

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.

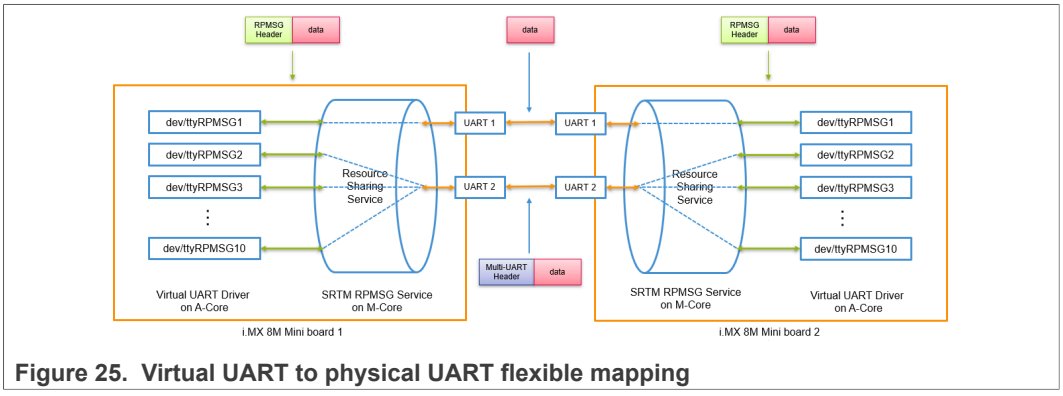


Figure 25. Virtual UART to physical UART flexible mapping

The mapping between virtual UART and physical UART is configured in Linux device tree, as shown in a dts node example below:

```
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`” is used to specify the physical UART instance ID that this virtual UART will map to.*
- *Physical UART ID is configured in the FreeRTOS application “`rpmsg_lite_uart_sharing_rtos`”.*

Currently only physical UART3 can be used, so all virtual UART ports are mapping to physical UART3 by default. If property of "`bus_id`" is not configured, the message sent from Linux to this virtual UART is display on M-Core's debug console directly. 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 default `dtb` file "`imx8mm-evk-rpmsg.dtb`". The virtual UART "`0`" to "`9`" are n:1 mapped to physical UART3, virtual UART "`10`" has no "`bus_id`" and displays messages sent from Linux to M-Core's debug console.

3.5.5.5 Building and running the demo

3.5.5.5.1 Hardware setup

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 26. PIN connection between two i.MX 8M Mini boards

i.MX 8M Mini Board1		Connection	i.MX 8M Mini Board2	
Pin	Function		Pin	Function
6	GND	<->	6	GND
8	UART3_TXD	<->	10	UART3_RXD
10	UART3_RXD	<->	8	UART3_TXD

3.5.5.5.2 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 "`/example`" 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.

3.5.5.5.3 Running the 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/rpmsg-lite-
uart-sharing-rtos/release/rpmsg_lite_uart_sharing_rtos.bin;
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. And then boot Linux kernel by executing the following command:

```
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, execute the following on the first board:

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

On the second board, execute the following command:

```
root@imx8mm-lpddr4-evk:~# echo "this is from virtual uart
 #6" > /dev/ttyRPMSG6
```

Then, the first board receives and displays the message "`this is from virtual uart #6`".

3.5.6 RPMSG communication for Heterogenous AMP

3.5.6.1 Overview

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

Currently Real-time Edge support the following Heterogenous AMP:

- Linux on Cortex-A Core(s)
- RTOS on Cortex-M Core
- RTOS on Cortex-A Core(s)

Between these OS running on different process, Real-time Edge support inter-core communications between Cortex-M Core and Cortex-A Core, and also support RPMSG between heterogenous AMP on different Cortex-A Cores.

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

The following diagram shows RPMSG communication between RTOS running on Cortex-M Core and Linux running Cortex-A Core.

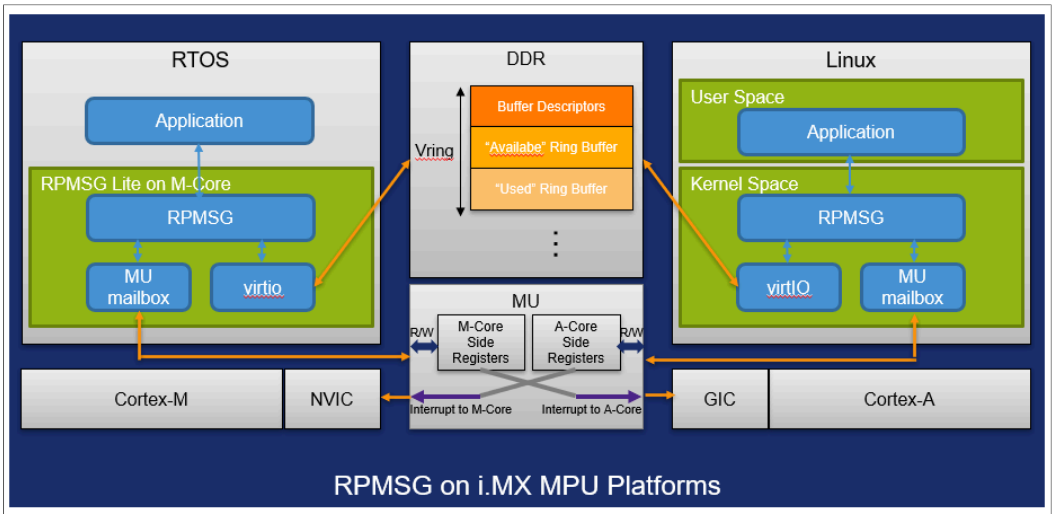


Figure 26. RPMSG between Cortex-A Core and Cortex-M Core

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. RPMsg-Lite is used on RTOS, it includes VirtIO driver, mailbox driver, and RPMSG driver. The RPMsg-Lite is an open-source component developed by NXP Semiconductors, it is a lightweight implementation of the RPMSG protocol, details can be found at RPMsg-Lite User's Guide

3.5.6.3 RPMSG between heterogenous AMP on different Cortex-A cores

The following diagram illustrates the software setup for RPMSG between Heterogenous AMP on different Cortex-A Cores.

```
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:   2039 ns
Timer fired, jitter:    879 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    959 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:   1039 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:   1079 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    959 ns, min:    879 ns, max:   2039 ns
```

There is no MU hardware mailbox can be used between different Cortex-A Cores, so a SGI virtual mailbox is created for message notification between Cortex-A Cores. SGI virtual mailbox uses DDR shared memory to simulate some MMIO registers to be used by mailbox driver. SGI interrupts in GIC are used as notification interrupts between Cortex-A cores.

RPMsg-Lite is also enabled on Cortex-A Core's RTOS.

3.5.6.3.1 Build and running the RPMSG demo

Real-time Edge software provides a demo to evaluate RPMSG communication between heterogenous AMP on different Cortex-A Cores.

The demo creates a virtual tty device (`/dev/ttyRPMSG30`) on a Linux machine that runs on Cortex-A Core. The string sent to this tty device is sent to RTOS (running on FreeRTOS). This string is then sent back from FreeRTOS to Linux, and finally it is received by this virtual tty device in Linux. The simple testing method uses an open virtual tty device along with a console software, such as minicom. The test is successful if the console echoes the string that was used as input.

3.5.6.3.2 Building the RPMSG demo

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.

The bellow command is used to compile the demo separately.

```
bitbake rpmsg-lite-str-echo-rtos
```

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

3.5.6.3.3 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/ rpmsg-lite-str-echo-
rtos/ddr_release/rpmsg_lite_str_echo_rtos.bin
                => dcache off; dcache flush; icache flush;
 icache off
                => mw 303d0518 f 1
                => 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` file.

```
root@imx8mm-lpddr4-evk:~# modprobe imx_rpmsg_tty
```

5. Use minicom to open a console using the command `ttyRPMSG30` on the Linux prompt as shown below:

```
root@imx8mm-lpddr4-evk:~# minicom -D /dev/ttyRPMSG30
```

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

3.6 Booting Cortex-M Core RTOS Image from Linux

On i.MX 8M Plus EVK and i.MX 8M Mini EVK platforms, there are two ways to boot ARM Cortex-M Core. These are described in the following sections.

3.6.1 Booting from U-Boot command line

After the board is booted into the U-Boot console, use the following command to boot Arm Cortex-M core:

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

3.6.2 Using remoteproc to boot from Linux command line

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:

```
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
```

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

4 Real-time Networking

4.1 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 Plus, LS1028ARDB, and LS1021A-TSN boards.

4.1.1 TSN hardware capability

Table 27. TSN hardware capability on different platforms

Platform	802.1Qbv (Enhancements for Scheduled Traffic)	802.1Qbu and 802.3br (Frame Preemption)	802.1Qav (Credit Based Shaper)	802.1AS (Precision Time Protocol)	802.1CB (Frame Replication and Elimination for Reliability)	802.1Qci (Per Stream Filtering and Policing)
ENETC (LS1028a)	Y	Y	Y	Y	N	Y
Felix switch (LS1028a)	Y	Y	Y	Y	Y	Y
SJA1105 (LS1021a-TSN)	Y	N	Y	Y	N	Pre-standard
Stmac (i.MX 8M Plus)	Y	Y	Y	Y	N	N

4.1.2 TSN configuration

The table below describes the TSN configuration tools support on different platforms

Table 28. TSN configuration tools support on different platforms

Platform	802.1Qbv (Enhancements for Scheduled Traffic)	802.1Qbu and 802.3br (Frame Preemption)	802.1Qav (Credit Based Shaper)	802.1AS (Precision Time Protocol)	802.1CB (Frame Replication and Elimination for Reliability)	802.1Qci (Per Stream Filtering and Policing)
ENETC (LS1028A)	tc-taprio tsntool	ethtool tsntool	tc-cbs tsntool	ptp4l	N/A	tc-flower tsntool
Felix switch (LS1028A)	tc-taprio tsntool	ethtool tsntool	tc-cbs tsntool	ptp4l, Gen AVB/TSN stack	tsntool	tc-flower tsntool
SJA1105 (LS1021A-TSN)	tc-taprio	N/A	tc-cbs	ptp4l	N/A	tc-flower
Stmac (i.MX 8M Plus)	tc-taprio	ethtool	tc-cbs	ptp4l, Gen AVB/TSN stack	N/A	N/A

4.1.2.1 Using Linux traffic control (tc)

Enable the following configurations in kernel when using Linux traffic control (tc):

```
Symbol: NET_SCH_MQPRIO [=y] && NET_SCH_CBS [=y] &&
 NET_SCH_TAPRIO [=y]
  [*] 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
        <*>   SKB 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:

https://man7.org/linux/man-pages/man8/tc-taprio.8.html

2. The below link provides details for using tc-cbs to set Qav:

https://man7.org/linux/man-pages/man8/tc-cbs.8.html

3. The below link provides details for using tc-flower to set Qci and ACL:

https://man7.org/linux/man-pages/man8/tc-flower.8.html

4.1.2.2 Tsntool

Tsntool is a tool to set the TSN capability of the Ethernet ports of TSN Endpoint and TSN switch. It's used on LS1028a platform, so 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 MDIO driver
        <*>     ENETC PTP clock driver
        [*]     ENETC hardware timestamping support
        [*]     TSN Support for NXP ENETC driver
Symbol: MSCC_FELIX_SWITCH_TSN [=y]
  Device Drivers --->
    [*] Network device support --->
      Distributed Switch Architecture drivers --->
        <*> Ocelot / Felix Ethernet switch support --->
        <*>   TSN on FELIX switch driver
```

Enable PKTGEN in Kernel to use pktgen for testing,

```
Symbol: NET_PKTGEN [=y]
  [*] Networking support --->
    Networking options --->
      Network testing --->
        <*> Packet Generator (USE WITH CAUTION)
```

See "Tsntool User Manual" for the details.

4.1.2.2.1 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.*

4.1.2.2.1.1 Getting the source code

Github of the tsntool code is mentioned below.

https://source.codeaurora.org/external/qoriq/qoriq-components/tsntool/

4.1.2.2.1.2 Tsn tool commands

The following table lists the TSN tool commands and their description.

Table 29. TSN tool commands and their description

Command	Description
help	Lists commands support
version	Shows software version
verbose	Debugs on/off for tsntool
quit	Quits prompt mode
qbvset	Sets time gate scheduling config for <ifname>
qbvget	Gets time scheduling entries for <ifname>
cbstreamidset	Sets stream identification table
cbstreamidget	Gets stream identification table and counters
qcisfiset	Sets stream filter instance
qcisfiget	Gets stream filter instance
qcisgiset	Sets stream gate instance
qcisgiget	Gets stream gate instance
qcisficounterget	Gets stream filter counters
qcifmiset	Sets flow metering instance
qcifmiget	Gets flow metering instance
cbsset	Sets TCs credit-based shaper configure
cbsget	Gets TCs credit-based shaper status
qbuset	Sets one 8-bits vector showing the preemptable traffic class
qbugetstatus	Not supported
tsdset	Not supported
tsdget	Not supported
ctset	Sets cut through queue status (specific for ls1028 switch)
cbgen	Sets sequence generate configure (specific for ls1028 switch)
cbrec	Sets sequence recover configure (specific for ls1028 switch)
dscpset	Sets queues map to DSCP of Qos tag (specific for ls1028 switch)
sendpkt	Not supported
regtool	Registers read/write of bar0 of PFs (specific for ls1028 enetc)
ptptool	ptptool get/set ptp timestamp. Useful commands: `#get ptp0 clock time` `ptptool -g` `#get ptp1 clock time` `ptptool -g -d /dev/ptp1`
dscpset	Set queues map to DSCP of QoS tag (specific for ls1028 switch)
qcicapget	Gets max capability of the qci instance
tsncapget	Gets tsn capability of the device

4.1.2.2.1.3 Tsntool commands and parameters

This section lists the tsntool commands along with the parameters and arguments, with which they can be used.

Table 30. qbvset

Parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`.
`--entryfile <filename>`	A file script to input gatelist format. It has the following arguments: `#'NUMBER' 'GATE_VALUE' 'TIME_LONG'` • `NUMBER`: # 't' or 'T' head. Plus entry number. Duplicate entry number will result in an error. • `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. • `TIME_LONG`: # nanoseconds. Do not input 0 time long. `t0 11101111b 10000 t1 11011111b 10000` ***Note:*** *`Entryfile` parameter must be set. If not set, there will be a vi text editor prompt, "`require to input the gate list`".*
`--basetime <value>`	AdminBaseTime 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.
`--cycletime <value>`	AdminCycleTime
`--cycleextend <value>`	AdminCycleTimeExtension
`--enable | --disable`	• `enable`: enables the `qbv` for this port. • `disable`: disables the `qbv` for this port. By default, the value is set to `enable`, if user does not provide any input.
`--maxsdu <value>`	queueMaxSDU
`--initgate <value>`	AdminGateStates
`--configchange`	ConfigChange. Default set to 1.
`--configchangetime <value>`	ConfigChangeTime

Table 31. qbvget

Parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`

Table 32. cbstreamidset

Parameter <argument>	Description
`--enable | --disable`	• enable: Enables the entry for this index. • disable: Disables the entry for this index. By default, this field is set to `enable` if there is no enable or disable input.

Table 32. cbstreamidset *...continued*

Parameter <argument>	Description
`--index <value>`	Index entry number in this controller. Mandatory parameter. This value corresponds to `tsnStreamIdHandle` on switch configuration.
`--device <string>`	An interface such as `eno0/swp0`
`--streamhandle <value>`	`tsnStreamIdHandle`
`--infacoutport <value>`	`tsnStreamIdInFacOutputPortList`
`--outfacoutport <value>`	`tsnStreamIdOutFacOutputPortList`
`--infacinport <value>`	`tsnStreamIdInFacInputPortList`
`--outfacinport <value>`	`tsnStreamIdOutFacInputPortList`
`--nullstreamid | --sourcemacvid | --destmacvid | --ipstreamid`	`tsnStreamIdIdentificationType`: • `-nullstreamid`:Null Stream identification • `-sourcemacvid`: Source MAC and VLAN Stream identification • `-destmacvid`: not supported • `-ipstreamid`: not supported
`--nulldmac <value>`	`tsnCpeNullDownDestMac`
`--nulltagged <value>`	`tsnCpeNullDownTagged`
`--nullvid <value>`	`tsnCpeNullDownVlan`
`--sourcemac <value>`	`tsnCpeSmacVlanDownSrcMac`
`--sourcetagged <value>`	`tsnCpeSmacVlanDownTagged`
`--sourcevid <value>`	`tsnCpeSmacVlanDownVlan`

Table 33. cbstreamidget

Parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`
`--index <value>`	Index entry number in this controller. Mandatory to have.

Table 34. qcisfiset

Parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`
`--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.

Table 34. qcisfiset *...continued*

Parameter <argument>	Description
`--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`
`--oversizeenable`	`StreamBlockedDueToOversizeFrameEnable`
`--oversize`	`StreamBlockedDueToOversizeFrame`

Table 35. qcisfiget

parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`
`--index <value>`	Index entry number in this controller. Mandatory to have.

Table 36. qcisgiset

Parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`
`--index <value>`	Index entry number in this controller. Mandatory to have.
`--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.

Table 36. qcisgiset *...continued*

Parameter <argument>	Description
--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 37. qcisgiget

Parameter <argument>	Description
--device <ifname>	An interface such as eno0/swp0
--index <value>	Index entry number in this controller. Mandatory to have.

Table 38. qcifmiset

Parameter <argument>	Description
--device <ifname>	An interface such as eno0/swp0
--index <value>	Index entry number in this controller. Mandatory to have.
--disable	If not set disable, then to be set enable.
--cir <value>	cir. kbit/s.
--cbs <value>	cbs. octets.
--eir <value>	eir.kbit/s.
--ebs <value>	ebs.octets.
--cf	cf. couple flag.
--cm	cm. color mode.
--dropyellow	drop yellow.
--markred_enable	mark red enable.
--markred	mark red.

Table 39. qcifmiget parameter

Parameter <argument>	Description
--device <ifname>	An interface such as eno0/swp0
--index <value>	Index entry number in this controller. Mandatory to have.

Table 40. qbuset parameter

Parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`
`--preemptable <value>`	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.

Table 41. cbsset command

Parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`
`--tc <value>`	Traffic class number.
`--percentage <value>`	Set percentage of tc limitation.
`--all <tc-percent:tc-percent...>`	Not supported.

Table 42. cbsget

Parameter <argument>	Description
`--device <ifname>`	An interface such as `eno0/swp0`
`--tc <value>`	Traffic class number.

Table 43. regtool

Parameter <argument>	Description
`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 44. ctset

Parameter <argument>	Description
`--device <ifname>`	An interface such as `swp0`
`--queue_stat <value>`	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.

Table 45. cbgen

Parameter <argument>	Description
`--device <ifname>`	An interface such as `swp0`
`--index <value>`	Index entry number in this controller. Mandatory to have. This value corresponds to `tsnStreamIdHandle` of `cbstreamidset` command.

Table 45. cbgen*...continued*

Parameter <argument>	Description
`--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. `tsnSeqGenSpace = 2**SEQ_SPACE_LOG2` For example, if this value is 12, then valid sequence numbers are from 0x0 to 0xFFF.
`--seq_num <value>`	GEN_REC_SEQ_NUM: The sequence number to be used for outgoing packet passed to `SEQ_GEN` function. Note: Only lower 16-bits are sent in RED_TAG.

Table 46. cbrec

Parameter <argument>	Description
`--device <ifname>`	An interface such as `swp0`
`--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. `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 47. dscpset

Parameter <argument>	Description
`--device <ifname>`	An interface such as `swp0`
`--disable`	Disables DSCP to traffic class for frames.
`--index`	DSCP value
`--cos`	Priority number of queue which is mapped to
`--dpl`	Drop level which is mapped to

Table 48. qcicapget

Parameter <argument>	Description
`--device <ifname>`	An interface such as `swp0`

Table 49. tsncapget

Parameter <argument>	Description
`--device <ifname>`	An interface such as `swp0`

4.1.2.2.1.4 Input tips

While providing the command input, user can use the following shortcut keys to make the input faster:

- When 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 user input parameters, if user does not remember the parameter name. User can just input "--" then press **TAB** key. It displays all the parameters.
  If user inputs 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.

4.1.2.2.1.5 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
```

4.1.2.3 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**

TSN offload	Real-time Edge			
	LS1028		SJA1105	i.MX 8M Plus
	libtsn	tc	tc	tc
802.1Qbv (Time Aware Shaper)	Y	Y	Y	Y
802.1Qbu/802.3br (Frame Preemption)	Y	Y	N/A	Y
802.1Qav (Credit Based Shaper)	-	-	-	-
802.1CB (Frame Replication and Elimination for Reliability)	-	-	N/A	N/A
802.1Qci (Per-Stream Filtering and Policing)	Y	Y	Y	N/A
IP config	Y	Y	Y	Y
MAC config	Y	Y	Y	Y
VLAN config	Y	Y	Y	Y

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.

4.1.2.4 Web-based configuration

4.1.2.4.1 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 want to setup the web server step by step, just follow 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 libxslt1-dev libcurl4-openssl-dev
 xsltproc python-setuptools
$ sudo apt install -y zlib1g-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 python3-pip python3-dev python3-flask
 python3-pexpect
$ 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. Install libnetconf:

```
$ git clone https://github.com/CESNET/libnetconf.git
$ cd libnetconf
$ git checkout 8e934324e4b1e0ba6077b537e55636e1d7c85aed -b
 fornetconf1
$ autoreconf --force --install
$ ./configure
$ make
$ sudo make install
```

5. Get tsntool source code on the web server PC:

```
git clone https://source.codeaurora.org/external/qoriq/qoriq-
components/tsntool
cd tsntool/demos/cnc/
```

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:

```
#Stop lldpd service.
pkill lldpd
#Start lldpd and limit interfaces to use. Use all ports
 except the control port.
lldpd -I 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.*

4.1.2.4.2 Remote configuration

The below section describes the steps for remote configuration.

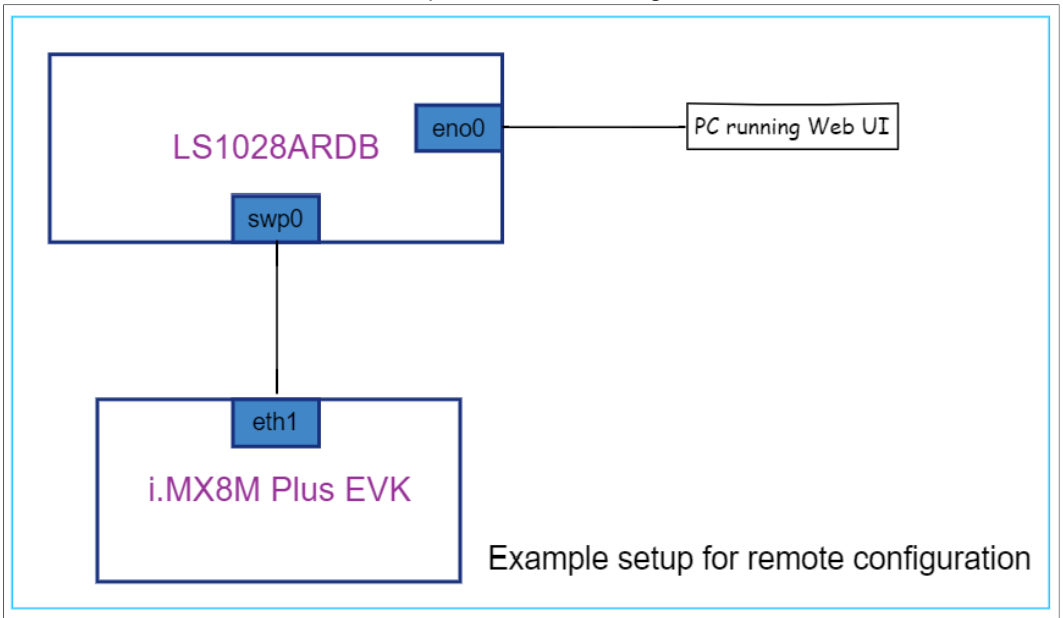


Figure 27. A sample setup for remote configuration

1. Overview

The Web UI allows the remote control of the YANG model. The user can connect http server, and input TSN parameter on web UI, and click "Yes, confirm" button to send them to the board.

2. User Interface

Click the device displayed on the homepage, and an interface description table will appear. Click the interface to jump to the configuration page.

2.1 Qbv Configuration

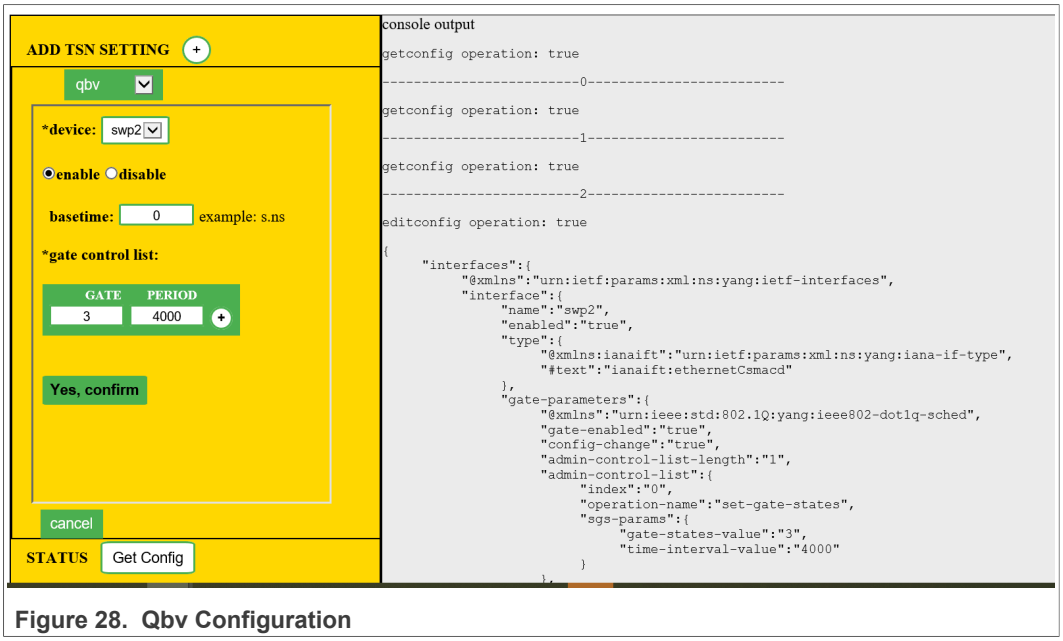


Figure 28. Qbv Configuration

2.2 Qbu Configuration

Figure 29. Qbu Configuration

2.3 Qci Configuration

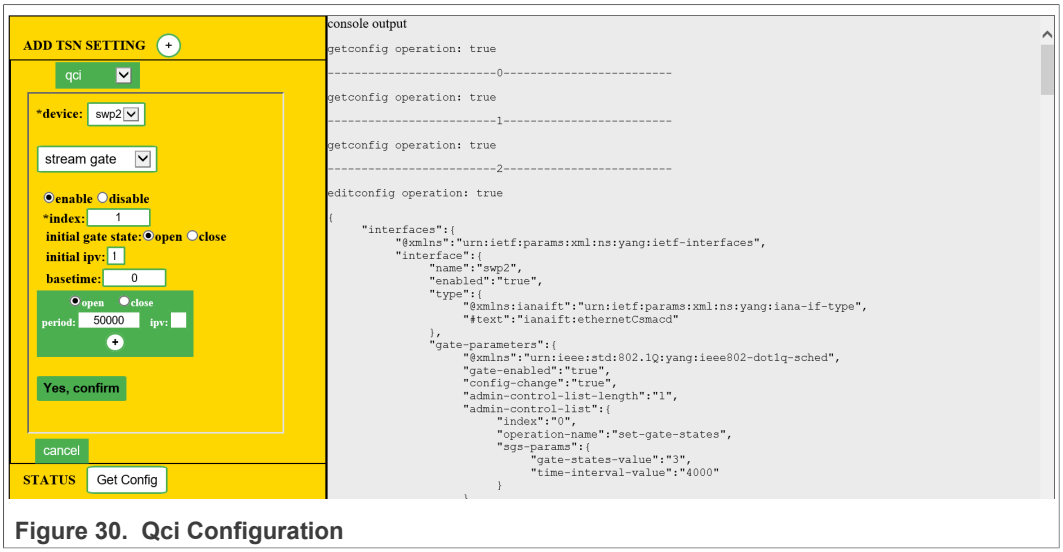


Figure 30. Qci Configuration

In this interface, user can choose configuration for "stream identify", "stream filter", "stream gate", and "flow metering".

4.1.2.4.3 Dynamic remote configuration

1. Overview

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.

2. Working Flow

Here is an example of the TSN configuration working flow:

After topology discovery and device registration, the network topology could be displayed over web-browser. The user just needs to select the nodes, specify the stream, and 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 be 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

Here is the architecture diagram.

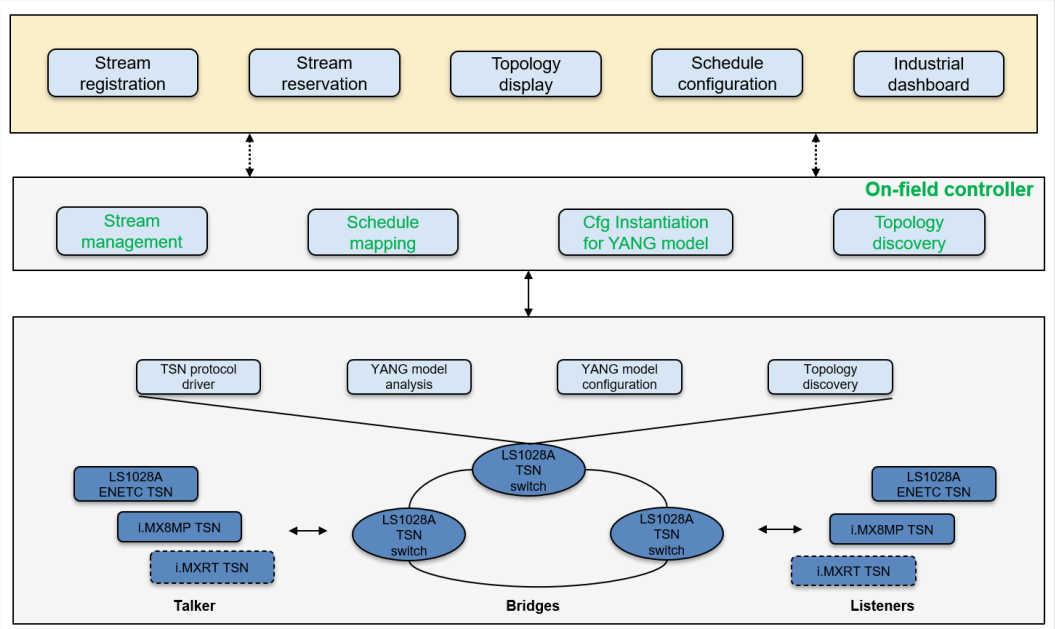


Figure 31. Architecture diagram for TSN

There are three layers for the architecture. The first layer is the TSN network layer, the second layer is the service layer running on the on-field controller/server, and the third layer is the service running in the cloud or on-field server which is an optional layer.

The first layer is the TSN network layer. It includes TSN switches, like LS1028 TSN switch and TSN endpoints, like LS1028 ENETC TSN, i.MX 8M Plus TSN endpoint, to be the TSN network. The different components are running on each of nodes, like the topology discovery component, to collect the network topology, YANG model for the TSN register configuration, and NETCONF server to parse 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.

3. 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-filed server.

4. Path Selection

Path selection implemented 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.

5. Schedule Mapping

The schedule mapping component is a critical component to convert the customer requirement to TSN register configuration. This component will:

- Get the user input and converting the input into TSN parameters.

- Get the path and path delay from the link object of the NetworkGraph file.
- Get the old TSN configuration for each node and calculate a new configuration to meet the user's requirements.

6. Path Delay

One prerequisite for schedule mapping is clock synchronization and path delay calculation. Clock synchronization is using gPTP to synchronize the clock of the system. We are using linuxptp PMC tool to get the path delay.

Here is an example to show the PMC running environment on LS1028ARDB boards.

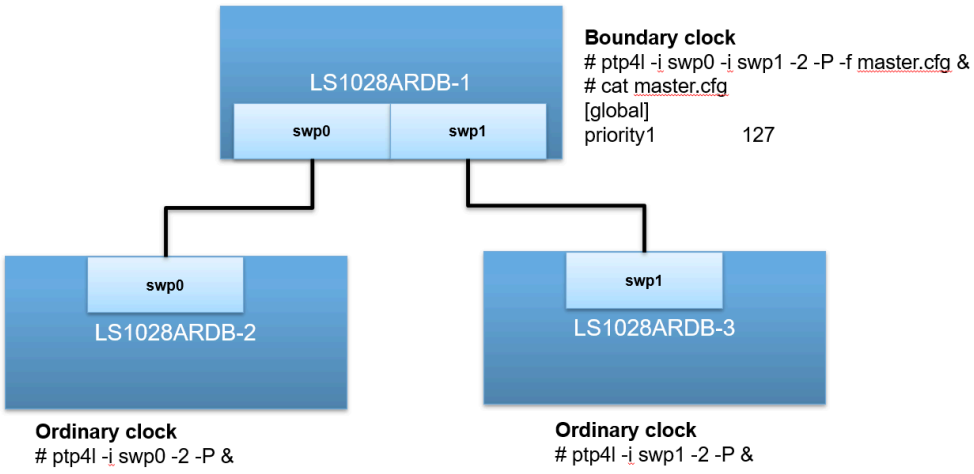


Figure 32. PMC running environment on LS1028ARDB boards

7. Dashboard Configuration Demo

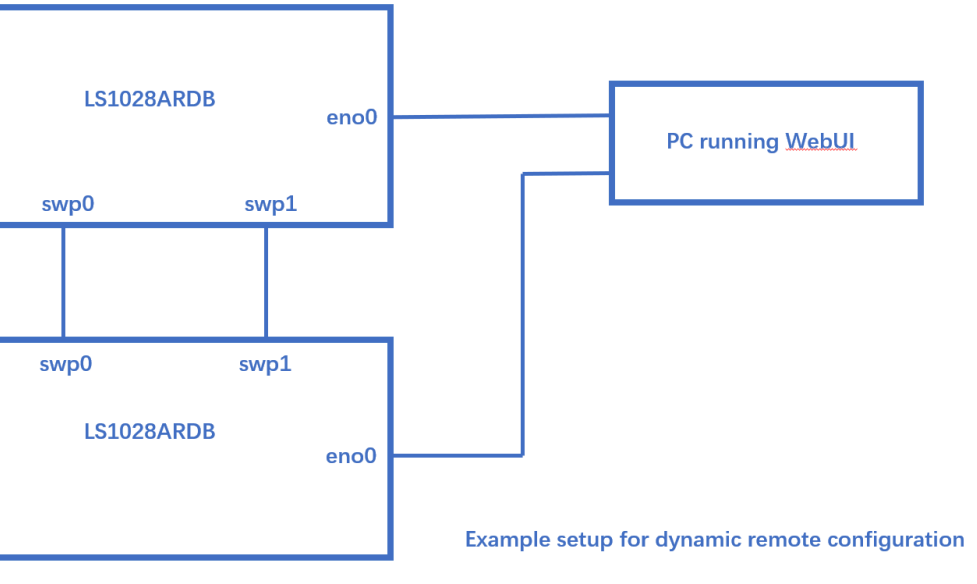


Figure 33. Dashboard Configuration Demo

7.1 Stream Register

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 topograph.

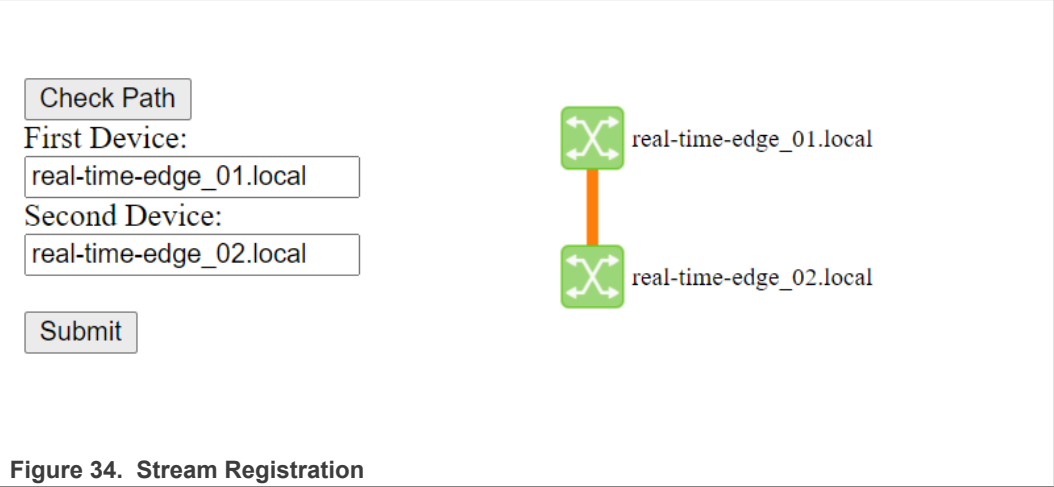
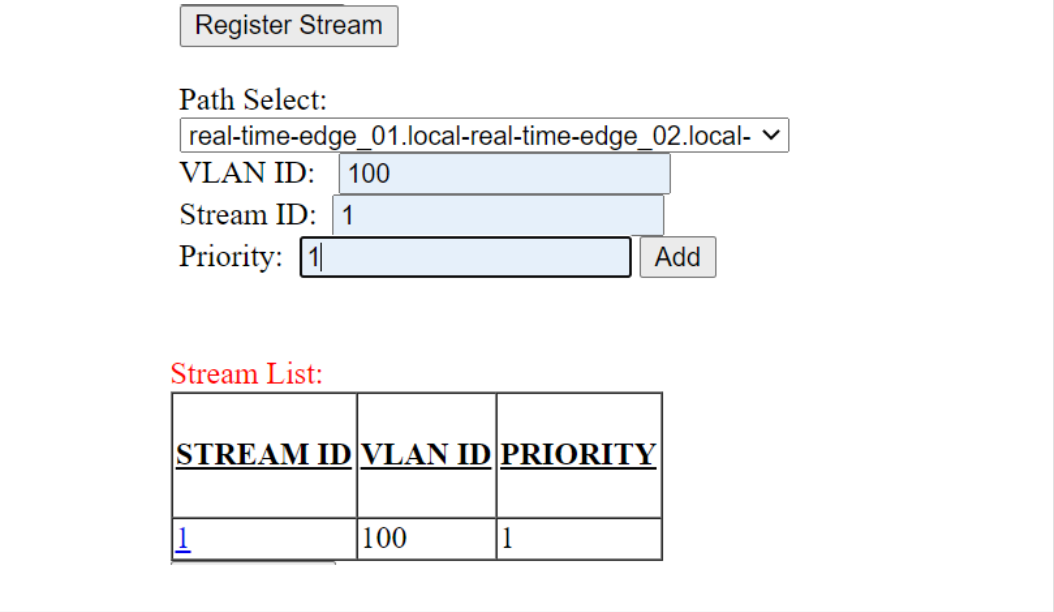


Figure 34. Stream Registration

Click “**Register Stream**” button, then select the path in path select. Fill VLAN ID, Stream ID, and priority, click “**Add**” button. There is output a stream table.

Figure 35. Registering a Stream



7.2 Configure stream identification

Click one stream ID in stream table, jump to stream configuration page. Select streamidentify and fill information in input boxes. The stream MAC information and VLAN ID identify a stream according to the 802.1CB definition. This is used by the PSFP configuration, so this streamidentify page need to be configured before configuring Qci and CFQ.

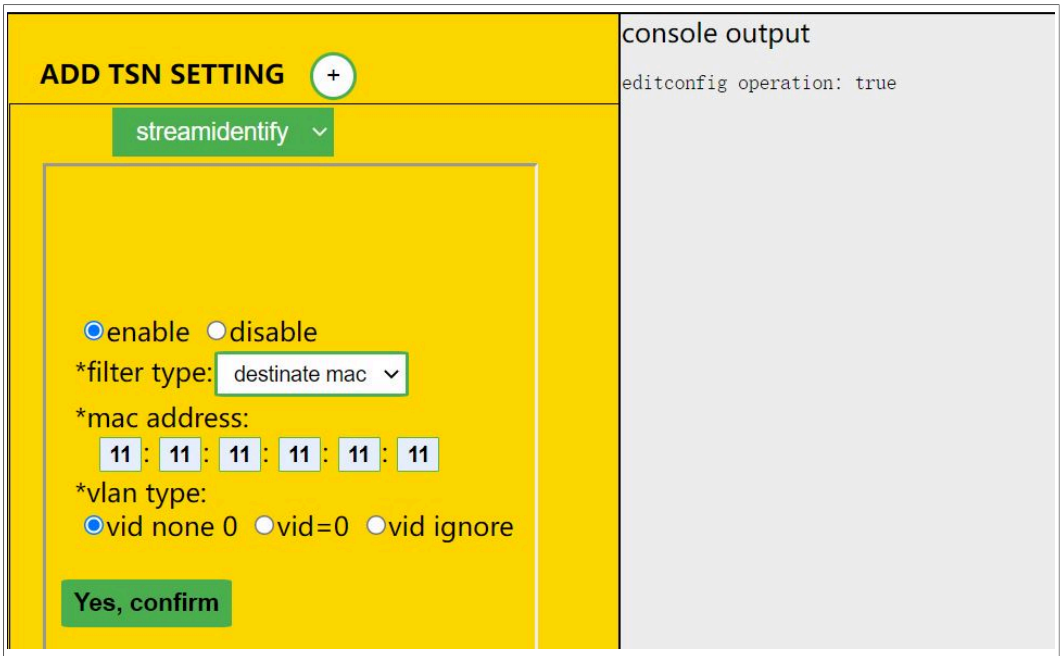


Figure 36. Configuring stream identification

7.3 Configure Qbv and Qci On Stream

Select Qbv, and then fill basetime, cycletime, and gate open time in input boxes. Selecting **enable Qci** button configures both Qci gate control on input port and Qbv control list on 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.

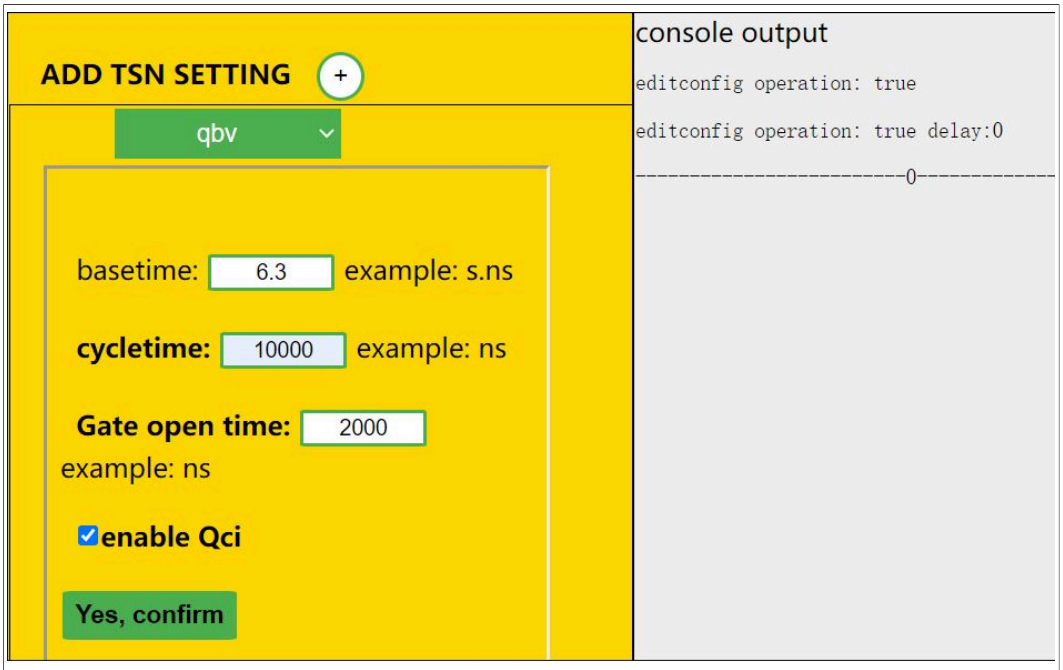


Figure 37. Configuring Qbv and Qci On Stream

7.4 Configure 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.

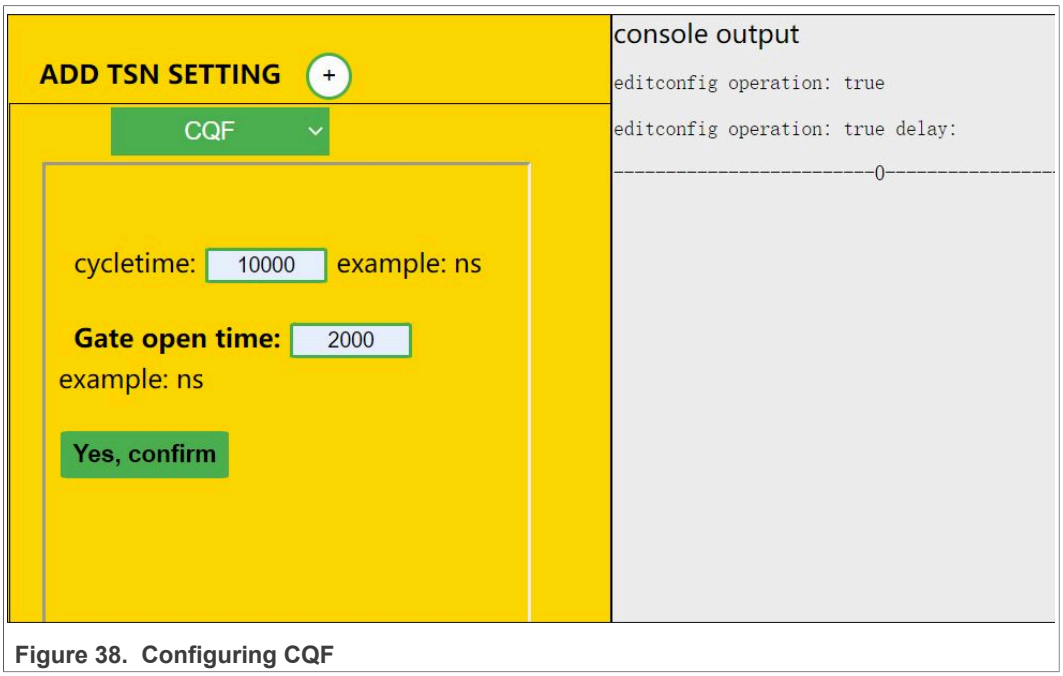


Figure 38. Configuring CQF

7.5: Configuring Flow Meter policy on stream

IEEE 802.1Qci avoids traffic overload condition, that impact the bridges and the end-devices on a network. This improves the robustness of a network, for instance, Daniel-of-Service (DoS) attack, error through streams transmission or likewise if we receive a flow that is not in the schedule time period then it is dropped.

Configuring flow meter on a stream can limit the bandwidth on the ingress port of each bridge through which the flow passes.. This can avoid traffic overload and protect all network nodes from flood attack.

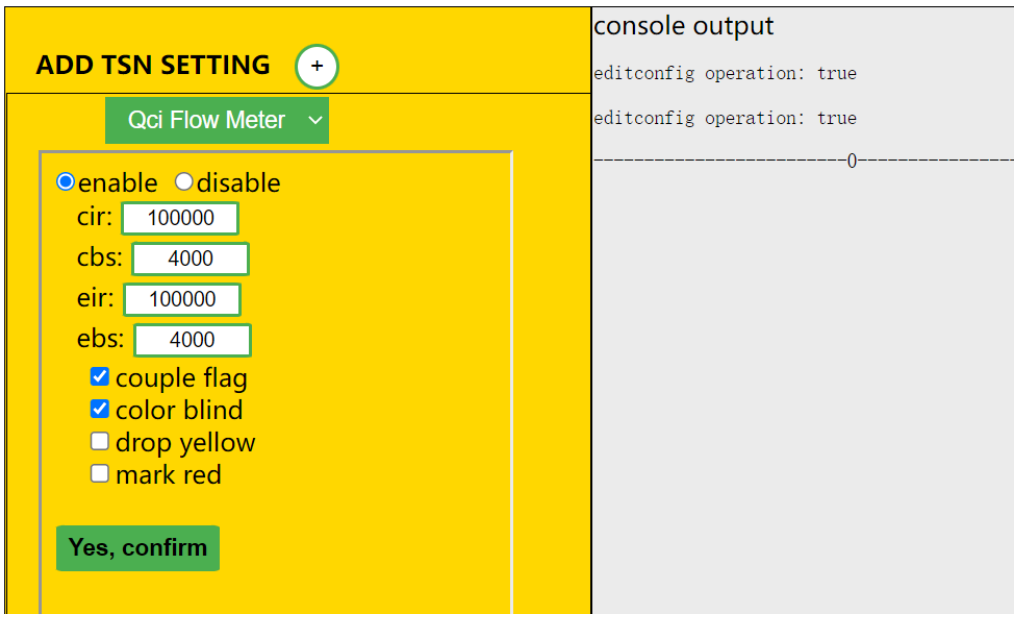


Figure 39. Configuring Flow Meter policy on stream

4.1.3 TSN on i.MX 8DXL / i.MX 8M Plus / i.MX 93

4.1.3.1 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, 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
```

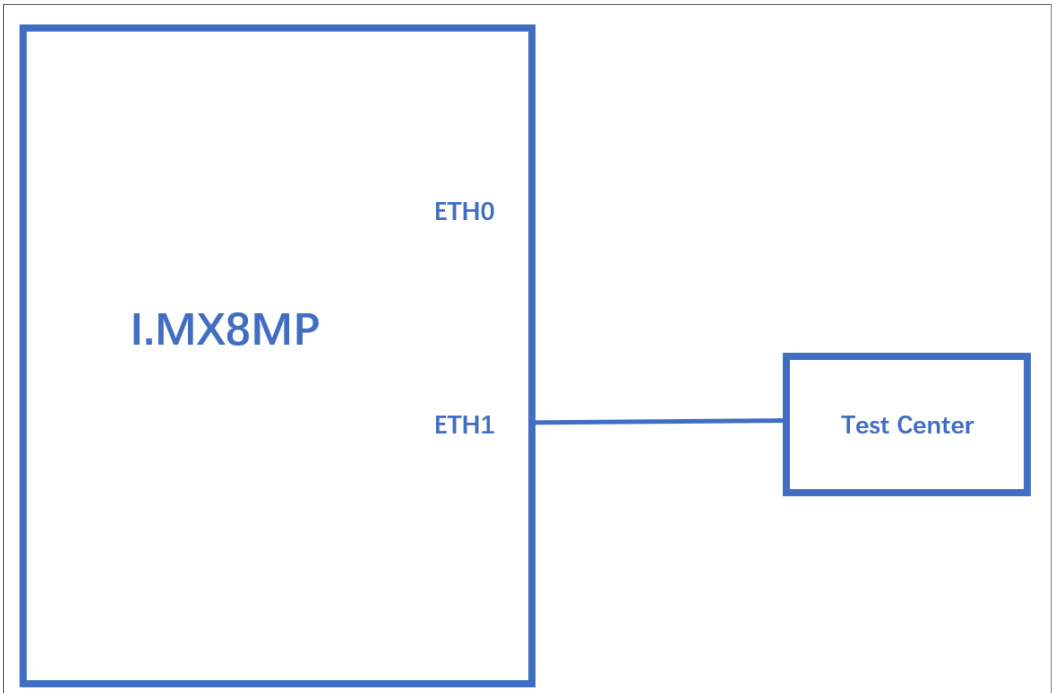


Figure 40. TSN test environment setup

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.*

4.1.3.2 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 timestamping 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)
       ptpv1-l4-delay-req
 (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
       ptpv2-l4-event         (HWTSTAMP_FILTER_PTP_V2_L4_EVENT)
       ptpv2-l4-sync          (HWTSTAMP_FILTER_PTP_V2_L4_SYNC)
       ptpv2-l4-delay-req
 (HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ)
       ptpv2-event            (HWTSTAMP_FILTER_PTP_V2_EVENT)
       ptpv2-sync             (HWTSTAMP_FILTER_PTP_V2_SYNC)
       ptpv2-delay-req        (HWTSTAMP_FILTER_PTP_V2_DELAY_REQ)
```

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 `dwmac` 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 `dwmac` only after "`ifconfig eth1 up`".*

Note: *If Qbu preemption is enabled on remote device and the PTP packets are sent as preemption frames, the ptp4l command should run clock synchronization with the parameter `--hwts_filter=full`. For example:*

```
 ptp4l -i eth1 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m --
hwts_filter=full
```

4.1.3.3 Qbv

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 \
    num_tc 5 map 0 1 2 3 4 queues 1@0 1@1 1@2 1@3 1@4  base-
time 1577187882000000000 \
        sched-entry S  1 100000 \
        sched-entry S  2 100000 \
        sched-entry S  4 100000 flags 2
```

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.*

4.1.3.4 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.

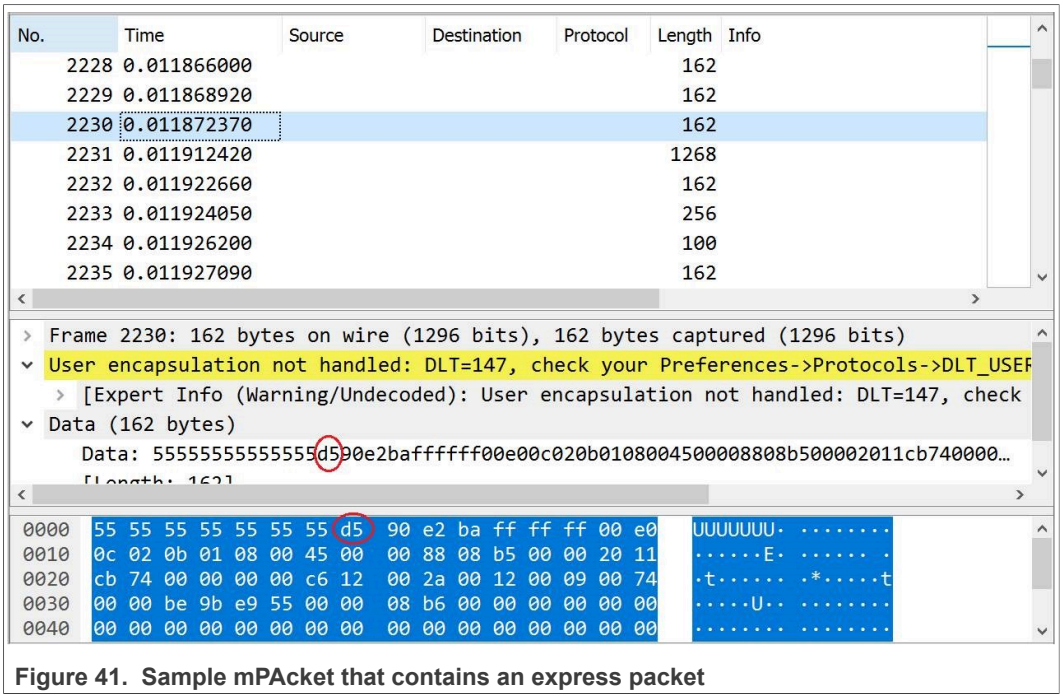


Figure 41. Sample mPAcket that contains an express packet

b. Below is an example mPacket containing an initial fragment of a preemptable packet, which has SMD-S1 value of 0x4C.

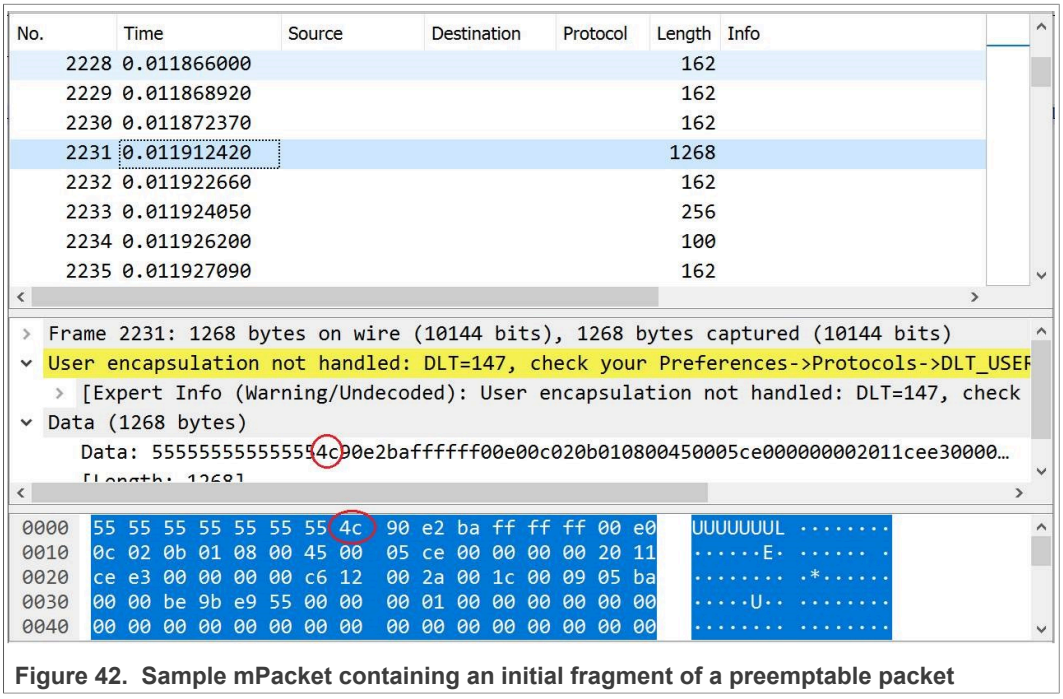


Figure 42. Sample mPacket containing an initial fragment of a preemptable packet

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.

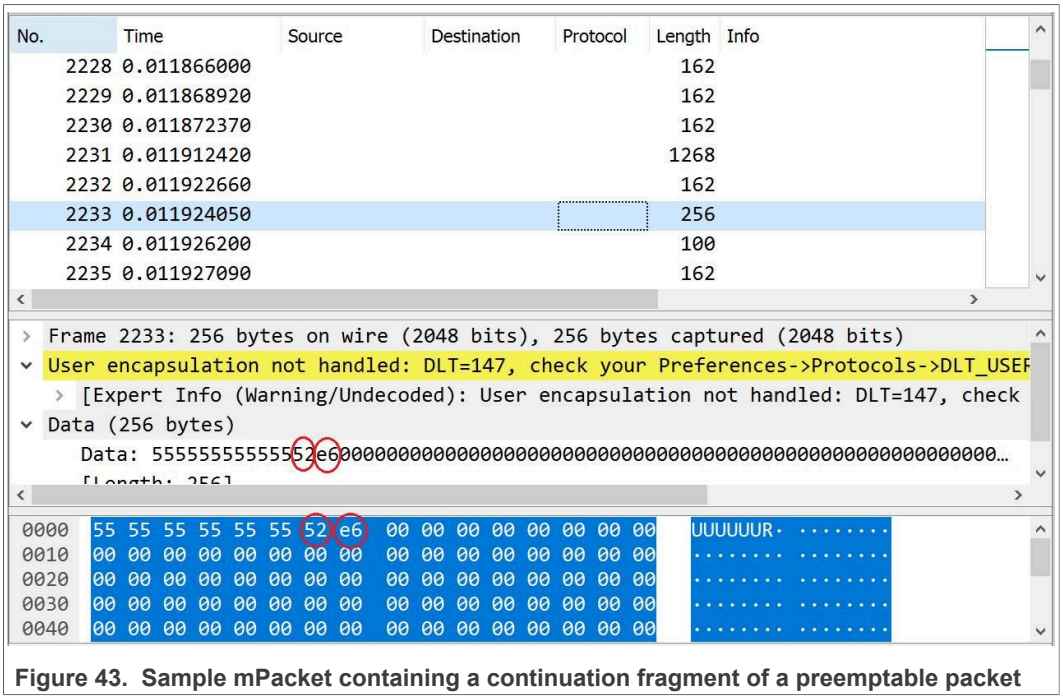


Figure 43. Sample mPacket containing a continuation fragment of a preemptable packet

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 1577187882000000000 \
        sched-entry H  2 100000 \
        sched-entry R  4 100000 flags 2
```

4.1.3.4.1 Preemption verify

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 two eth1 of boards back to back. Then, use the following command to enable the LLDP verification and hardware verification:

```
ethtool --set-frame-preemption eth1 lldp-verify on fp on
 preemptible-queues-mask 0x04 min-frag-size 60
```

After running the command on two boards, wait a moment until LLDP frames have been exchanged. Use the following command to show the preemption status.

```
 ethtool --show-frame-preemption eth1
```

Capture the LLDP frames on eth1 port to see the additional Ethernet capabilities TLV.

4.1.3.5 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:
     src_mac: a6:85:82:fc:89:bf dst_mac: 02:5d:ae:ba:e0:00
     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:
  src_mac: a6:85:82:fc:89:bf dst_mac: 02:5d:ae:ba:e0:00
  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: 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 -s
 1500 -n 0
```

8. Capture the streams on TestCenter, the frames sort by one Q3 frame and two Q4 frames.

4.1.4 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.

4.1.4.1 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
```

4.1.4.1.1 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
   ```

4.1.4.1.2 Qbv

This test includes the Basic Gates Closing test, Basetime test, and the `Qbv` performance test. These are described in the following sections.

4.1.4.1.2.1 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.
```

4.1.4.1.2.2 Basetime test

Base on case 1 `qbv1.txt` gate list.

```
#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:
tsntool> ptptool -g
#Below example shows basetime start at 260.666 s (start of 1
 January 1970):
tsntool> qbvset --device eno0 --entryfile qbv1.txt --basetime
 260.666
tsntool> qbvget --device eno0 #User can check configchange time
tsntool> 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.

```
ping 192.168.0.2 -c 1 -s 1300 #frame should not pass
```

4.1.4.1.2.3 Qbv performance test

Use the setup described in the figure below for testing ENETC port0 (MAC0).

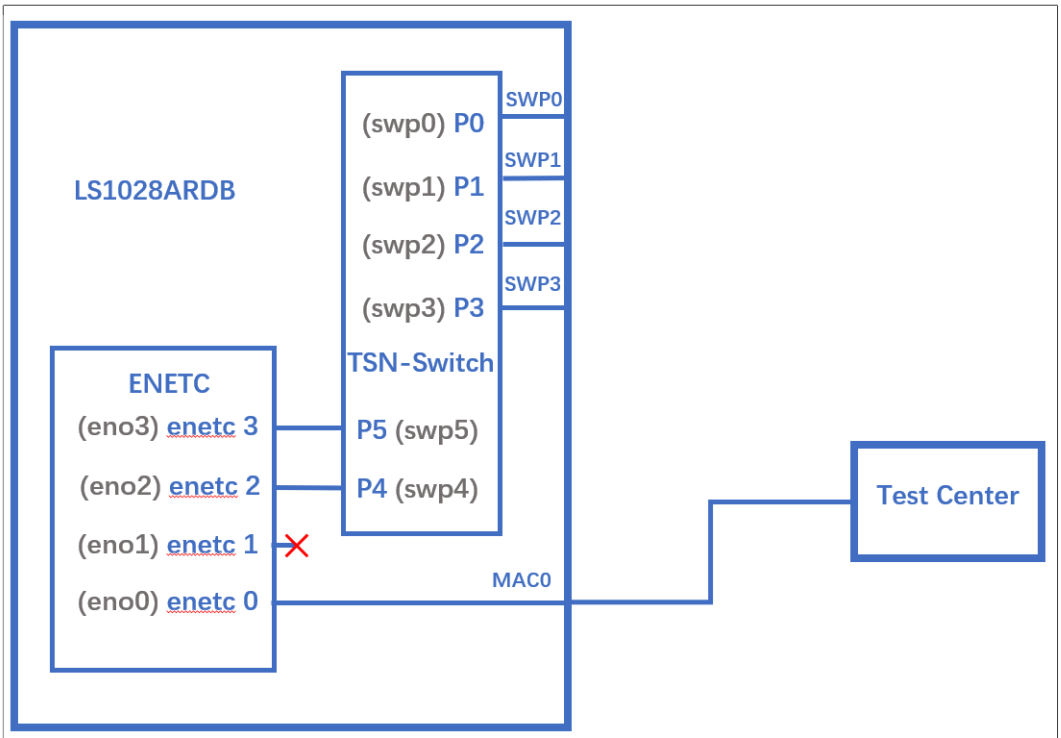


Figure 44. Setup for testing ENETC port0

Note: *TestCenter is a device 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.*

```
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
```

4.1.4.1.2.4 Using taprio Qdisc Setup Qbv

LS1028ardb support the taprio qdisc to setup Qbv either. Below is an example Setup.

```
#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 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7
 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 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7
 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
tc qdisc del dev eno0 parent root handle 100 taprio
```

4.1.4.1.3 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 following figure for the Qbu test.

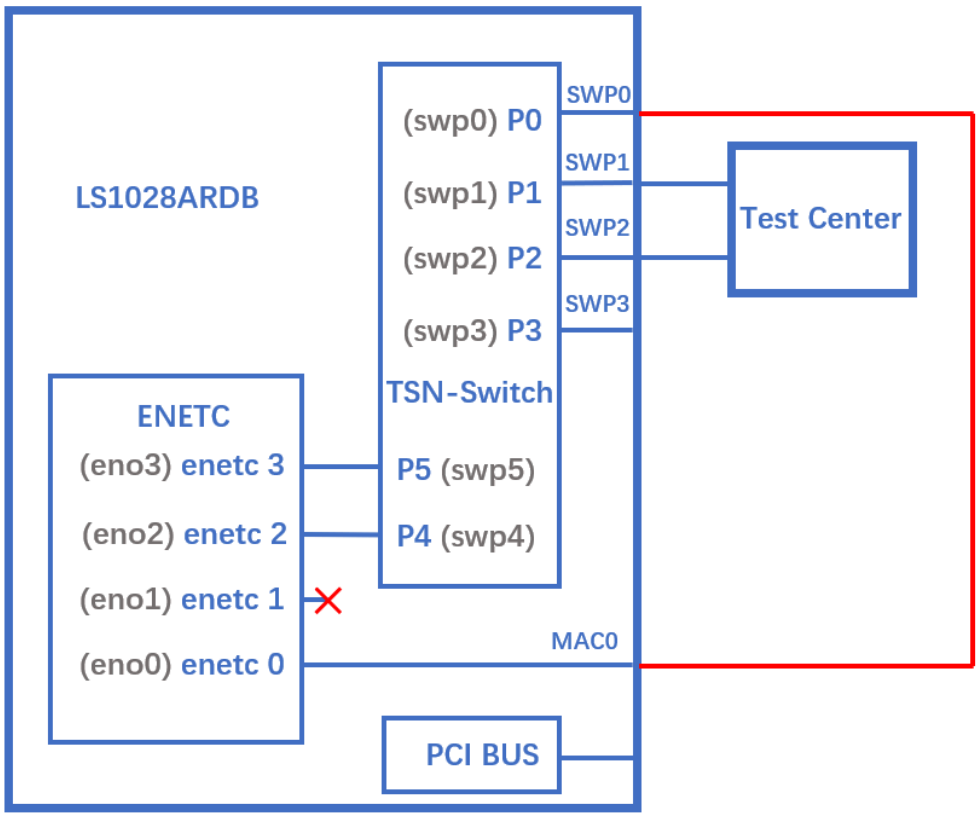


Figure 45. Qbu test

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:

```
ethtool eno0
```

3. If it is not 1 Gbps, set it to 1 Gbps by using the command:

```
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):

```
devmem2 0x1fc100048 w 0x111 #DEV_GMII:MM_CONFIG:ENABLE_CONFIG
```

5. ENETC port setting set and frame preemption test:

```
ip link set eno0 address 90:e2:ba:ff:ff:ff
tsntool 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.

```
tsntool regtool 0 0x11f18
```

Note: *0x11f18 counting the merge frame count:*

```
0x11f18 Port MAC Merge Fragment Count TX Register
 (MAC_MERGE_MMFCTXR)
```

LS1028ARDB also supports ethtool setup for preemption as in the example below:

```
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 4.1.3.4.1.

4.1.4.1.4 Qci

Use the following as the background setting:

- Set eno0 MAC address

```
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.

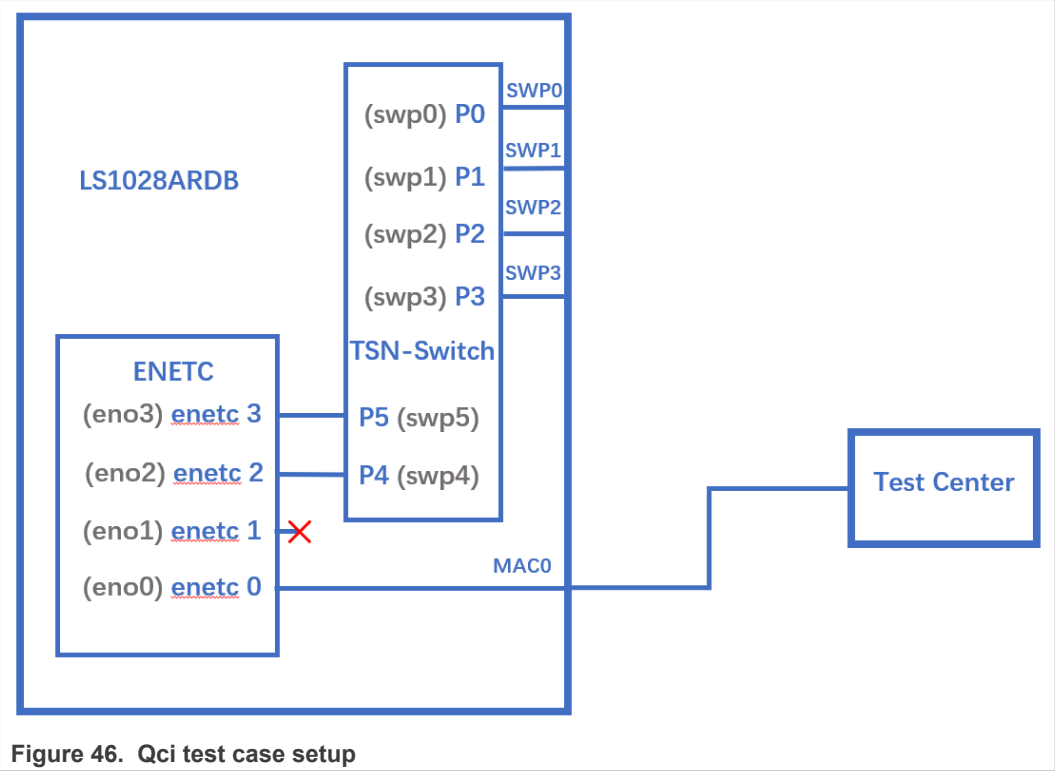


Figure 46. Qci test case setup

Note: *TestCenter is a device to send streams to enetc0 of LS1028ardb board. User also can use another board to send streams.*

4.1.4.1.4.1 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).

```
tsntool> qcisfiget --device eno0 --index 2
```

- Set Stream Gate entry 2

```
tsntool> qcisgiset --device eno0 --index 2 --initgate 1
```

- Send a frame from the opposite device port.

```
tsntool> qcisfiget --device eno0 --index 2
```

- Set Stream Gate entry 2, gate closes permanently.

```
tsntool> qcisgiset --device eno0 --index 2 --initgate 0
```

- Send a frame from the opposite device port.

```
tsntool> qcisfiget --device eno0 --index 2
#The result should look like below:
 match  pass  gate_drop  sdu_pass  sdu_drop red
   1    0   1     1    0    0
```

4.1.4.1.4.2 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.

```
tsntool> cbstreamidset --device eno0 --index 1 --nullstreamid
 --nulldmac
0x000000800010 --nulltagged 3 --nullvid 10 --streamhandle 100
```

3. Get stream identify entry index 1.

```
tsntool> cbstreamidget --device eno0 --index 1
```

4. Set Stream filer entry 1 with stream gate entry id 1.

```
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).

```
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.

```
tsntool> qcisfiget --device eno0 --index 1
```

7. The result should look like the output below:

```
match pass gate_drop sdu_pass sdu_drop red
1 0 1 1 0 0
```

4.1.4.1.4.3 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
```

4.1.4.1.4.4 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.

4.1.4.1.4.5 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 --
flowmeterid 2
tsntool> qcifmiset --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:

```
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:

```
tsntool> qcifmiget --device eno0 --index 2
================================================================
bytecount drop dr0_green dr1_green dr2_yellow remark_yellow dr3_red
 remark_red
1c89 0 4c 0 0 0 0 0
================================================================
index = 2
cir = c34c
cbs = 5dc
eir = 4c4b3c
ebs = 5dc
couple flag
color mode
```

4.1.4.1.5 Qav

4.1.4.1.5.1 Using tsntool

The following figure illustrates the hardware setup diagram for the Qav test.

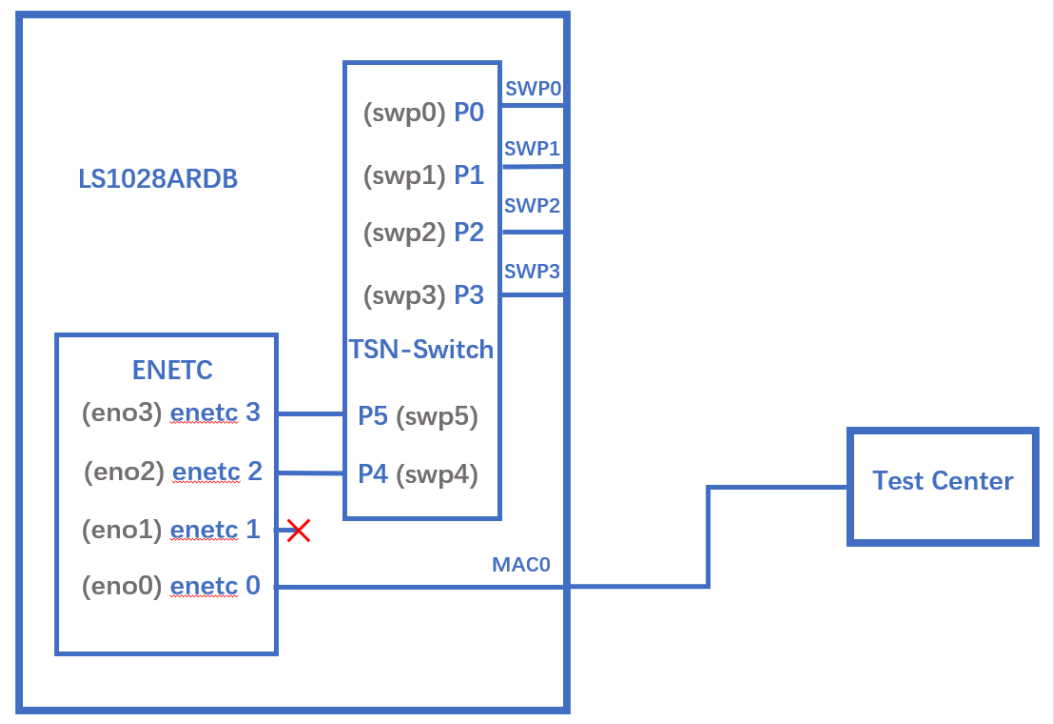


Figure 47. Qav test setup

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
tsntool cbsset --device eno0 --tc 6 --percentage 20
```

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.

4.1.4.1.5.2 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
```

4.1.4.2 TSN configuration on Felix switch

The following sections describe examples for the basic configuration of TSN switch.

4.1.4.2.1 Switch configuration

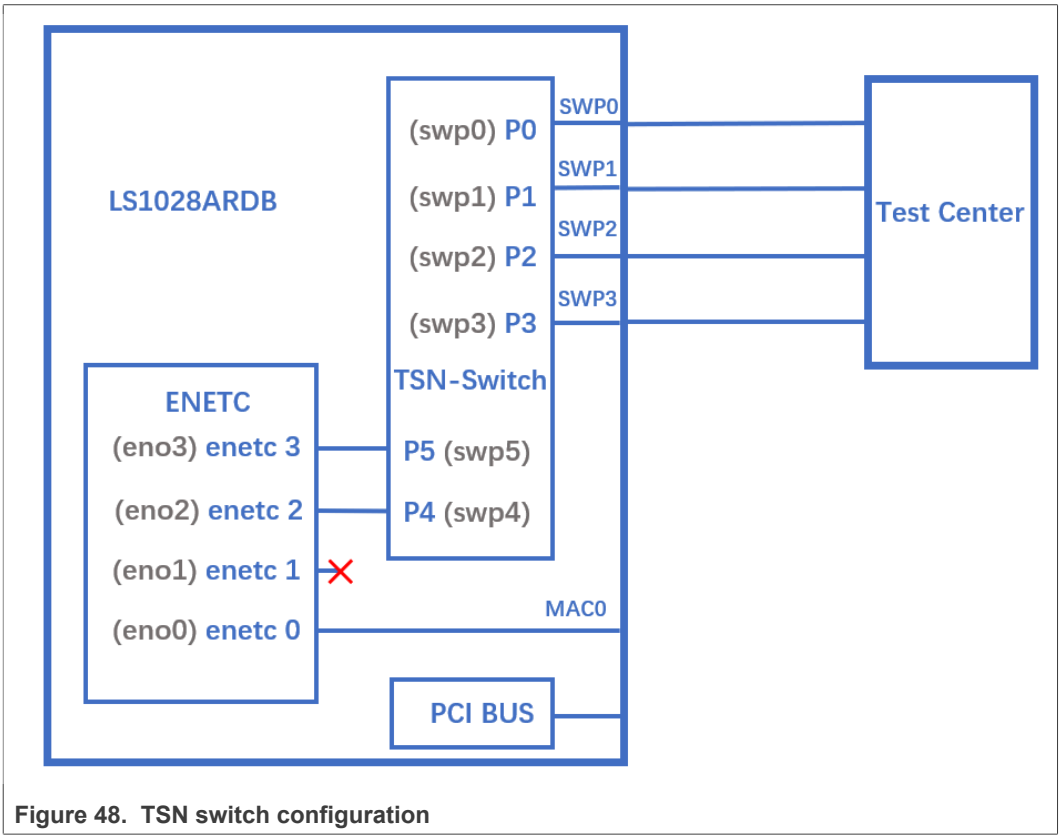


Figure 48. TSN switch configuration

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
```

4.1.4.2.2 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

```
# 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 20000 protocol 0x88f7
 flower skip_sw action trap action goto chain 21000
tc filter add dev 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 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 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
```

4.1.4.2.3 Qbv test setup for LS1028ARDB

The following figure describes the setup for `Qbv test` on LS1028ARDB.

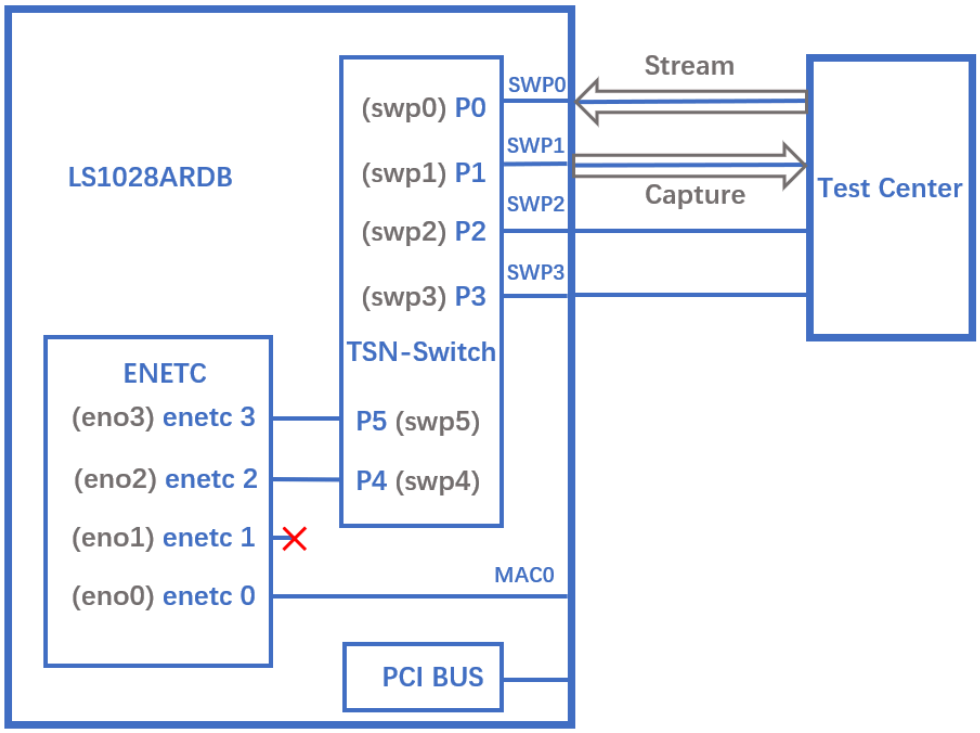


Figure 49. Qbv test

Reserve buffer for each queue on ingress and egress port to avoid resource depletion when Qbv gate is closed.

```
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
```

4.1.4.2.3.1 Tsntool usage

Closing basic gates

Use the set of commands below for basic gate closing.

```
echo “t0 00000000b 20000” > qbv0.txt
#Explaination:
# '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
t1 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.*

4.1.4.2.3.2 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@1 1@2 1@3 1@4 1@5 1@6 1@7 base-time 0
 sched-entry S 02 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
```

4.1.4.2.4 Qbu

The figure below illustrates the setup for performing the Qbu test using the TSN switch.

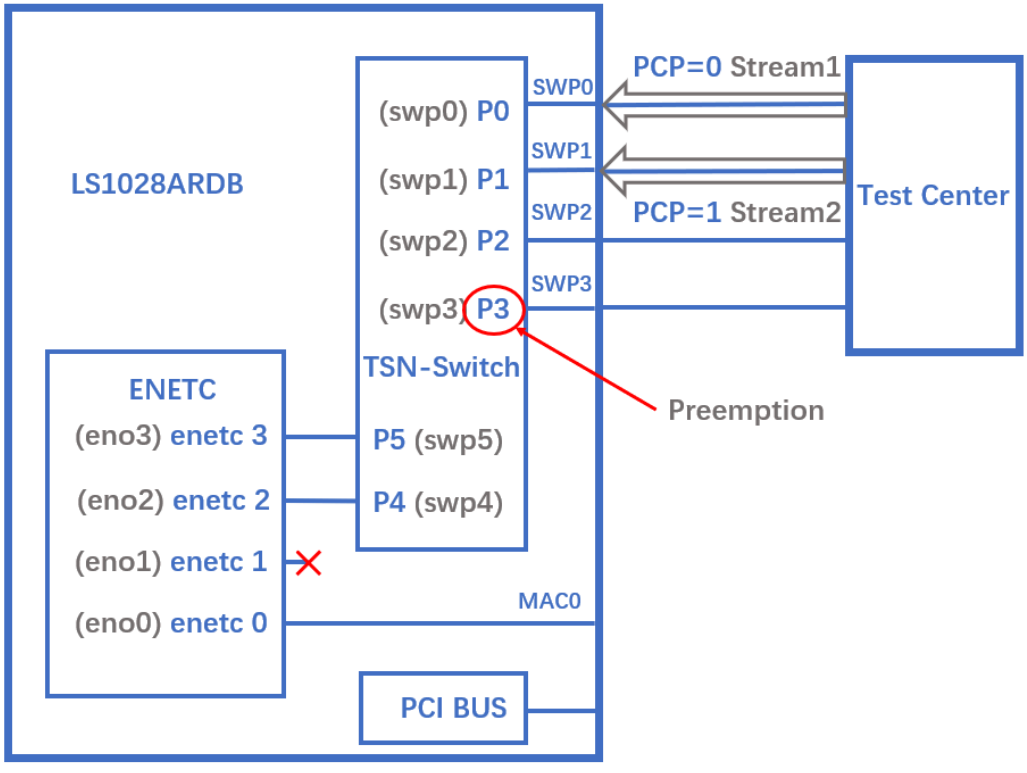


Figure 50. Qbu test on switch

4.1.4.2.4.1 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
qbvset --device swp3 --entryfile qbv0.txt
```

Send two streams from TestCenter. Observe that packets in queue 1 are preempted when gate 1 is closed.

4.1.4.2.4.2 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 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 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.

4.1.4.2.5 Qci

The figure below illustrates the Qci test case setup.

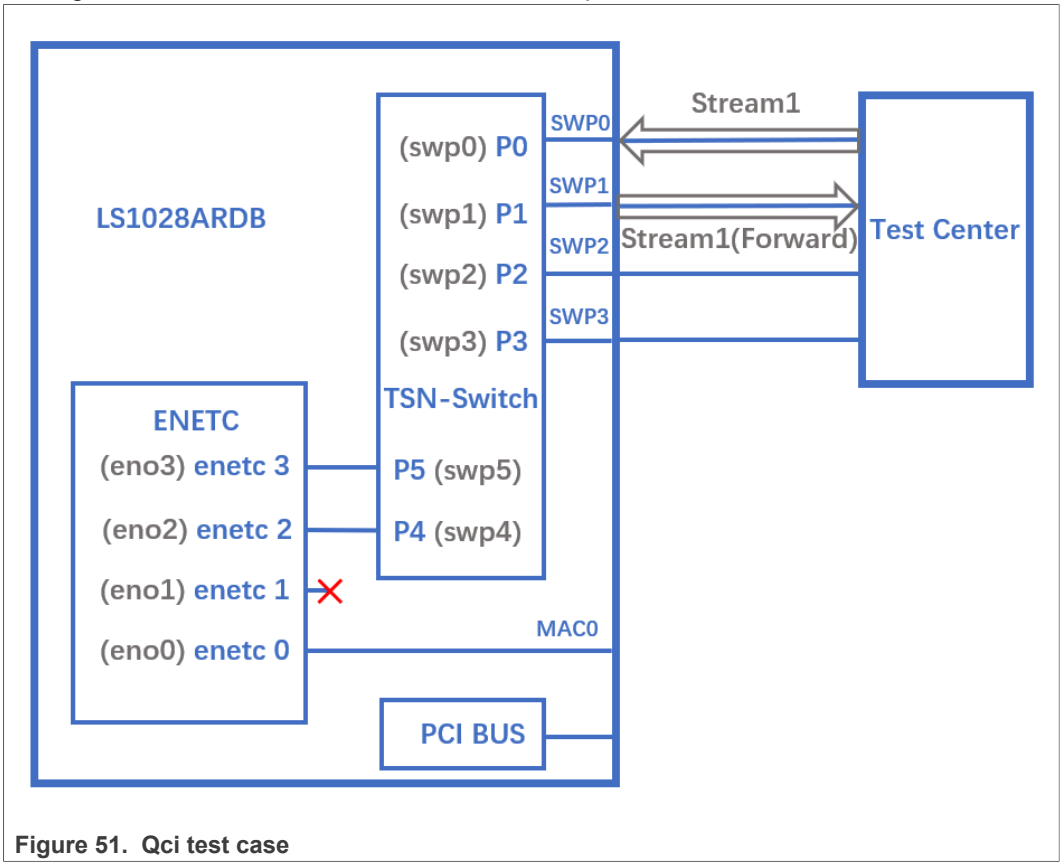


Figure 51. Qci test case

4.1.4.2.5.1 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
 --nulldmac  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.txtx
tsntool> 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

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.

```
ethtool -S swp0
```

Note that not all frames could pass and get yellow and red frames.

8. Test cf mode.

```
tsntool> qcifmiset --device swp0 --index 68 --cir 100000 --
cbs 4000 --ebs 4000 --eir 100000 --cf
```

9. Send one yellow stream (rate = 200M) on TestCenter.

```
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.

```
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.

```
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.

```
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.

```
ethtool -S swp1
```

Note that the frame could pass, and green_prio_4 has increased.

4.1.4.2.5.2 Tc-flower usage

The figure below illustrates the TC-flower-based Qci test case setup.

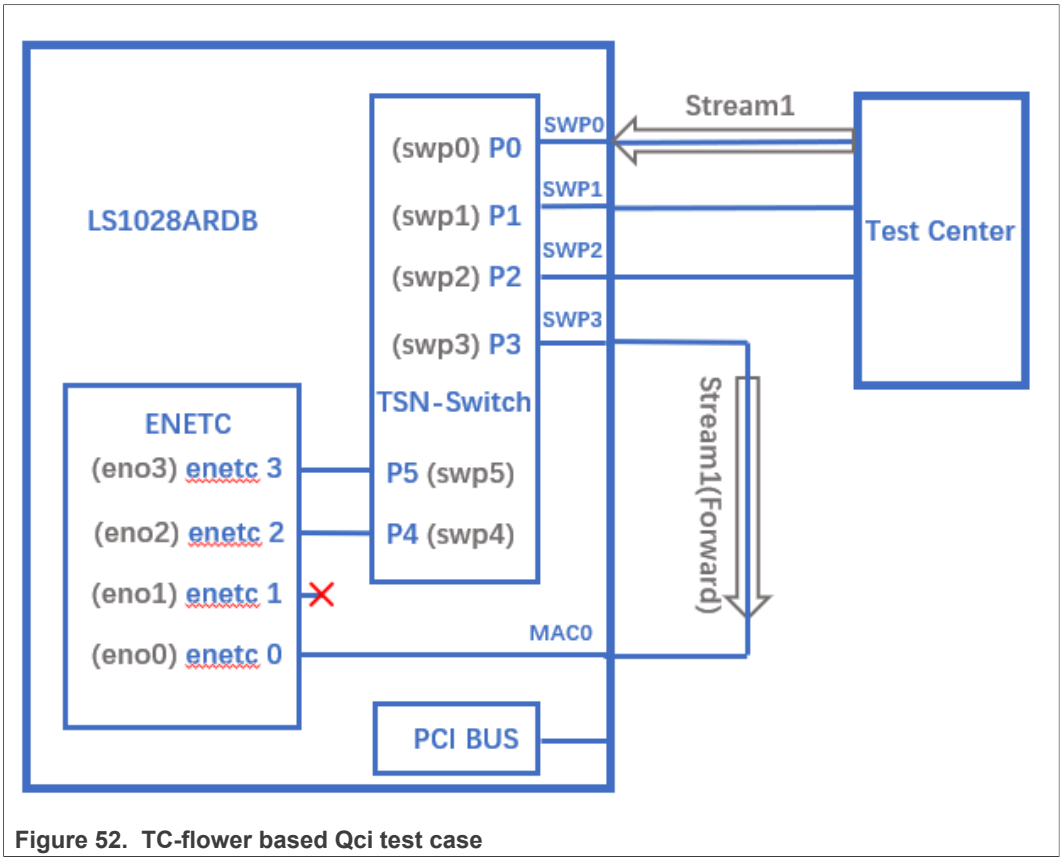


Figure 52. TC-flower based Qci test case

1. Add the MAC "CA:9C:00:BC:6D:68" in MAC table by using "bridge fdb" command if it is not learned.

```
bridge fdb add dev swp3 CA:9C:00:BC:6D:68 vlan 1 master static
```

2. Register chains on ingress port swp0. Refer to Section 4.4.2.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.*

4.1.4.2.6 Qav

The below figure illustrates the Qav test case setup.

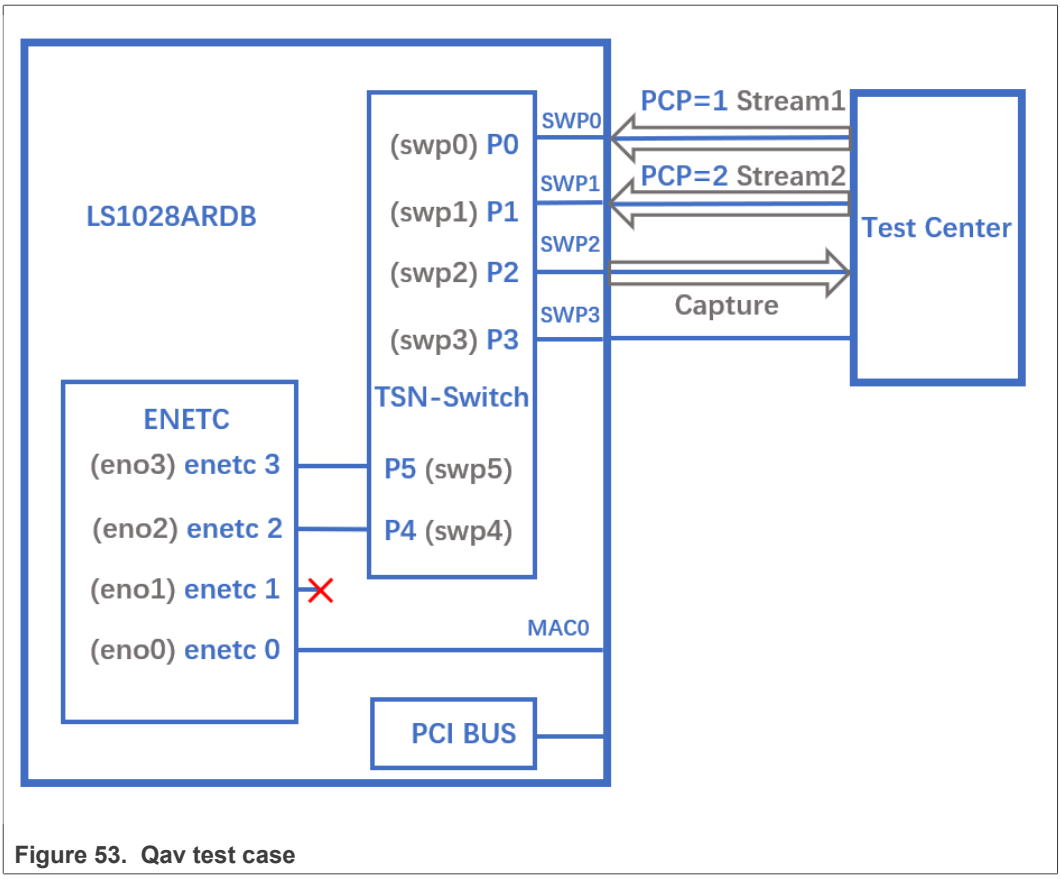


Figure 53. Qav test case

4.1.4.2.6.1 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 lager 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),…
```

4.1.4.2.6.2 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: mqprio 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 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 bandwith 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
tc qdisc del dev swp2 parent 1:3 cbs
```

4.1.4.2.7 802.1CB

The following figure describes the test setup for the seamless redundancy test case.

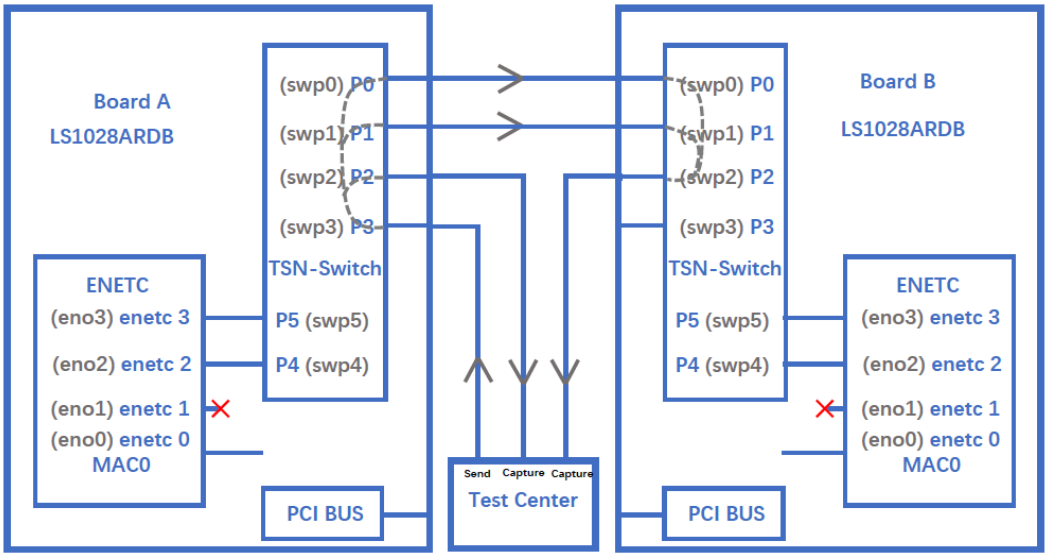


Figure 54. Seamless redundancy test

4.1.4.2.7.1 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
```

On board B

```
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
bridge vlan add dev swp0 vid 1 pvid
bridge vlan add dev swp1 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.

4.1.4.2.7.2 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 swp0 vlan 1 master
 static
tsntool> cbstreamidset --device swp2 --index 1  --
nullstreamid --nulldmac 0x7EA88C9B41DD --nullvid 1  --
streamhandle 1
tsntool> cbrec --device swp0 --index 1 --seq_len 16 --his_len
 31 --rtag_pop_en
```

In the cbrec 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 dest mac to be 7E:A8:8C:9B:41:DD.
3. Capture frames from swp2 of board B on TestCenter, we can get only one frame without sequence tag.

4.1.4.2.8 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.

4.1.4.2.8.1 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.

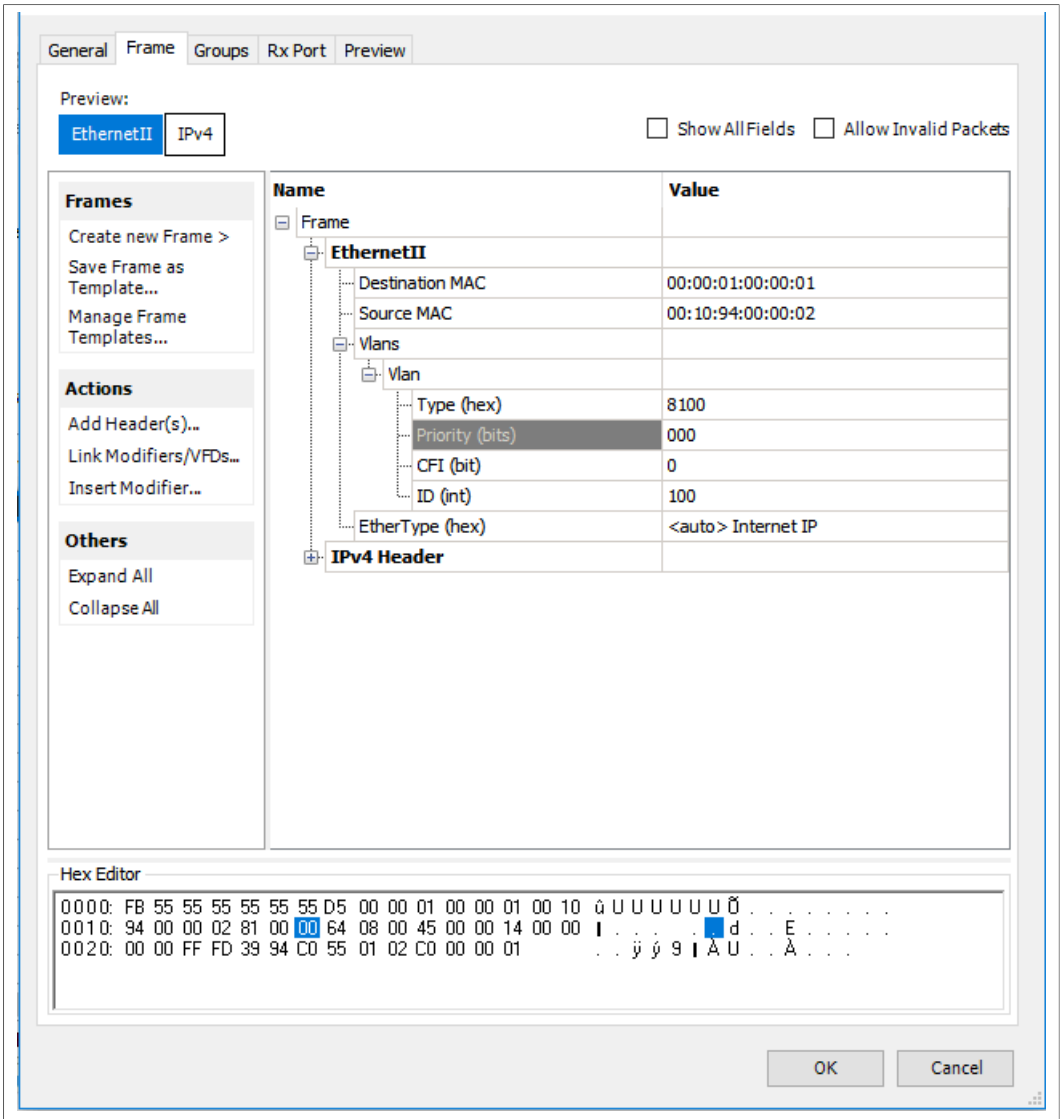


Figure 55. Using PCP value of Vlan tag

4.1.4.2.8.2 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 the following figure.

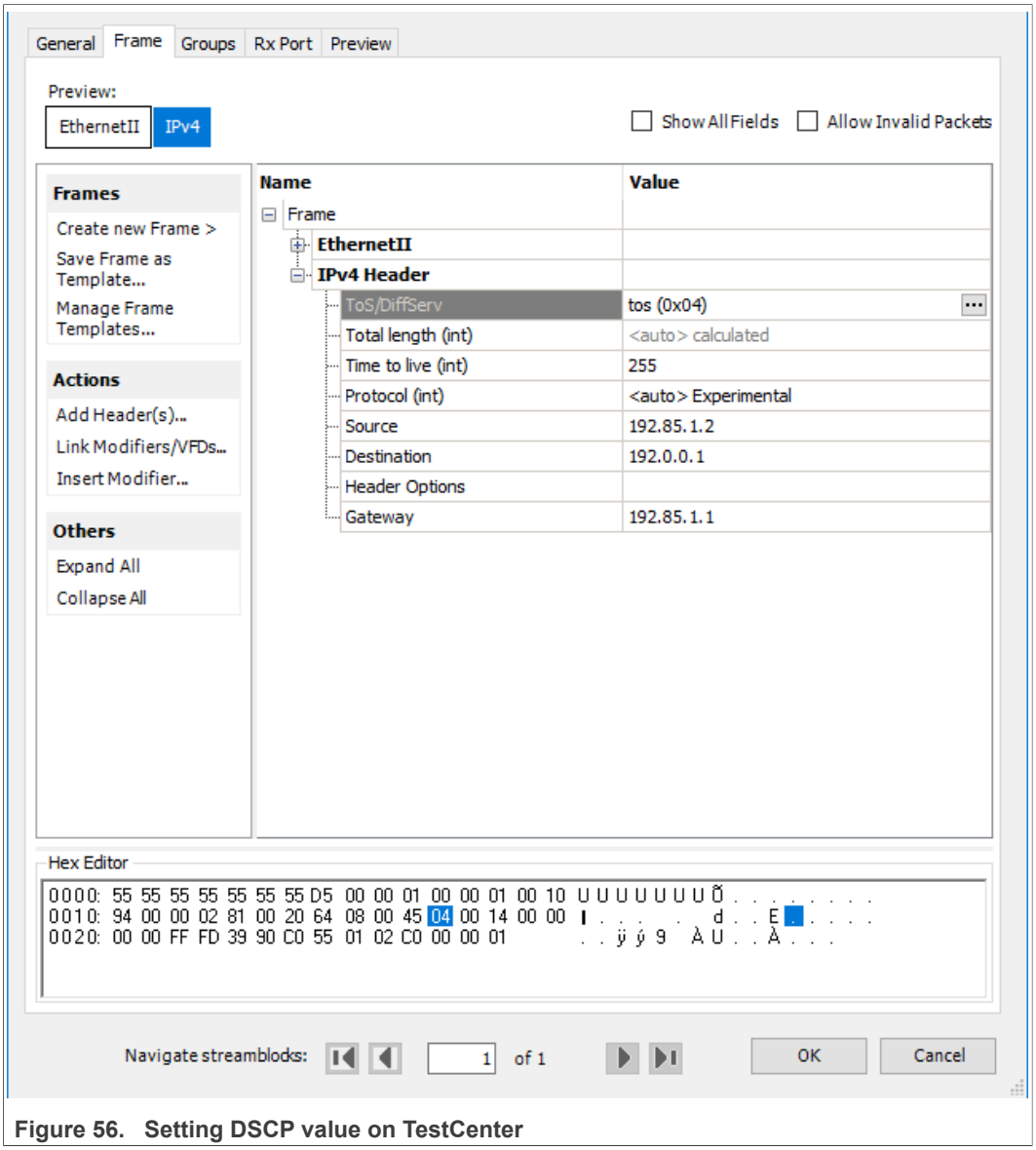


Figure 56. Setting DSCP value on TestCenter

4.1.4.2.8.3 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
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.*

4.1.5 TSN on LS1021A-TSN

On the LS1021A-TSN platform, TSN features are provided by the SJA1105TEL Automotive Ethernet switch. These hardware features comply to pre-standard (draft) versions of the following IEEE specifications:

- 802.1Qbv - Time Aware Shaping
- 802.1Qci - Per-Stream Filtering and Policing
- 1588v2 - Precision Time Protocol

The following demonstration illustrates the SJA1105 hardware features listed below:

- Ingress rate limiting via the L2 (best-effort) policers
- Time-aware shaping
- 802.1AS gPTP synchronization

4.1.5.1 Topology

For demonstrating the SJA1105 TSN features, the following topology is required:

- 1 LS1021A-TSN board, acting as a TSN switch
- 1 generic host (can be a PC or another board) capable of PTP hardware timestamping, acting as a sender of latency-sensitive traffic
- 1 generic host (can be a PC or another board), acting as a sender of high-bandwidth traffic
- 1 generic host (can be a PC or another board) capable of PTP hardware timestamping, acting as receiver for the latency-sensitive and for the high-bandwidth traffic

The required software packages for the generic hosts are:

- ptp4l, phc2sys and phc_ctl from the linuxptp package: https://sourceforge.net/projects/linuxptp/files/v3.1/linuxptp-3.1.tgz
- iperf3
- isochron from the tsn-scripts package: https://github.com/vladimiroltean/tsn-scripts/tree/isochron

The generic hosts are assumed to be connected to the LS1021A-TSN board through an interface called eth0.

This topology is depicted in the following figure.

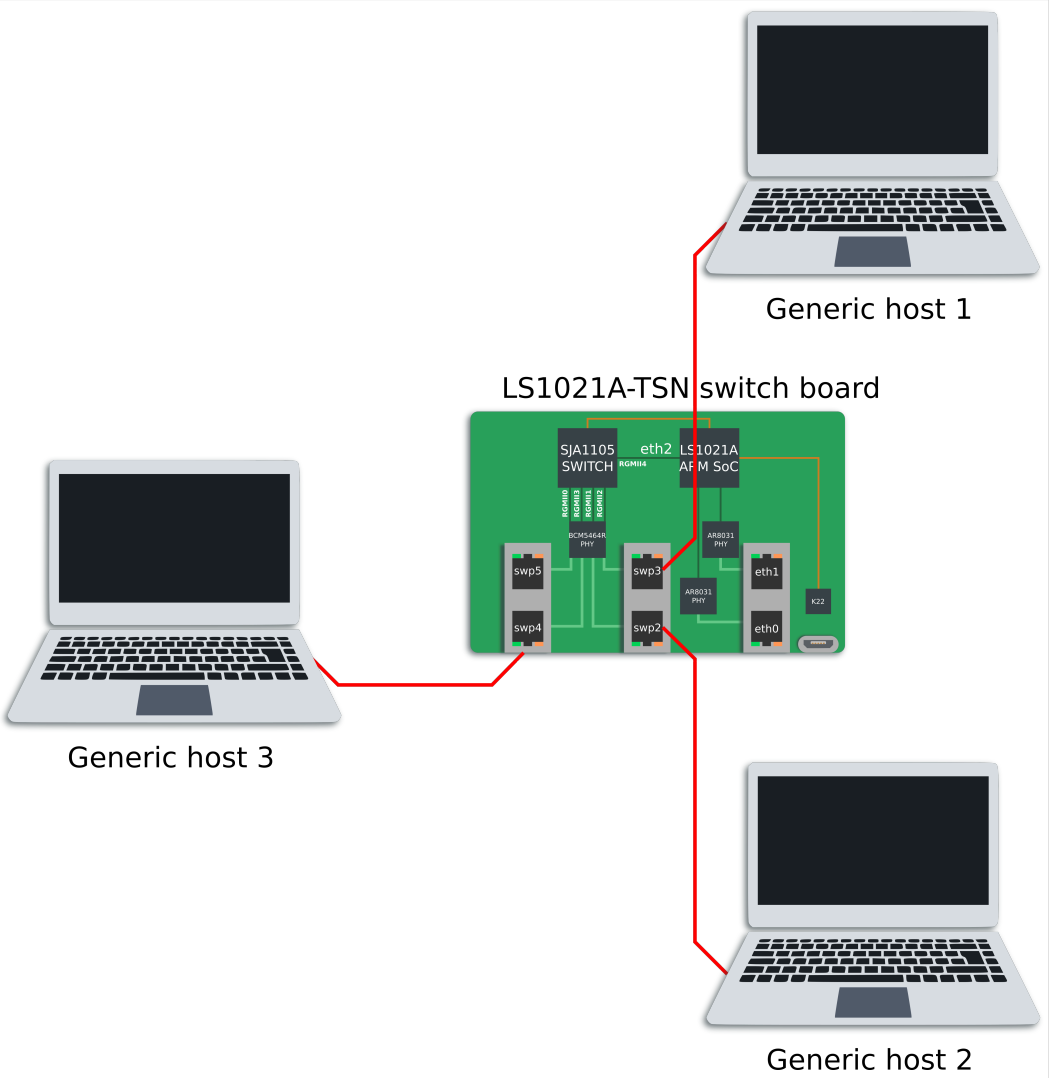


Figure 57. Topology of the demo network

4.1.5.2 SJA1105 Linux support

The Real-time Edge Linux kernel supports SJA1105 switch using the Distributed Switch Architecture (DSA) framework (an overview of which can be found at *https://netdevconf.info/2.1/papers/distributed-switch-architecture.pdf*).

The following kernel configuration options are available for controlling its features:

- CONFIG_NET_DSA_SJA1105: enables base support for probing the SJA1105 ports as four standalone net devices capable of sending and receiving traffic
- CONFIG_NET_DSA_SJA1105_PTP: enables additional support for the PTP Hardware Clock (PHC), visible in /dev/ptp1 on the LS1021A-TSN board, and for PTP timestamping on the SJA1105 ports
- CONFIG_NET_DSA_SJA1105_TAS: enables additional support for the Time-Aware Scheduler (TAS), which is configured via the tc-taprio qdisc offload

The documentation for this kernel driver is available at https://www.kernel.org/doc/html/latest/networking/dsa/sja1105.html. Below is a listing of several driver features.

The LS1021A-TSN device tree (`arch/arm/boot/dts/ls1021a-tsn.dts`) defines the sja1105 port names as swp2, swp3, swp4 and swp5. The numbers have a direct correspondence with the chassis labels ETH2, ETH3, ETH4 and ETH5. The ETH2 chassis label (represented in Linux by the swp2 net device) should not be confused with the eth2 net device, which represents the LS1021A host port for this switch (called DSA master).

On the LS1021A-TSN board, network management is done by the systemd-networkd daemon, whose configuration files are located in `/etc/systemd/network/`. On this board, the following configuration files for systemd-networkd are present by default:

- `br0.netdev`: Creates a bridge net device with VLAN filtering disabled, STP disabled and MVRP disabled
- `br0.network`: Configures the net devices enslaved to br0 to request an IPv4 address via DHCP
- `eth0.network`, `eth1.network`, `swp.network`: Configures all six ports of the LS1021A-TSN board to be part of the same br0 bridge (four ports are bridged in hardware, two ports are bridged in software)
- `eth2.network`: Configures the DSA master port to come up automatically, and assigns it a dummy link-local IP address. For using the switch net devices, ensure that the DSA master interface is up.

Although all ports are configured for L2 forwarding by default (and therefore the only IP address for this board should be assigned to br0), this can be changed by removing the "`Bridge=br0`" line from the files in `/etc/systemd/network/` and then running "`systemctl restart systemd-networkd`".

In standalone mode, each SJA1105 port is able of acquiring an IP address and transferring general purpose packets to/from the kernel. This is internally supported by the kernel driver by repurposing the VLAN tagging functionality for switch port separation and identification. Therefore the ability to support general purpose traffic I/O only works as long as the user does not request VLAN tagging, via the bridge `vlan_filtering` option. When this happens, the switch driver goes to a reduced functionality mode, where the swpN net devices are no longer capable of sending and receiving general packets to/from the kernel. This is a hardware limitation that can be somewhat mitigated by enabling the `best_effort_vlan_filtering devlink` parameter (by following the steps in the kernel documentation).

Actually there is a second mechanism of frame tagging, which works for STP and PTP traffic and does not rely on VLAN tagging. Therefore, the STP and PTP protocols remain operational on the sja1105 driver even when the ports are enslaved to a bridge with `vlan_filtering=1`.

When VLAN awareness is disabled, the sja1105 ports perform no checks on VLAN port membership or PCP, and performs no alteration to the VLAN tags. For these operations, the following command is necessary:

```
ip link set dev br0 type bridge vlan_filtering 1
```

Once VLAN filtering is enabled, the VLAN table of each switch port can be inspected and modified using the "bridge vlan" commands from the iproute2 package.

The STP state machine can be started on the bridge using the following command:

```
ip link set dev br0 type bridge stp_state 1
ip link set dev br0 down
```

```
ip link set dev br0 up
```

The switch L2 address forwarding database (FDB) can be inspected and modified using the "bridge fdb" set of commands.

Port statistics counters can be inspected using the `ethtool -S swpN` command.

The sja1105 port MTU can be configured up to a maximum of 2021 using the following command:

```
ip link set dev swp2 mtu 2000
```

Port mirroring on a sja1105 port (mirroring of ingress and/or egress packets) can be configured via the following set of commands:

```
tc qdisc add dev swp2 clsact
tc filter add dev swp2 ingress matchall skip_sw \
    action mirred egress mirror dev swp3
tc filter show dev swp2 ingress
tc filter del dev swp2 ingress pref 49152
```

There are three types of policers currently supported by the sja1105 driver:

- Port policers: These affect all traffic that is incoming on a port, except traffic that hits a more specific rule (see below). These are configured as follows:

  ```
  tc qdisc add dev swp2 clsact
  tc filter add dev swp2 ingress matchall skip_sw \
      action police rate 10mbit burst 64k
  ```

- Traffic class policers: These affect only traffic having a specific VLAN PCP. To limit traffic with VLAN PCP 0 (also includes untagged traffic) to 100 Mbit/s on port swp2 only:

  ```
  tc qdisc add dev swp2 clsact
  tc filter add dev swp2 ingress protocol 802.1Q flower skip_sw
   \
      vlan_prio 0 action police rate 100mbit burst 64k
  ```

- Broadcast policers: These affect only broadcast traffic (destination MAC ff:ff:ff:ff:ff:ff) received on an ingress port.

  ```
  tc qdisc add dev swp2 clsact
  tc filter add dev swp2 ingress flower skip_sw dst_mac
   ff:ff:ff:ff:ff:ff \
      action police rate 10mbit burst 64k
  ```

In absence of a specific policer allocated to a traffic class or to broadcast traffic, these packets will consume from the bandwidth budget of the port policer.

It is also possible to combine the bandwidth allocation of a traffic class, or of broadcast traffic on multiple ports, and assign them to a single policer. This functionality is called "shared filter blocks" and can be configured as follows (the example below limits broadcast traffic coming from all switch ports to a total of 10 Mbit/s):

```
tc qdisc add dev swp2 ingress_block 1 clsact
tc qdisc add dev swp3 ingress_block 1 clsact
tc qdisc add dev swp4 ingress_block 1 clsact
tc qdisc add dev swp5 ingress_block 1 clsact
```

```
tc filter add block 1 flower skip_sw dst_mac ff:ff:ff:ff:ff:ff
 \
    action police rate 10mbit burst 64k
```

For PTP, the sja1105 driver implements the kernel primitives required for interoperating with the linuxptp and other open source application stacks. Real-time Edge on the LS1021A-TSN is configured to start linuxptp by default in 802.1AS bridge mode on ports swp2, swp3, swp4, and swp5. The following system components are involved:

- ptp4l: Daemon that implements the IEEE 1588/802.1AS state machines. Configured via the `/etc/linuxptp.cfg` file and controlled via the `linuxptp.service systemctl` service.
- phc2sys: Daemon that synchronizes the system time (CLOCK_REALTIME) to the active PHC (`/dev/ptp1`) or vice versa, depending on the board role in the network (PTP master or slave). Configured via the `/etc/linuxptp-system-clock.cfg` file and controlled via the `phc2sys.service` systemctl service.

To inspect the PTP synchronization status of the board, the following commands can be used:

```
systemctl start --now ptp4l
systemctl start --now phc2sys
journalctl -b -u ptp4l -f
journalctl -b -u phc2sys -f
```

Under steady state, the switch ports are expected to maintain a synchronization offset of +/- 100 ns offset to the PTP master.

During normal operation, the static configuration of the sja1105 needs to be changed by the driver. In turn, this requires a switch reset, which temporarily disrupts Ethernet traffic and PTP synchronization. After a switch reset, the PTP synchronization offset may jump to a higher momentary range of +/- 2,500,000 ns. Below is a list of reset reasons in the sja1105 kernel driver:

- Enabling or disabling VLAN filtering, via the "`ip link`" command
- Enabling or disabling PTP timestamping
- Configuring the ageing time (which is done automatically by the kernel STP state machine when STP is active)
- Configuring the Time-aware Scheduler via the `tc-taprio` command
- Configuring the L2 policers (for MTU or for policing)

4.1.5.3 Synchronized 802.1Qbv demo

The objectives of this demonstration are the following:

- Synchronize the SJA1105 PTP clock using IEEE 802.1AS.
- Run the SJA1105 time-aware scheduler (802.1Qbv engine) based on the PTP clock.
- Create a small switched TSN network with a flow requiring deterministic latency. Prove the latency is not affected by interfering traffic.

In the topology described earlier in this chapter, the boards which require to be synchronized by PTP are hosts 1, 2, and the LS1021A-TSN board. Host 3 only generates iperf traffic, which is not time-sensitive.

The following commands are required to start PTP synchronization using the 802.1AS profile on host 1 and 2:

```
ptp4l -i eth0 -f /etc/linuxptp/gPTP.cfg -m
phc2sys -a -rr --transportSpecific 0x1 --step_threshold 0.0002
 --first_step_threshold 0.0002
```

Different output is expected on the two hosts. One becomes PTP grandmaster and shows the following logs:

- ptp4l:

```
ptp4l[13.067]: port 1: link up
ptp4l[13.104]: port 1: FAULTY to LISTENING on INIT_COMPLETE
ptp4l[16.113]: port 1: LISTENING to MASTER on
 ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[16.113]: selected local clock 00049f.fffe.05de06 as best
 master
ptp4l[16.113]: port 1: assuming the grand master role
ptp4l[16.692]: port 1: new foreign master 001f7b.fffe.630248-1
ptp4l[16.692]: selected best master clock 00049f.fffe.05f627
ptp4l[16.692]: port 1: assuming the grand master role
```

- phc2sys:

```
phc2sys[73.382]: eno0 sys offset        12 s2 freq   +2009
 delay   1560
phc2sys[74.382]: eno0 sys offset         2 s2 freq   +2003
 delay   1560
phc2sys[75.382]: eno0 sys offset       -18 s2 freq   +1983
 delay   1600
phc2sys[76.383]: eno0 sys offset        27 s2 freq   +2023
 delay   1600
phc2sys[77.383]: eno0 sys offset         7 s2 freq   +2011
 delay   1600
phc2sys[78.383]: eno0 sys offset       -18 s2 freq   +1988
 delay   1560
phc2sys[79.383]: eno0 sys offset        -8 s2 freq   +1993
 delay   1560
```

The other board becomes a PTP slave, as can be seen by the following logs:

- ptp4l:

```
ptp4l[68484.668]: rms   17 max   36 freq  +1613 +/-  15 delay
  737 +/-   0
ptp4l[68485.668]: rms    8 max   15 freq  +1622 +/-  11 delay
  737 +/-   0
ptp4l[68486.669]: rms   14 max   28 freq  +1643 +/-  13 delay
  737 +/-   0
ptp4l[68487.670]: rms   11 max   17 freq  +1650 +/-  10 delay
  738 +/-   0
ptp4l[68488.671]: rms   11 max   20 freq  +1633 +/-  15 delay
  738 +/-   0
ptp4l[68489.672]: rms    8 max   16 freq  +1640 +/-  11 delay
  737 +/-   0
ptp4l[68490.673]: rms   16 max   32 freq  +1640 +/-  23 delay
  737 +/-   0
ptp4l[68491.674]: rms   12 max   21 freq  +1622 +/-  13 delay
  737 +/-   0
```

```
ptp4l[68492.675]: rms   13 max   19 freq  +1648 +/-  13 delay
  738 +/-   0
ptp4l[68493.676]: rms   18 max   34 freq  +1668 +/-  15 delay
  737 +/-   0
```

- phc2sys:

```
phc2sys[68508.790]: CLOCK_REALTIME phc offset        10 s2
 freq    -342 delay   1600
phc2sys[68509.791]: CLOCK_REALTIME phc offset         2 s2
 freq    -347 delay   1560
phc2sys[68510.791]: CLOCK_REALTIME phc offset         9 s2
 freq    -339 delay   1600
phc2sys[68511.791]: CLOCK_REALTIME phc offset       -22 s2
 freq    -368 delay   1560
phc2sys[68512.791]: CLOCK_REALTIME phc offset       -19 s2
 freq    -371 delay   1560
phc2sys[68513.791]: CLOCK_REALTIME phc offset       -13 s2
 freq    -371 delay   1560
phc2sys[68514.791]: CLOCK_REALTIME phc offset        48 s2
 freq    -314 delay   1560
phc2sys[68515.792]: CLOCK_REALTIME phc offset        22 s2
 freq    -325 delay   1560
phc2sys[68516.792]: CLOCK_REALTIME phc offset        17 s2
 freq    -324 delay   1560
phc2sys[68517.792]: CLOCK_REALTIME phc offset       -29 s2
 freq    -365 delay   1560
```

The role of the LS1021A-TSN board is to relay the PTP time from the 802.1AS grandmaster to the slave. It acts as a slave on the port connected to the GM and as a master on the port connected to the other host.

```
[root] # journalctl -b -u ptp4l -f
-- Logs begin at Tue 2020-04-07 14:02:11 UTC. --
ptp4l[86640.528]: rms   10 max   23 freq -19731 +/-  11 delay
 737 +/-   0
ptp4l[86641.528]: rms    9 max   15 freq -19740 +/-  13 delay
 736 +/-   0
ptp4l[86642.529]: rms   12 max   19 freq -19757 +/-  10 delay
 737 +/-   0
ptp4l[86643.530]: rms    9 max   14 freq -19747 +/-  13 delay
 737 +/-   0
ptp4l[86644.530]: rms   13 max   22 freq -19733 +/-  15 delay
 736 +/-   0
ptp4l[86645.531]: rms    7 max   14 freq -19735 +/-   9 delay
 737 +/-   0
ptp4l[86646.532]: rms    7 max   13 freq -19735 +/-   9 delay
 737 +/-   0
ptp4l[86647.532]: rms   11 max   19 freq -19750 +/-  12 delay
 737 +/-   0
ptp4l[86648.533]: rms    6 max   14 freq -19745 +/-   8 delay
 737 +/-   0
ptp4l[86649.534]: rms    9 max   15 freq -19750 +/-  12 delay
 736 +/-   0
```

The above information can be interpreted as follows (only the last line is interpreted here):

- Because the default (implicit) summary_interval in `/etc/linuxptp.cfg` is 0 (print stats once per second) and the logSyncInterval required by 802.1AS is -3 (the

sync messages are sent at an interval of 1/8 seconds - 125 ms), this means that synchronization stats cannot be printed in full (for each packet) and are printed in an abbreviated form (there is no "offset" in the logs).

- The offset to the master has a root mean square value of 9 ms, with a maximum of 15 ns in the past 1 second.
- The frequency correction required to synchronize to the GM was on average -19750 parts per billion (ppb). If the frequency adjustment exceeds a certain sanity threshold (depending on kernel driver), ptp4l may display "clockcheck" warnings and stop synchronization. This can be sometimes remedied manually by running the following command to reset the PTP clock frequency adjustment to zero:

```
phc_ctl /dev/ptp0 freq 0
```

- The measured path delay (MAC to MAC propagation delay for ~70 bytes frames at 1 Gbps) between its device and its link partner is exactly 736 ns.

The clock distribution tree in this network is as follows: the system clock of the PTP GM (for example, Host 1) disciplines its PTP hardware clock (/dev/ptp0), using phc2sys. Over Ethernet, the PTP GM disciplines the SJA1105 PHC, which disciplines the PTP slave (e.g. Host 2). On the slave host, the phc2sys process runs in the reverse direction, disciplining the system clock (CLOCK_REALTIME) to the PTP hardware clock (/dev/ptp0).

Note: *While using the LS1021A-TSN board as a gPTP GM for this scenario (in place of Host 1), there is no battery-backed RTC. Therefore, there is no persistent source of time on-board. One has to rely on the NTP service (ntpd.service) to provide time, otherwise a time in 1970 is relayed into the PTP network.*

Note: *While using phc2sys on the slave host, you must manually ensure that other daemons such as ntpd in the system do not attempt to do the same thing. phc2sys attempts to discipline CLOCK_REALTIME. in case this is not ensured, there might be access conflicts between phc2sys and the other daemon, and phc2sys would keep displaying clockcheck warning messages.*

Install the following schedule into the sja1105 port egressing towards Host 2:

```
tc qdisc add dev swp2 parent 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 0 \
      sched-entry S 80  50000 \
      sched-entry S 40  50000 \
      sched-entry S 3f 300000 \
      flags 2
```

The base-time of 0 indicates the phase offset of the network schedule. This time corresponds to Jan 1st 1970, but it is automatically advanced into an equivalent time into the immediate PTP future (it is advanced by an integer number of cycle-time nanoseconds).

The cycle-time in this example is not provided explicitly, but it is calculated as the sum of the durations of all gate events: 400 microseconds (us).

The schedule at the egress of swp2 is divided as follows:

- 50 µs for PTP traffic (S 80). The traffic class assignment of 7 for link-local management traffic (STP, PTP, etc) is fixed to 7 at driver level and is not user configurable at this time.
- 50 µs for traffic class 6 (S 40). The latency-sensitive traffic generator will be injecting into this window.
- 300 µs for all other traffic classes 0-5 (S 3f).

Enabling QoS classification on the sja1105 switch based on VLAN PCP is done by running:

```
ip link set dev br0 type bridge vlan_filtering 1
```

First, start the receiver for latency-sensitive traffic on Host 2. This process waits for connections from the sender and then transmits its statistics to it.

```
ip addr add 192.168.1.2/24 dev eth0
isochron rcv --interface eth0 --quiet
```

Start the sender on Host 1 as follows:

```
ip addr add 192.168.1.1/24 dev eth0
isochron send --interface eth0 --dmac 00:04:9f:05:de:06 --
priority 6 --vid 0 \
        --base-time 0 --cycle-time 400000 --shift-time 50000 --
advance-time 90000 \
        --num-frames 10000 --frame-size 64 --client 192.168.1.2
 --quiet
```

The log should look as follows:

```
Base time 0.000040000 is in the past, winding it into the
 future
       Now: 1586282691.751150218
 Base time: 1586282691.751160000
Cycle time: 0.000400000
Collecting receiver stats
Summary:
Path delay: min 4329 max 4444 mean 4387.987 stddev 24.508
HW TX deadline delta: min -65238 max -18938 mean -59707.395
 stddev 1371.995
SW TX deadline delta: min -33528 max 25058 mean -28221.001
 stddev 1844.235
HW RX deadline delta: min -60874 max -14529 mean -55319.408
 stddev 1372.222
SW RX deadline delta: min -43398 max 130659 mean -38212.966
 stddev 2514.592
HW TX deadline misses: 0 (0.000%)
SW TX deadline misses: 1 (0.010%)
```

The following clarifications are necessary:

- The destination MAC is that of Host 2's interface eth0
- The sent packets have a VLAN tag with VID 0 and PCP 6. Since they are priority-tagged (802.1p) the sja1105 switch ports would accept these packets without any "`bridge vlan add vid 0 dev swp3`" command.
- The `isochron` program sends a number of 10000 frames, at an interval of 400 µs. The base-time is the same as on the sja1105 egress port swp2, but it is shifted with 50

µs to the right, in order to align with the beginning of traffic class 6's window (which is the second timeslot in the schedule). The packet transmission deadlines are therefore at (base-time + shift-time + N * cycle-time).

- Packets must in fact be transmitted earlier than the TX deadline, in order to compensate for scheduling latencies in the Linux kernel and the actual propagation delay of the packet. So the isochron program sleeps until 90 us in advance of the next deadline.
- By "winding the base time into the future", one understands the process by which the original base time (0) is incremented by the smallest number N of cycles such that it becomes greater than the current PTP time (1586282691.751150218). In this case, the new base-time is 1586282691.751160000.
- For each packet, the sender collects 2 TX timestamps: one hardware and one software. The receiver also collects two timestamps. These timestamps are not printed to the console because the --quiet option was specified.
- Correlation between timestamps at the sender and at the receiver is done through a secondary socket. The receiver waits for connections on TCP port 5000, and transmits its log to the sender, which correlates with its own log by using a key formed out of {sequence number, scheduled TX time (deadline)}. Both these values are embedded into the packet payload. If the --client option is omitted, the statistics correlation is not performed. This TCP socket is the only reason for which IP communication is necessary in this network.
- The path delay is calculated as the delta between the RX hardware timestamp at the receiver and the TX hardware timestamp at the sender.
- Each "deadline delta" is calculated as the difference between the timestamp and the scheduled TX time of this packet. The HW TX deadline delta should always be negative, as that indicates the packets were sent before the scheduled TX time has expired. The SW TX timestamps are taken after the HW TX timestamps in this case, so their meaning is less relevant for this driver. The RX deadline deltas will become relevant once the 802.1Qbv schedule is installed on the sja1105 switch port.

The above log was taken with no 802.1Qbv schedule active on the sja1105 port and no background traffic. After starting background traffic:

```
# Host 2
iperf3 -s > /dev/null &
sysctl -w kernel.sched_rt_runtime_us=-1
chrt --fifo 90 isochron rcv -i eth0 --quiet
# Host 3
ip addr add 192.168.1.3/24 dev eth0
iperf3 -c 192.168.1.2 -t 48600
Connecting to host 10.0.0.112, port 5201
[  5] local 10.0.0.113 port 60360 connected to 10.0.0.112 port
 5201
[ ID] Interval           Transfer     Bitrate         Retr
 Cwnd
[  5]   0.00-1.00   sec   105 MBytes   878 Mbits/sec    0
 489 KBytes
[  5]   1.00-2.00   sec   102 MBytes   859 Mbits/sec    0
 513 KBytes
[  5]   2.00-3.00   sec   102 MBytes   858 Mbits/sec    0
 513 KBytes
[  5]   3.00-4.00   sec   101 MBytes   851 Mbits/sec    0
 513 KBytes
[  5]   4.00-5.00   sec   102 MBytes   860 Mbits/sec    0
 539 KBytes
```

a re-run of the isochron traffic generated by Host 1 looks as follows:

```
chrt --fifo 90 isochron send -i eno0 -d 00:04:9f:05:de:06 -
p 6 -v 0 -b 0 -S 50000 -c 400000 -a 90000 -n 10000 -s 64 -C
 10.0.0.112 -q
Base time 0.000040000 is in the past, winding it into the
 future
       Now: 1586286409.635121693
 Base time: 1586286409.635160000
Cycle time: 0.000400000
Collecting receiver stats
Summary:
Path delay: min 4314 max 16774 mean 9725.688 stddev 3919.150
HW TX deadline delta: min -64273 max -8538 mean -59894.931
 stddev 1467.284
SW TX deadline delta: min -33286 max 37575 mean -28498.114
 stddev 2006.546
HW RX deadline delta: min -58924 max -904 mean -50169.243
 stddev 4183.042
SW RX deadline delta: min -52757 max 1109472 mean -29436.032
 stddev 23537.847
HW TX deadline misses: 0 (0.000%)
SW TX deadline misses: 4 (0.040%)
```

It can be seen that the path delay variance has increased due to the prolonged wait of packets until MTU-sized packets generated by iperf3 have finished transmission.

Finally, installing the 802.1Qbv schedule on the switch has effects upon all statistics calculated by isochron:

```
chrt --fifo 90 isochron send -i eno0 -d 00:04:9f:05:de:06 -
p 6 -v 0 -b 0 -S 50000 -c 400000 -a 90000 -n 10000 -s 64 -C
 10.0.0.112 -q
Base time 0.000040000 is in the past, winding it into the
 future
       Now: 1586286689.223100936
 Base time: 1586286689.223160000
Cycle time: 0.000400000
Collecting receiver stats
Summary:
Path delay: min 14199 max 65684 mean 61357.368 stddev 1494.831
HW TX deadline delta: min -64128 max -12643 mean -59822.445
 stddev 1494.557
SW TX deadline delta: min -33616 max 25621 mean -28448.709
 stddev 1974.185
HW RX deadline delta: min 1476 max 2041 mean 1534.924 stddev
 24.822
SW RX deadline delta: min 5243 max 1122800 mean 21040.814
 stddev 16752.155
HW TX deadline misses: 0 (0.000%)
SW TX deadline misses: 5 (0.050%)
```

The path delay has increased, but that is because now it contains the time spent by the packets blocked on the switch waiting for gate 6 to open.

The HW RX deadline delta now has a new meaning, since in the last example (with 802.1Qbv enabled on the switch), the gate acts as a barrier and eliminates the jitter in HW TX timestamps, which is induced by scheduling latencies in the sender's operating system. Generally speaking, the jitter of the sender is eliminated by the first switch upon

packet admission into the TSN network. The effect is that the receiver sees a packet stream with low jitter.

The path delay can be reduced by decreasing the advance time. It is configured in such a way that the packets arrive on the switch prior to the gate opening, which depends on the jitter of the sender. Minimizing the TX jitter is outside the scope of this demonstration.

4.2 GenAVB/TSN stack

4.2.1 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 and i.MX 8M Plus 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 8M Mini, and i.MX 6ULL SoCs (using hardware support if available).

4.2.1.1 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.

4.2.1.2 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.

4.2.1.3 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.

4.2.1.4 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.

4.2.1.5 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.

4.2.1.6 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 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.

The TSN stack supports and is enabled in the following Yocto Real-time Edge machines:

- imx93evk
- imx8mp-lpddr4-evk
- ls1028ardb

The AVB Endpoint stack supports and is enabled in the following Yocto Real-time Edge machines:

- imx6ull14x14evk
- imx8mm-lpddr4-evk
- imx8mp-lpddr4-evk

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

4.2.1.7 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.

4.2.1.8 TSN Endpoint example application

The TSN example application provides example code and re-usable middleware exercising the GenAVB/TSN API. It is used to exercise and verify the real time behavior of the local system as well as TSN properties of the network between endpoints.

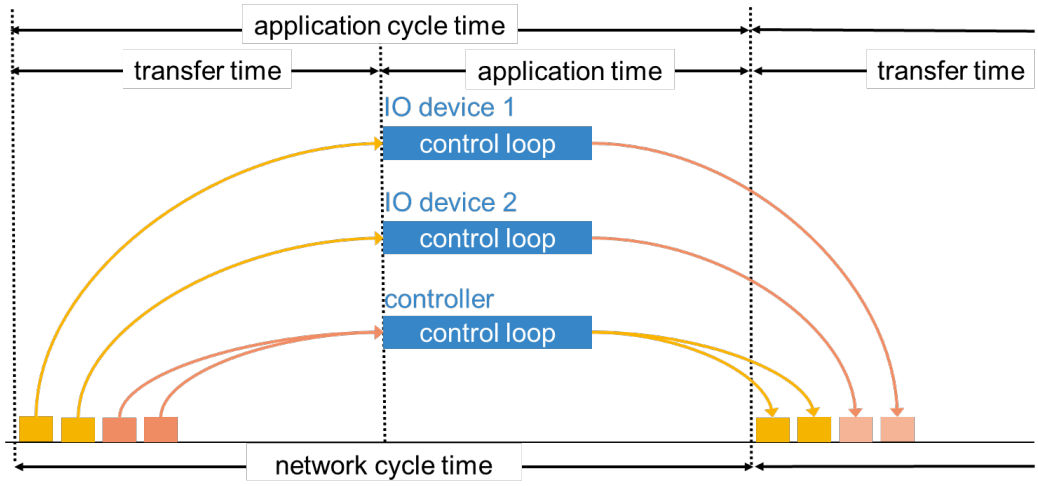


Figure 58. TSN application cycle

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 above figure). Currently the cycle is configured with a period of 2ms, and periods as low as 1ms 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.

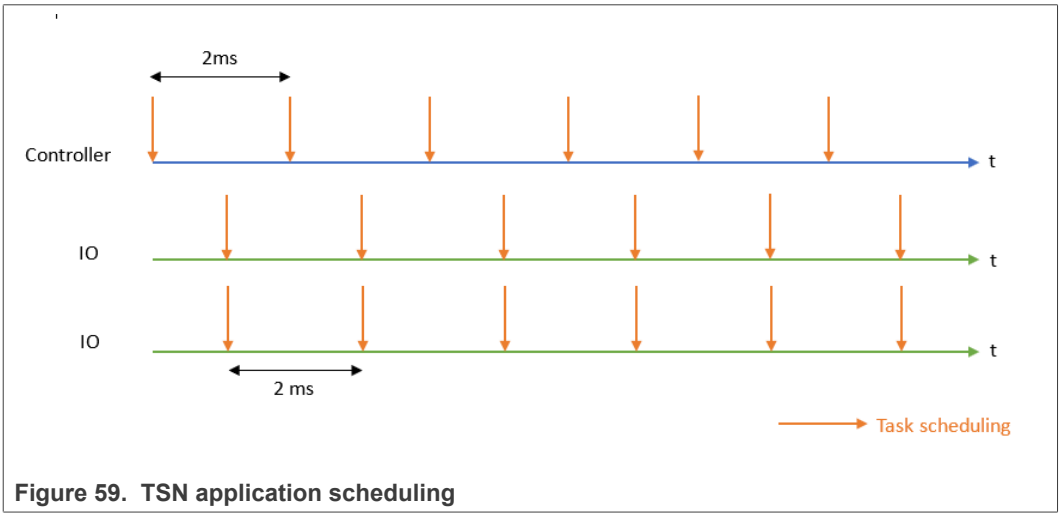


Figure 59. TSN application scheduling

As shown in the figure above, 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, a 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.

4.2.2 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
```

4.2.3 Use cases description

4.2.3.1 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.

4.2.3.2 gPTP Bridge

LS1028ARDB can be used as a generic time-aware bridge, connected to other time-aware end stations or bridges.

By default, LS1028ARDB does not forward packets if no bridge interface is configured under Linux. Enabling bridge interface is dependent on the board used. For example, how to configure bridge interface on LS1028ARDB is described in section Section 4.1.4.2.1.

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 4.2.4.2.2.

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 4.2.5.2.

4.2.3.3 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
...
gptp_stats_dump : Port(0) domain(1,20): Role: Slave Link:
 Up AS_Capable: Yes neighborGptpCapable: Yes DelayMechanism:
 COMMON_P2P
```

Role status can also take the value *Master* depending on *Grandmaster Parameters* described in section Section 4.2.4.2.2.

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 4.2.5.1.

4.2.3.4 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.

4.2.3.4.1 Requirements

The reference setup for gPTP multiple domains is made of:

- Two gPTP endpoints (i.MX 8MPlus LPDDR4 EVK or i.MX 93 EVK connected through TSN interface eth1: dwmac): EP1-DUT and EP2-DUT
- One gPTP bridge (LS1028ARDB): BR-DUT

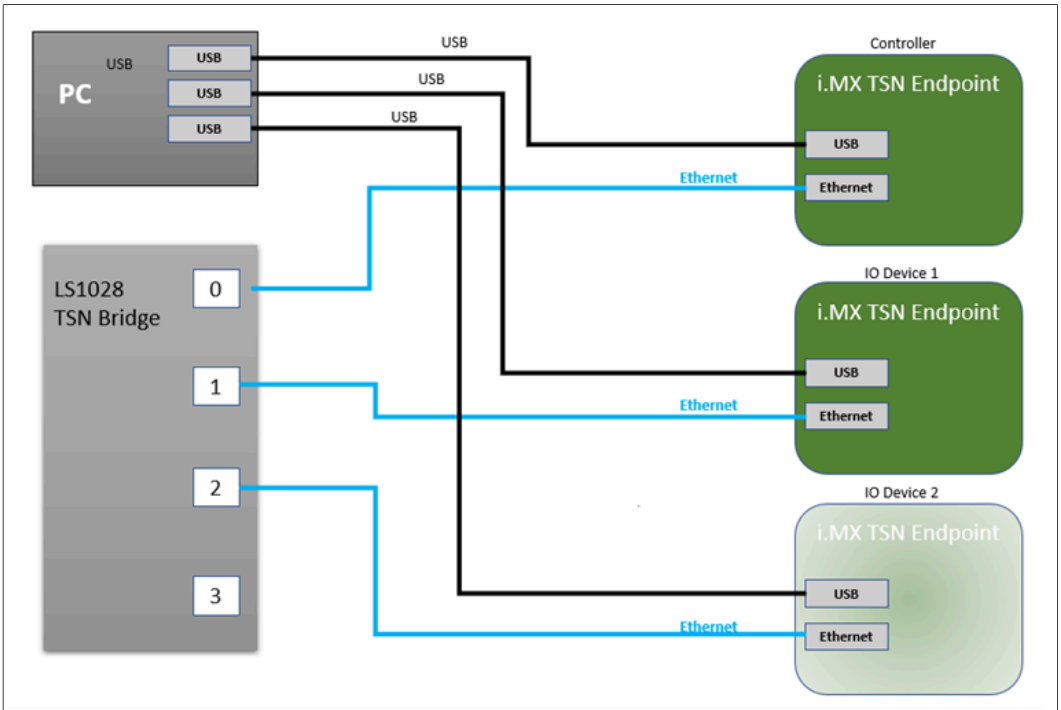


Figure 60. gPTP multiple domains setup

4.2.3.4.2 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 priority1=246, EP1-DUT and EP2-DUT priority1=248).*
- *On Domain 1, EP1-DUT is the GrandMaster with the highest priority (lowest value) among all devices in the domain (BR-DUT priority1=246, EP1-DUT priority1=245 and EP2-DUT priority1=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).*

4.2.3.4.3 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
```

4.2.3.5 AVB Bridge

This use case illustrates an AVB Bridge (mixing gPTP and SRP stack) with other AVB Endpoints

4.2.3.5.1 Requirements

- Two AVB endpoints
- One AVB bridge (LS1028ARDB)

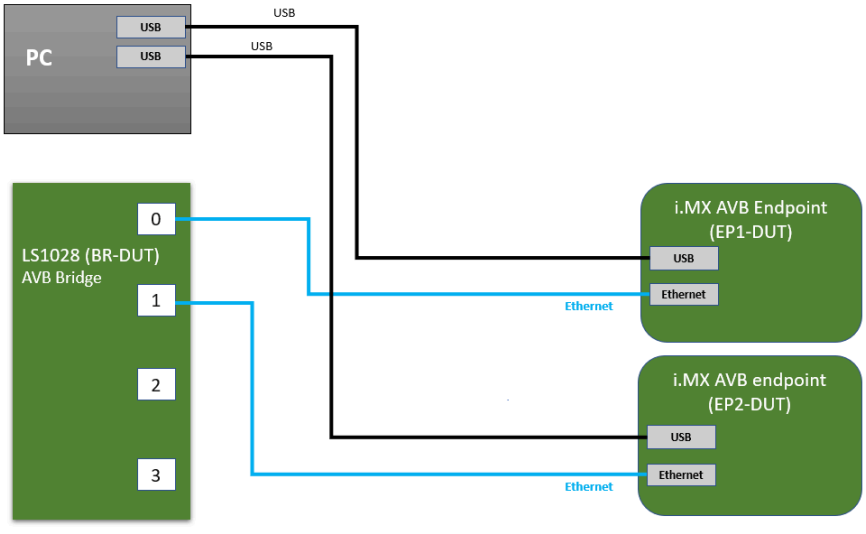


Figure 61. AVB Bridge setup

4.2.3.5.2 AVB network configuration

This topic describes AVB configuration.

4.2.3.5.2.1 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 51. Priority to traffic class mapping

Priority	0	1	2	3	4	5	6	7
Traffic Class	1	0	6	7	2	3	4	5

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 4.2.3.5.3 for the bridge PCP mapping configuration.

4.2.3.5.2.2 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 4.2.3.5.3 for the bridge qdisc configuration.

4.2.3.5.2.3 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 4.2.3.5.3 for the skb priorities remapping configuration.

4.2.3.5.2.4 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 4.2.3.5.3 for the bridge vlan configuration.

4.2.3.5.3 Setup preparation

This has two steps described in the following sections.

4.2.3.5.3.1 Bridge configuration

This configuration needs to 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**

- Execute the automated configuration script and start the AVB bridge stack:

```
# avb-bridge.sh
# avb.sh start
```

- The bridge can be manually configured using the following commands:
- Setup bridge forwarding:

```
# ip link set dev eno2 up
# 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
```

- Establish the PCP to QoS mapping for every port on the bridge:

```
#
 pcp_to_qos_map=([0]="1" [1]="0" [2]="6" [3]="7" [4]="2" [5]="3" [6]="4"
 \
avb_ports="swp0 swp1 swp2 swp3"; \
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
```

- Configure the qdiscs and shapers, with the correct handles, for every external port:

```
#
 pcp_to_qos_map=([0]="1" [1]="0" [2]="6" [3]="7" [4]="2" [5]="3" [6]="4"
 \
avb_ports="swp0 swp1 swp2 swp3"; \
for port in $avb_ports; do \
    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 0 ; \
    tc qdisc replace dev $port handle 0x9007 parent 100:8 \
                            cbs locredit -2147483646 hicredit
 2147483647 sendslope -1000000 idleslope 0 offload 0 ; \
    tc qdisc replace dev $port handle 0x9006 parent 100:7 \
                            cbs locredit -2147483646 hicredit
 2147483647 sendslope -1000000 idleslope 0 offload 0 ; \
done
```

Attention:

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.*

- Setup skb priorities remapping for every external port:

```
# 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
```

- Enable Vlan filtering, set the correct Vlan IDs and disable multicast flooding, for every external port:

```
# 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
```

- 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
```

4.2.3.5.3.2 GenAVB/TSN stack configuration

This configuration needs to be done once and is saved accross reboots.

For a proper gPTP operation with AVB endpoints, the gPTP stack needs to compensate for PHY delay in PTP timestamps:

In the `/etc/genavb/fgptp-br.cfg`, apply the settings (rxDelayCompensation and txDelayCompensation) described in Table 63 on all bridge ports.

Attention: *The PHY Delay Compensation Values in Table 63 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.*

Future releases shall have proper compensation values for each supported link speed.

4.2.3.5.4 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

4.2.3.5.4.1 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:

```
gptp_stats_dump: Port(0) domain(0, 0): Role: Master   Link: Up
  asCapable: Yes neighborGptpCapable: Yes DelayMechanism: P2P
...
gptp_stats_dump: Port(1) domain(0, 0): Role: Master   Link:
 Up   asCapable: Yes neighborGptpCapable: Yes DelayMechanism:
 COMMON_P2P
```

Refer to Section 4.2.3.2, for more details on gPTP Bridge operation.

4.2.3.5.4.2 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:

```
# 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
 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 2 1 2 0
 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:5 bands 3 priomap 1 2 2 2 1 2 0
 0 1 1 1 1 1 1 1 1
```

```
qdisc pfifo_fast 0: parent 100:4 bands 3 priomap 1 2 2 2 1 2 0
 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:3 bands 3 priomap 1 2 2 2 1 2 0
 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:2 bands 3 priomap 1 2 2 2 1 2 0
 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:1 bands 3 priomap 1 2 2 2 1 2 0
 0 1 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:

```
# bridge mdb show
dev br0 port swp0 grp 91:e0:f0:00:fe:11 permanent offload vid
 2
```

4.2.3.6 TSN endpoint sample application

4.2.3.6.1 Requirements

- Two TSN endpoints (i.MX 8MPlus LPDDR4 EVK or i.MX 93 EVK connected through TSN interface eth1: dwmac, or optionally an i.MX RT1170 EVK)
- One TSN bridge (LS1028ARDB)

Note: *The second IO Device is optional.*

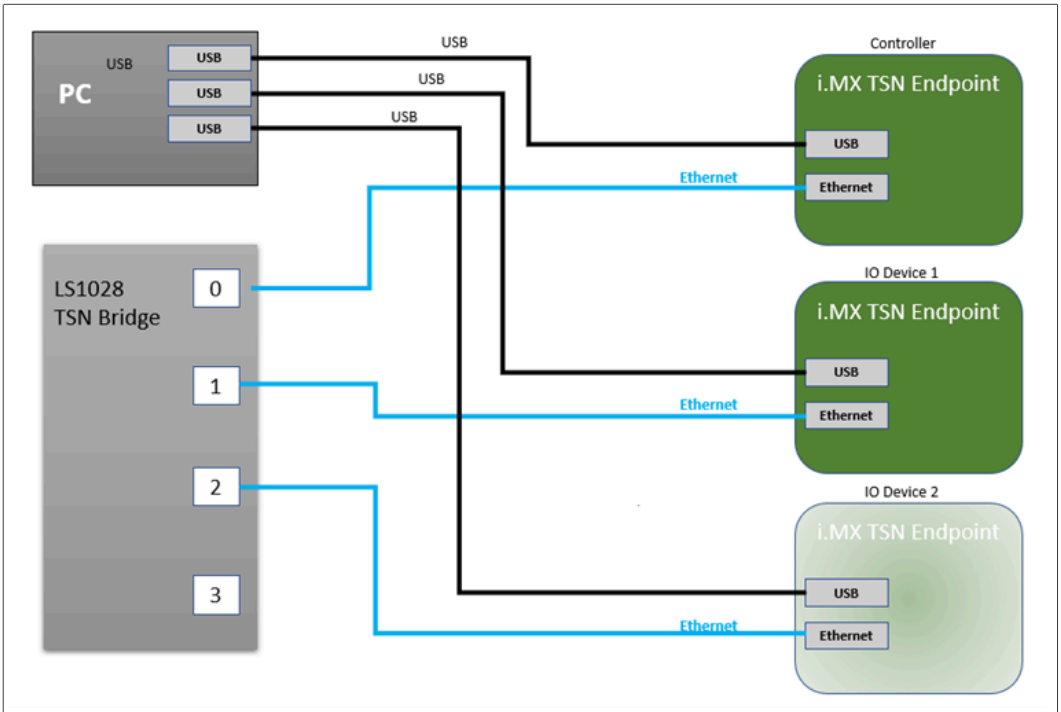


Figure 62. TSN endpoint sample application setup

4.2.3.6.2 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 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.

4.2.3.6.3 TSN network configuration

This topic describes TSN configuration.

4.2.3.6.3.1 Streams

The stream details can be used for analysis and also for computing scheduled traffic timings.

Table 52. TSN streams definition

Stream No	Source	Destination	Unicast / Multicast	Destination MAC Address	Vlan ID	Vlan PCP	Frame Length[1] (bytes)
Stream1	Controller	IO device(s)	Multicast	91:e0:f0:00:fe:70	2	5	84
Stream2	IO device 1	Controller	Multicast	91:e0:f0:00:fe:71	2	5	84
Stream3	IO device 2	Controller	Multicast	91:e0:f0:00:fe:80	2	5	84

[1] The frame length includes inter frame gap, preamble, start of frame and CRC (can be used as is for timing calculations)

4.2.3.6.3.2 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 59, 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.

4.2.3.6.4 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.

4.2.3.6.4.1 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:

```
# tsn-app-setup.sh eth1
```

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. Restore default coalescing setting and disable it only on TSN traffic queue (HW queue 2):

```
# ethtool -C eth1 rx-usecs 153 tx-usecs 1000 tx-frames 25
# ethtool --per-queue eth1 queue_mask 0x04 --coalesce rx-
usecs 16 tx-usecs 10000 tx-frames 1
```

6. Start the TSN demo application using the following command:

```
avb.sh start
```

4.2.3.6.4.2 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. The system configuration needed for the `tsn-app` can be done (after setting the correct PROFILE) using the following command:

```
tsn-app-setup.sh eth1
```

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:

```
avb.sh start
```

4.2.3.6.4.3 Preparing the Bridge

Refer to section Section 4.1.2 and Section 4.1.4.2.3.2 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:

```
# ip link set dev eno2 up
# 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:

```
# 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:

```
# tsn.sh start
```

4. Setup scheduled traffic (see above)

```
# 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 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time 1500000 \
        sched-entry S 0x20 20000 \
        sched-entry S 0xdf 1980000 \
        flags 0x2
# 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 500000 \
        sched-entry S 0x20 20000 \
        sched-entry S 0xdf 1980000 \
        flags 0x2
# tc qdisc replace dev swp2 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 500000 \
        sched-entry S 0x20 20000 \
        sched-entry S 0xdf 1980000 \
        flags 0x2
```

4.2.3.6.4.4 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:
   a. FreeOpcUa: client with a Qt GUI interface.
      Can be found here: http://freeopcua.github.io/
   b. opcua-commander: CLI alternative based nodejs node-opcua stack. Can be found here:
      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.

4.2.3.6.5 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.

4.2.3.6.5.1 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 4.2.3.3 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).

4.2.3.6.5.2 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 4.2.5.4 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.

4.2.3.6.5.3 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(0xaaab06ed74b0) 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(0xaaab06ed7910) processing time min 23400 mean 29185 max
 59100 rms^2 857707540 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
- stddev^2 less than 3000

```
stats(0x419a28) traffic latency min 503417 mean 503503 max
 503637 rms^2 253515981945 stddev^2 2004 absmin 503397 absmax
 504337
```

4.2.3.6.5.4 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:

```
# ifconfig eth1 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 -l 64 -i 2
 -t 100 &
# taskset b iperf3 -p 5203 -c 192.168.1.10 -u -b 0 -l 64 -i 2
 -t 100 &
```

```
# taskset b iperf3 -p 5204 -c 192.168.1.10 -u -b 0 -l 64 -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):

	min	mean	max	stddev^2
Sched err (controller)	21	29	41	
Processing time (controller)	47	80	260	
Traffic latency (controller and IO device)	503	503	503	<3000

4.2.3.6.5.5 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:

```
# ip link add link eth1 name eth1.5 type vlan id 5
# ifconfig eth1.5 192.168.5.80 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):

```
# ip link add link ethX name ethX.5 type vlan id 5
# ifconfig ethX.5 192.168.5.10 up
# 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):

	min	mean	max	stddev^2
Sched err (controller)	9	13	26	
Processing time (controller)	25	33	70	
Traffic latency (controller and IO device)	503	503	503	<130000

4.2.3.6.5.6 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:

```
# avb.sh stop
```

2. Edit the application configuration file:

```
# vi /etc/genavb/apps-tsn-network-controller.cfg
```

or for an IO device:

```
# 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 controller -p
 1000000"
```

4. Update the traffic schedule using 'tc' command:

```
# tc qdisc del dev eth1 root
#tc qdisc replace dev eth1 root taprio \
num_tc 5 \
map 0 0 1 1 2 2 3 4 0 0 0 0 0 0 0 0 \
queues 1@0 1@1 1@2 1@3 1@4 \
base-time 250000 \
sched-entry S 0x4 4000 \
sched-entry S 0x1b 996000 \
flags 0x2
```

5. Restart the tsn-app application:

```
# avb.sh start
```

On the IO device(s):

1. Stop the application if it was already running:

```
# avb.sh stop
```

2. Edit the application configuration file:

```
# 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:

```
# tc qdisc del dev eth1 root
#tc qdisc replace dev eth1 root taprio \
num_tc 5 \
```

```
map 0 0 1 1 2 2 3 4 0 0 0 0 0 0 0 0 \
queues 1@0 1@1 1@2 1@3 1@4 \
base-time 750000 \
sched-entry S 0x4 4000 \
sched-entry S 0x1b 996000 \
flags 0x2
```

5. Restart the tsn-app application:

```
# avb.sh start
```

On the bridge, update the Qbv schedule on all ports:

```
# 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 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time 750000 \
        sched-entry S 0x20 20000 \
        sched-entry S 0xdf 980000 \
        flags 0x2
# 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 250000 \
        sched-entry S 0x20 20000 \
        sched-entry S 0xdf 980000 \
        flags 0x2
# tc qdisc replace dev swp2 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 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.

4.2.3.6.5.7 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 or i.MX 93 EVK board to use `AF_XDP` sockets.

1. Stop the application and TSN stack if they were already running:

```
# avb.sh stop_all
```

2. Edit the application configuration file:

```
# vi /etc/genavb/apps-tsn-network-controller.cfg
```

3. To enable AF_XDP mode, replace the line:

```
CFG_EXTERNAL_MEDIA_APP_OPT="-m network_only -r controller"
```

With:

```
CFG_EXTERNAL_MEDIA_APP_OPT="-m network_only -r controller -x"
```

4. Attach the XDP program to the eth1 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):

```
# ip l set dev eth1 xdp obj /lib/firmware/genavb/genavb-
xdp.bin
```

5. Restart the tsn-app application in AF_XDP mode:

```
# 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 53. Timing statistics without any concurrent traffic

	min	mean	max
Sched err (controller)	6	7	16
Processing time (controller)	10	13	19
Total time (controller)	16	20	33

Table 54. Timing statistics with TX best-effort traffic

	min	mean	max
Sched err (controller)	18	25	51
Processing time (controller)	21	26	52
Total time (controller)	42	51	108

Table 55. Timing statistics with RX best-effort traffic

	min	mean	max
Sched err (controller)	7	9	34
Processing time (controller)	8	11	25

Table 55. Timing statistics with RX best-effort traffic...*continued*

	min	mean	max
Total time (controller)	15	21	55

4.2.3.6.5.8 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:

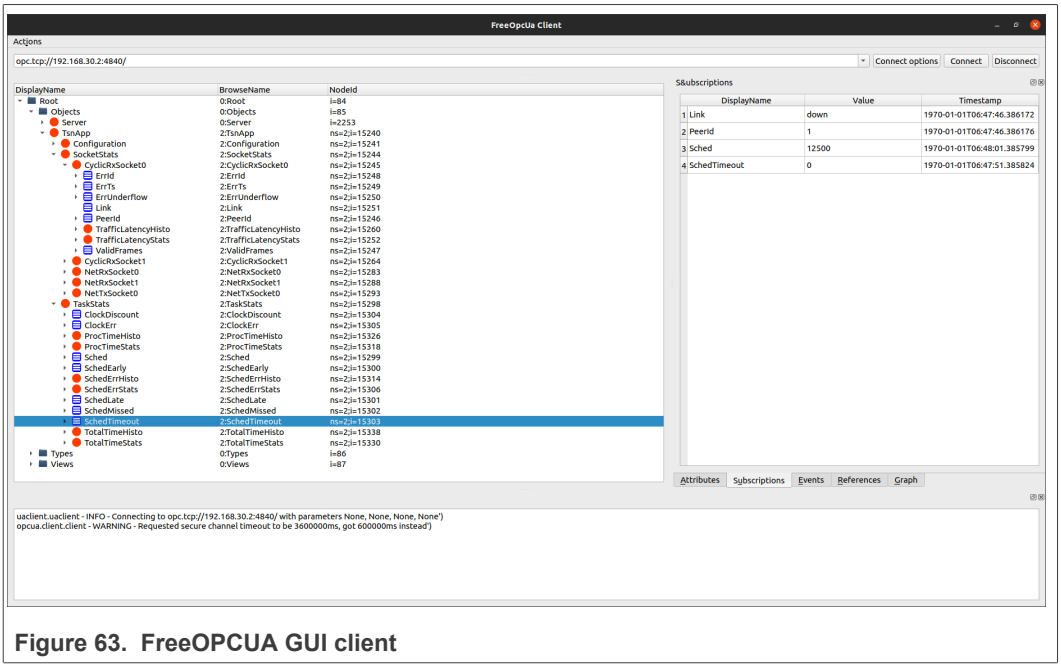


Figure 63. FreeOPCUA GUI client

4.2.4 Configuration files

4.2.4.1 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 56. Logical ports

This section lists network interface names.

Currently endpoint package supports a single endpoint and bridge package a single bridge (with up to 5 ports).

Name	Key	Default value	Description
Endpoint interface	`endpoint`	eth0	Endpoint network interface name. Only valid for endpoint package, otherwise should be set to “off”
Bridge 0 interfaces	`bridge_0`	SJA1105P_p0, SJA1105P _p1, SJA1105P _p2, SJA1105P_ p3, SJA1105P_p4*	Bridge 0 network interface names (comma separated). Only valid for bridge package, otherwise should be set to “off”

Table 57. 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).*

Name	Key	Default value	Description
Endpoint gPTP domain 0 target clock	`endpoint_gptp_0`	`/dev/ptp0`	Endpoint clock for gPTP domain 0 target clock. Only valid for endpoint package.
Endpoint gPTP domain 1 target clock	`endpoint_gptp_1`	`sw_clock`	Endpoint clock for gPTP domain 1 target clock. Only valid for endpoint package.
Endpoint local clock	`endpoint_local`	`/dev/ptp0`	Endpoint clock for the local clock. Only valid for endpoint package.
Bridge gPTP domain 0 target clock	`bridge_gptp_0`	`sw_clock`	Bridge clock for gPTP domain 0 target clock. Only valid for bridge package.
Bridge gPTP domain 1 target clock	`bridge_gptp_1`	`sw_clock`	Bridge clock for gPTP domain 1 target clock. Only valid for bridge package.
Bridge local clock	`bridge_local`	`/dev/ptp1`	Bridge clock for the local clock. Only valid for bridge package.

4.2.4.2 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`)

4.2.4.2.1 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 58. General parameters

General configuration parameters[1]

Name	Key	Default value	Range	Description
Profile	profile	“standard”	"standard" or "automotive"	Set fgptp main profile. "standard" - IEEE 802.1AS specs, "automotive" - AVnu automotive profile
Grandmaster ID	gm_id	"0x0001f2fffe0025fe"	64bits EUI format	Set static grandmaster ID in host order (used by automotive profile, ignored in case of standard profile)
Domains	domain_number	0: for default domain -1: for domains different from 0	-1 to 127	Disable (-1) or assign a gPTP domain number to a domain instance.
802.1AS-2011 mode	force_2011	no	"no" or "yes"	Set to "yes" to force 802. 1AS-2011 standard. "no" to enable 802.1AS-2020 full support.
Log output level	log_level	info	crit, err, init, info, or dbg	Set this configuration to dbg to enable debug mode
Reverse sync feature control	reverse_sync	0	0 or 1	Set to 1 to enable reverse sync feature.
Reverse sync feature interval	reverse_sync_interval	112	32 to 10000	Reverse sync transmit interval in ms units
Neighbor propagation delay threshold	neighborPropDelayThresh	800	32 to 10000000	Neighbor propagation delay threshold expressed in ns
Statistics output interval	statsInterval	10	0 to 255	Statistics output interval expressed in seconds. Use 0 to disable statistics

[1] For domain instances other than 0, only domain_number is configurable in this section.

4.2.4.2.2 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.

Table 59. Grandmaster parameters

Grandmaster capabilities parameters[1]

Name	Key	Default value	Range	Description
Grandmaster capable setting	gmCapable	1	0 or 1	Set to 1 if the device has grandmaster capability. Ignored in automotive profile if the port is SLAVE.
Grandmaster priority1 value	priority1	248 for AED-E and 246 for AED-B	0 to 255	Set the priority1 value of this clock
Grandmaster priority2 value	priority2	248	0 to 255	Set the priority2 value of this clock
Grandmaster clock class value	clockClass	248	0 to 255	Set the class value of this clock
Grandmaster clock accuracy value	clockAccuracy	0xfe	0x0 to 0xff	Set the accuracy value of this clock
Grandmaster variance value	offsetScaledLogVariance	17258	0x0 to 0xffff	Set the offset scaled log variance value of this clock

[1] The parameters in this section are configurable for all supported domains.

4.2.4.2.3 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 nvram database specified by nvram_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 nvram 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 60. Automotive parameters

Name	Key	Default value	Value & Range	Description
Pdelay mode	neighborPropDelay_mode	static	'static', 'silent' or 'standard'	Defines pdelay mechanism used

Table 60. Automotive parameters*...continued*

Name	Key	Default value	Value & Range	Description
Static pdelay value	initial_neighborPropDelay	250	0 to 10000	Predefined pdelay value applied to all ports. Expressed in ns.
Static pdelay sensitivity	neighborPropDelay_sensitivity	10	0 to 1000	Amount of ns between two pdelay measurements required to trigger a change indication. Expressed in ns.
Nvram file name	nvram_file	/etc/genavb/ fgptp.nvram		Path and nvram file name.

4.2.4.2.4 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 61. Timing parameters

Name	Key	Default value	Value and Range	Description
Initial pdelay request interval value	initialLogPdelayReqInterval	0	0 to 3	Set pdelay request initial interval between the sending of successive Pdelay_Req messages. Expressed in log2 unit (default 0 -> 1s).
Initial sync interval value	initialLogSyncInterval	-3	-5 to 0	Set sync transmit initial interval between the sending of successive Sync messages. Expressed in log2 unit (default -3-> 125ms).
Initial announce interval value	initialLogAnnounceInterval	0	0 to 3	Set initial announce transmit interval between the sending of successive Announce messages. Expressed in log2 unit (default 0 -> 1s).
Operational pdelay request interval value	operLogPdelayReqInterval	0	0 to 3	Set pdelay request transmit interval used during normal operation state. Expressed in log2 unit (default 0 -> 1s).

Table 61. Timing parameters*...continued*

Name	Key	Default value	Value and Range	Description
Operational sync interval value	operLogSyncInterval	-3	-5 to 0	Set sync transmit interval used during normal operation state. Expressed in log2 unit (default -3 -> 125ms).

4.2.4.2.5 PORTn

This section describes the settings per port where `n` represents the port index starting at `n=1`.

Table 62. Port related parameters

Name	Key	Default value	Value & Range	Description
Port role	portRole	disabled	'slave', 'master', 'disabled'	Static port role (ref. 802.1AS-2011, section 14.6.3, Table 10-1), applies to "automotive" profile only.
Ptp port enabled	ptpPortEnabled	1	0 or 1	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).
RX timestamp compensation	rxDelayCompensation	0	min=-100000 max=100000 (in ns units)	Compensation delay subtracted from receive timestamps.
TX timestamp compensation	txDelayCompensation	0	min=-100000 max=100000 (in ns units)	Compensation delay added to transmit timestamps.
Delay Mechanism	delayMechanism	P2P	'P2P' or 'COMMON_P2P'	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.

The following table lists the recommended Rx and Tx compensation values to be applied to the supported NXP boards for optimized gPTP synchronization.

Table 63. PHY Delay Compensation Values

Board Type	rxDelayCompensation	txDelayCompensation
LS1028ARDB	-274	349
I.MX 8M Plus EVK	-569	184

4.2.4.3 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 required an update to match the system configuration.

Table 64. SRP General

This section lists general SRP stack component parameters.

Name	Key	Default value	Range	Description
Log output level	log_level	info	crit, err, init, info, or dbg	Log level for the SRP stack component.

Table 65. MSRP

This section lists MSRP parameters.

Name	Key	Default value	Range	Description
Enabled	enabled	1	0-disabled , 1-enabled	Enable/disable MSRP at runtime.

4.2.5 Log files

Several log files are available at runtime to monitor the different stack components.

4.2.5.1 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 00049ffffe039e35
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 66. Port statistics displayed on slave and master entities

Receive counters	
`PortStatRxPkts`	Number of gPTP packets received (ether type 0x88F7)
`PortStatRxSyncCount`	Number of SYNC packets received
`PortStatRxSyncReceiptTimeouts`	Number of SYNC packets receive timeout
`PortStatRxFollowUpCount`	Number of FOLLOW-UP packets received
`PortStatRxAnnounce`	Number of ANNOUNCE packets received
`PortStatAnnounceReceiptTimeouts`	Number of ANNOUNCE packets timeout
`PortStatAnnounceReceiptDropped`	Number of ANNOUNCE packets dropped by the entity
`PortStatRxSignaling`	Number of SIGNALING packets received
`PortStatRxPdelayRequest`	Number of PDELAY REQUEST packets received
`PortStatRxPdelayResponse`	Number of PDELAY RESPONSE packets received
`PortStatPdelayAllowedLostResponsesExceeded`	Number of excess of allowed lost responses to PDELAY requests
`PortStatRxPdelayResponseFollowUp`	Number of PDELAY FOLLOW-UP packets received
`PortStatRxErrEtype`	Number of ether type errors (not 0x88F7)
`PortStatRxErrPortId`	Number or port ID errors

Table 66. Port statistics displayed on slave and master entities*...continued*

Transmit counters	
`PortStatTxPkts`	Number of gPTP packets transmitted
`PortStatTxSyncCount`	Number of SYNC packets transmitted
`PortStatTxFollowUpCount`	Number of FOLLOW-UP packets transmitted
`PortStatTxAnnounce`	Number of ANNOUNCE packets transmitted
`PortStatTxSignaling`	Number of SIGNALING packets transmitted
`PortStatTxPdelayReques`	Number of PDELAY REQUEST packets transmitted
`PortStatTxPdelayResponse`	Number of PDELAY RESPONSE packets transmitted
`PortStatTxPdelayResponseFollowUp`	Number of PDELAY FOLLOW-UP packets transmitted
`PortStatTxErr`	Number of transmit errors
`PortStatTxErrAlloc`	Number of transmit packets allocation errors
Miscellaneous counters	
`PortStatAdjustOnSync`	Number of adjustments performed upon SYNC received
`PortStatMdPdelayReqSmReset`	Number of reset of the PDELAY REQUEST state machine
`PortStatMdSyncRcvSmReset`	Number of reset of the SYNC RECEIVE state machine
`PortStatHwTsRequest`	Number of egress timestamp requests
`PortStatHwTsHandler`	Number of egress timestamp notification
`PortStatNumSynchronizationLoss`	Number or synchronization loss on the slave endpoint (e.g. GM change, GM reference clock discontinuity...)
`PortStatNumNotAsCapable`	Number of transitions from AS_Capable=TRUE to AS_Capable=FALSE

4.2.5.2 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
 neighborGptpCapable: No DelayMechanism: P2P
Port 1 domain(0,0): Role: Disabled Link: Up AS_Capable: No
 neighborGptpCapable: 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: No 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 2 domain(1,20): Role: Disabled Link: Up AS_Capable: Yes
 neighborGptpCapable: Yes DelayMechanism: COMMON_P2P
Port 3 domain(1,20): Role: Disabled Link: Up AS_Capable: No
 neighborGptpCapable: No DelayMechanism: COMMON_P2P
Port 4 domain(1,20): Role Master Link: Up AS_Capable: Yes
 neighborGptpCapable: Yes DelayMechanism: COMMON_P2P
```

4.2.5.3 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_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
INFO  srp    msrp_vector_add_event            : port(0)
 domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_MT
```

```
INFO  srp    msrp_vector_add_event            : port(0)
 domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_MT
INFO  srp    msrp_vector_add_event            : port(1)
 domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_MT
INFO  srp    msrp_vector_add_event            : port(1)
 domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_MT
INFO  srp    msrp_vector_add_event            : port(2)
 domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_MT
INFO  srp    msrp_vector_add_event            : port(2)
 domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_MT
INFO  srp    msrp_vector_add_event            : port(4)
 domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_MT
INFO  srp    msrp_vector_add_event            : port(4)
 domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_MT
```

4.2.5.4 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^2 (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.

4.2.5.4.1 Main TSN task

The main TSN task logs are described below:

- Scheduling counters (“sched” should increment of 500 per second):

```
INFO 1604531064  tsn_task_stats_print            tsn
 task(0x37d8d630)
INFO 1604531064  tsn_task_stats_print            sched
    : 1700000
INFO 1604531064  tsn_task_stats_print            sched early
    : 0
INFO 1604531064  tsn_task_stats_print            sched late
    : 0
INFO 1604531064  tsn_task_stats_print            sched missed
    : 0
INFO 1604531064  tsn_task_stats_print            sched
 timeout   : 0
INFO 1604531064  tsn_task_stats_print            clock
 discont   : 0
INFO 1604531064  tsn_task_stats_print            clock err
    : 0
```

- Scheduling error statistics (difference between actual task scheduling time and programmed time, in ns):

```
stats(0x2ddd4560) sched err min 8062 mean 10662 max 18862
 rms^2 116252492 stddev^2 2558383 absmin 2842 absmax 35082
```

- Scheduling error histogram (XXX ns bucket)

```
n_slot 101 slot_size 10000
18081119 21722197 1676 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

- Processing time statistics (main task duration in ns)

```
stats(0x2ddd49c0) processing time min 21600 mean 27564 max
 47460 rms^2 768483628 stddev^2 8664988 absmin 12540 absmax
 152100
```

- Processing time histogram (XXX ns bucket)

```
n_slot 101 slot_size 1000
0 0 0 0 0 0 0 0 0 0 0 0 9 13 7 259 302 187 14273 64041 91896
 427011 1473790 2154889 4239252 4681777 5682883 5516429
 4915621 3789351 2688466 1709802 1036649 587716 327653 170711
 89620 47637
 25422 15211 9573 6908 4852 3694 3101 2910 3028 3036 3331 3091
 2842 2375 1748 1251 895 585 342 208 100 63 64 26 21 13 9 14 6
 7 4 4 6 3 4 1 0 2 2 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0
```

- Total time statistics (scheduling error + processing time, in ins)

```
stats(0x2ddd4e20) total time min 30082 mean 38227 max 55862
 rms^2 1473506486 stddev^2 12160755 absmin 18962 absmax 170082
```

- Total time histogram (XXX ns bucket)

```
n_slot 101 slot_size 1000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 2 15 16 57 273
 1674 7795 30165 80952 240707 651779 1497160 2290907 3606869
 4077757 4761502 4647779 4239893 3631222 2831749 2209868
 1631293 1173544
 819740 536234 338535 201816 115114 65529 36480 21413 12892
 8249 5613 4178 3310 3034 2890 2921 2615 2461 2132 1790 1427
 1082 844 585 353 266 170 98 73 48 20 25 6 14 14 1 6 2 3 8 2 2
 1 3 2 1 1 2 1 2 0 1 1 0 0 0 10 0
```

4.2.5.4.2 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:

```
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
```

4.2.5.4.3 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.

```
INFO 1604531069 socket_stats_print cyclic rx socket(0x419560)
 net_sock(0x37d8d660) peer id: 1
INFO 1604531069 socket_stats_print valid frames : 1702497
INFO 1604531069 socket_stats_print err id : 0
INFO 1604531069 socket_stats_print err ts : 305
INFO 1604531069 socket_stats_print err underflow : 2
INFO 1604531069 socket_stats_print link up
```

- Traffic latency statistics (the difference between the theoretical scheduling time of the peer that sent the frame and the frame receive time (measured by the MAC), in ns)

```
stats(0x419a28) traffic latency min 503417 mean 503503 max
 503637 rms^2 253515981945 stddev^2 2004 absmin 503397 absmax
 504337
```

- Traffic latency histogram (XXX ns bucket)

```
n_slot 101 slot_size 1000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 299998 0
```

4.3 IEEE 1588/802.1AS

IEEE 1588 is the IEEE standard for a precision clock synchronization protocol for networked measurement and control systems.

IEEE 802.1AS is the IEEE standard for local and metropolitan area networks – timing and synchronization for time-sensitive applications in bridged local area networks. It specifies the use of IEEE 1588 specifications where applicable in the context of IEEE Std 802.1D-2004 and IEEE Std 802.1Q-2005.

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.

4.3.1 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.*

4.3.2 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.*

4.3.3 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.

4.3.4 Software stacks

4.3.4.1 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.
  - The telecom profiles G.8265.1, G.8275.1, and G.8275.2.
  - 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.

4.3.4.2 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.

4.3.5 Quick Start for IEEE 1588

4.3.5.1 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,

```
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

```
Delay Mechanism
-E        E2E, delay request-response (default)
-P        P2P, peer delay mechanism
Network Transport
-2        IEEE 802.3
-4        UDP IPV4 (default)
-6        UDP IPV6
```

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
```

4.3.5.2 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.

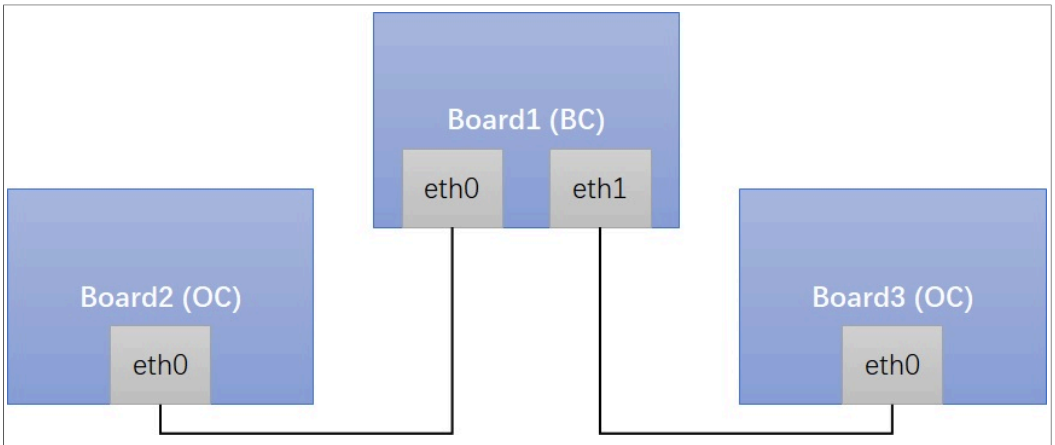


Figure 64. Hardware setup for boundary clock verification

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

```
Delay Mechanism
-E        E2E, delay request-response (default)
-P        P2P, peer delay mechanism
Network Transport
-2        IEEE 802.3
-4        UDP IPV4 (default)
-6        UDP IPV6
```

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
```

4.3.5.3 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.

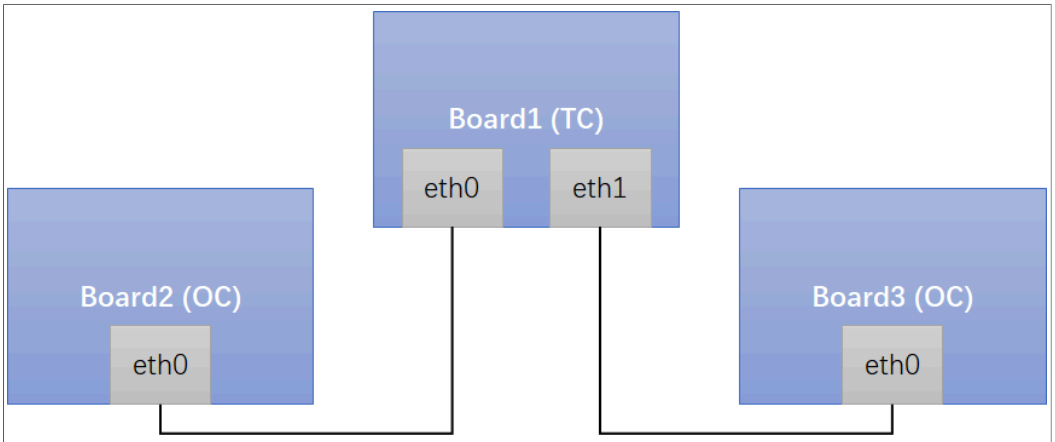


Figure 65. Sample network connection of three boards

Run `linuxptp` on Board1 (transparent clock). If you want Board1 works as E2E TC, use `E2E-TC.cfg`. If your 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).

```
$ ptp4l -i eth0 -m -2
```

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.

4.3.6 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.

4.3.6.1 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.

```
neighborPropDelayThresh 800
```

Run linuxptp on each board. For example, eth0 is used on each board.

```
$ 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.

4.3.6.2 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.

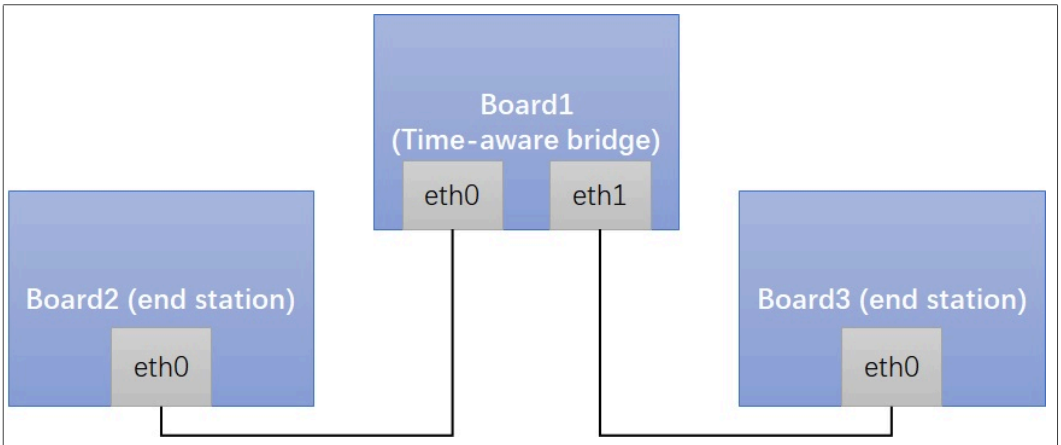


Figure 66. Setup for time-aware bridge verification

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).

4.3.7 Long term test

This section describes the long term test results for Linux PTP stack implementation.

4.3.7.1 linuxptp basic synchronization

Linux PTP

Connection: back-to-back master to slave

Configuration: Sync internal is -3

Test boards: two LS1021A-TSN boards, one as master and another one as slave.

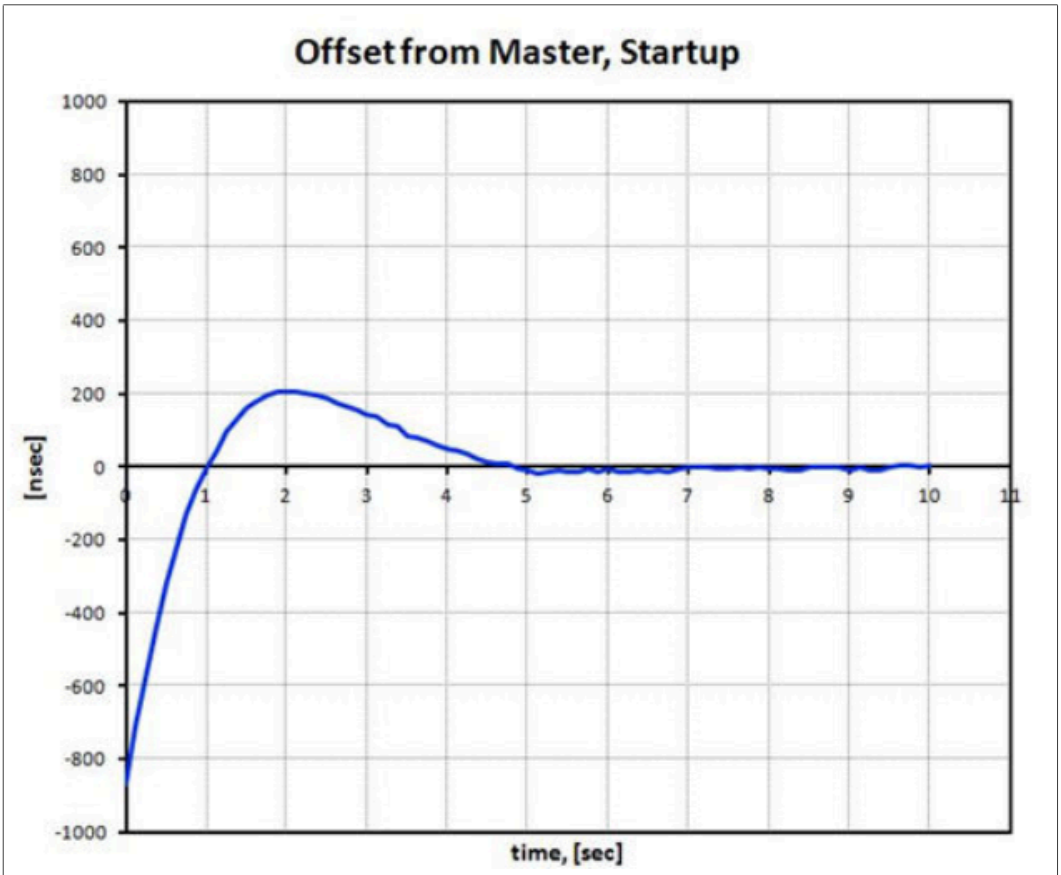


Figure 67. Offset from master in start up state

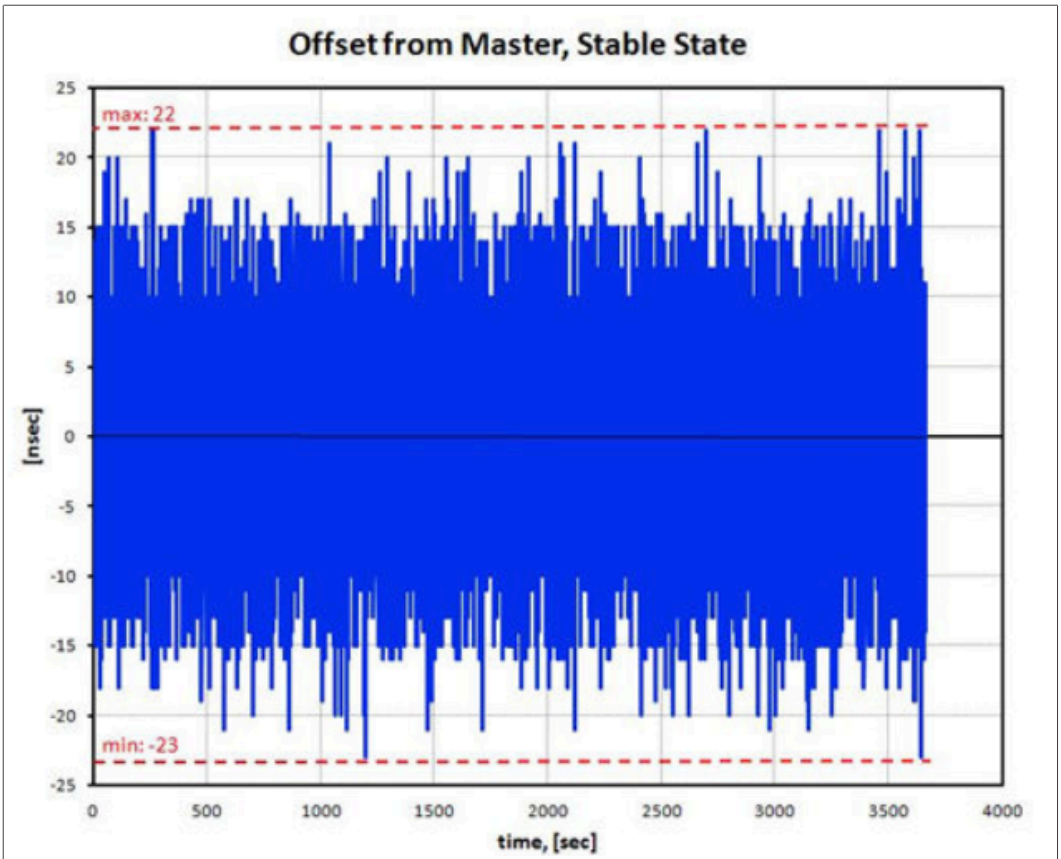


Figure 68. Offset from master in stable state

4.3.8 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.

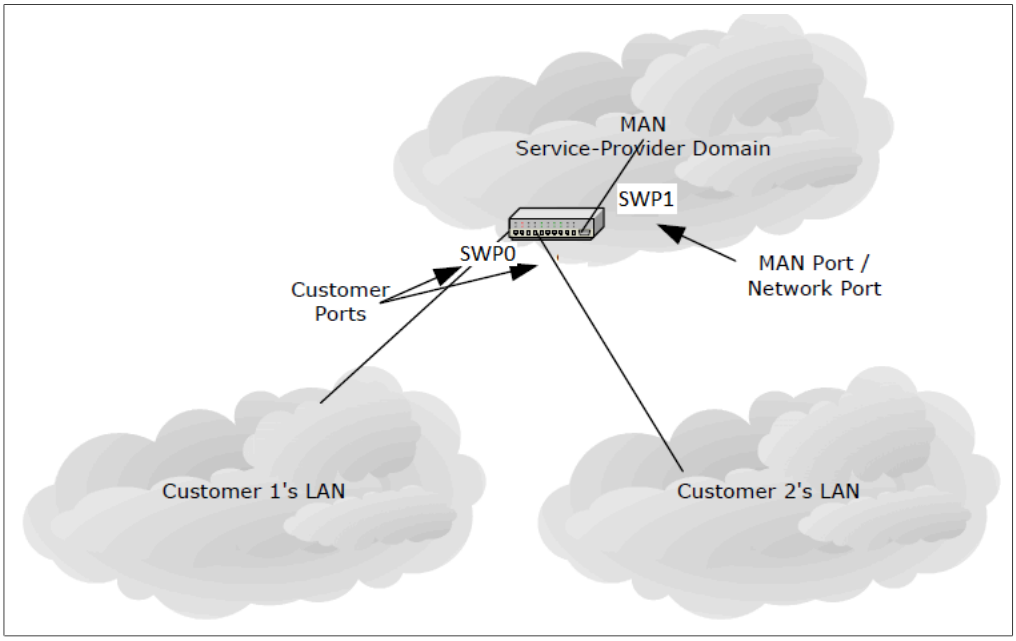
For example, use option `--tx_timestamp_timeout=20` when running ptp4l as shown below:

```
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
```

4.4 Networking

4.4.1 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 different user's VLAN traffic on a link or bundle using different user VLANs. When using Q-in-Q, the user's 802.1Q VLAN tags `(C-TAG:0x8100)` are prepended by the service VLAN tag `(S-TAG: 0x88A8)`.
2. **Q-in-Q application scenario**
   In the following scenario, switch's port swp0 connects with Customer 1's LAN, swp1 connects with ISP's MAN,
   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

```
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:

```
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:

```
bridge vlan del dev swp0 vid 1 pvid
bridge vlan add dev swp0 vid 100 pvid untagged
bridge vlan add dev swp1 vid 100
```

d. Result

```
Customer(tpid:8100 vid:111) -> swp0 -> swp1 -> ISP(STAG
 tpid:88A8 vid:100, CTAG tpid:8100 vid:111)
ISP(tpid:88A8 vid:100 tpid:8100 vid:222) -> swp1 -> swp0 -
> Customer(tpid:8100 vid:222)
```

4.4.2 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 -s 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 67. Chain allocation

chain ID	Actions	Hardware module	keys
10000	skbedit priority	IS1 lookup 0	Source MAC address, source IP address (32 bits) outer VLAN, IP protocol, source TCP/UDP ports.
11000	vlan pop; vlan modify	IS1 lookup 1	Inner and outer VLAN, source and destination IP addresses (32 bits), IP protocol, source and destination TCP/UDP ports.
12000	goto chain [PAG]	IS1 lookup 2	Source MAC address, source IP address (32 bits) outer VLAN, IP protocol, source TCP/UDP ports.

Table 67. Chain allocation*...continued*

chain ID	Actions	Hardware module	keys
20000-20255	police; trap	IS2 lookup 0	Source and destination MAC address, source and destination IP addresses (32 bits), IP protocol, source and destination TCP/UDP ports.
21000-21255	drop; redirect	IS2 lookup 1	Source and destination MAC address, source and destination IP addresses (32 bits), IP protocol, source and destination TCP/UDP ports.
30000	gate; police	PSFP	destination MAC address and Vlan ID

Before using chains, users should register each chain and set chain pipeline order for a packet. The hardware ingress order is: **IS1->IS2->PSFP**.

```
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
```

After registering the chain, add rules to the corresponding chain. Following are the use cases for testing:

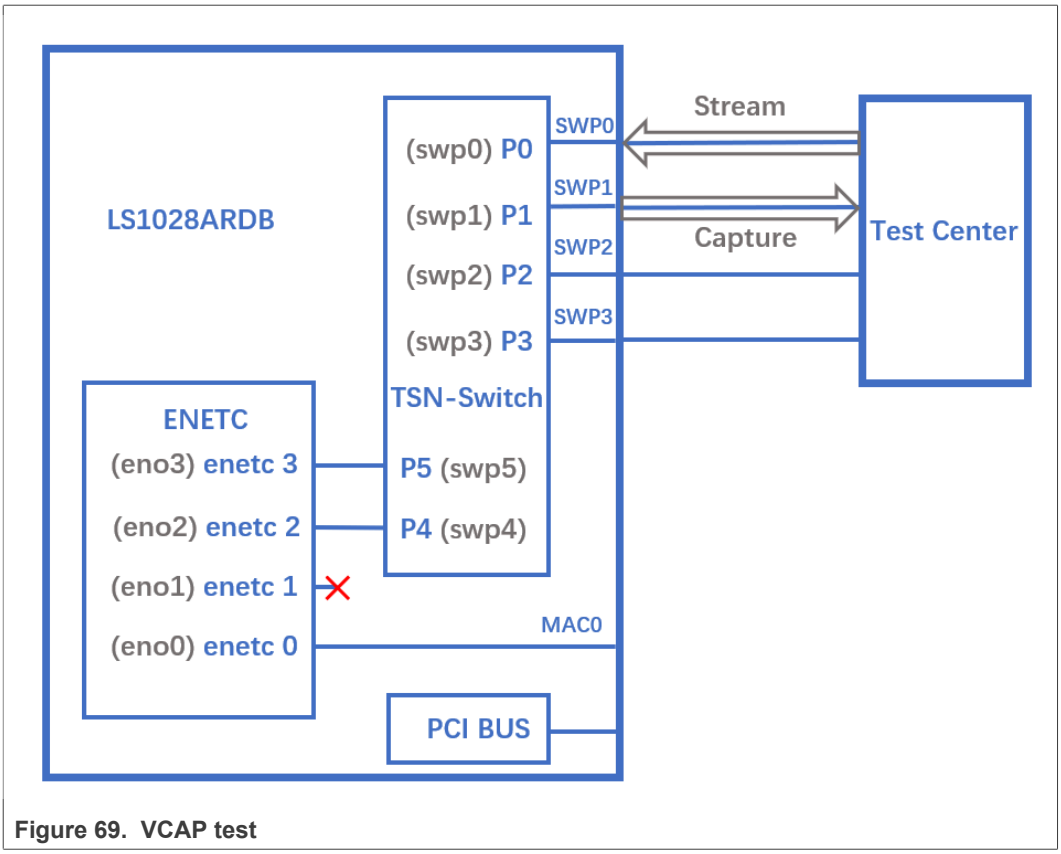


Figure 69. VCAP test

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)`

5 Protocols

5.1 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.

5.1.1 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. The goal during development of EtherCAT was to apply Ethernet for automation applications requiring short data update times (also called cycle times; ≤ 100 µs) with low communication jitter (for precise synchronization purposes; ≤ 1 µs) and reduced hardware costs.

- 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.

At present, there are two open source EtherCAT masters, IGH for Cortex-A core and SOEM for Cortex-M core, supported on Real-time Edge. Both IgH and SOEM are solutions that help users get rid of the low-level development directly on EtherCAT protocol, and provide a common APIs 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 a 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, window, or FreeRTOS. It can even run without an operating system (aka '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.

5.1.2 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 following figure shows the EtherCAT frame structure.

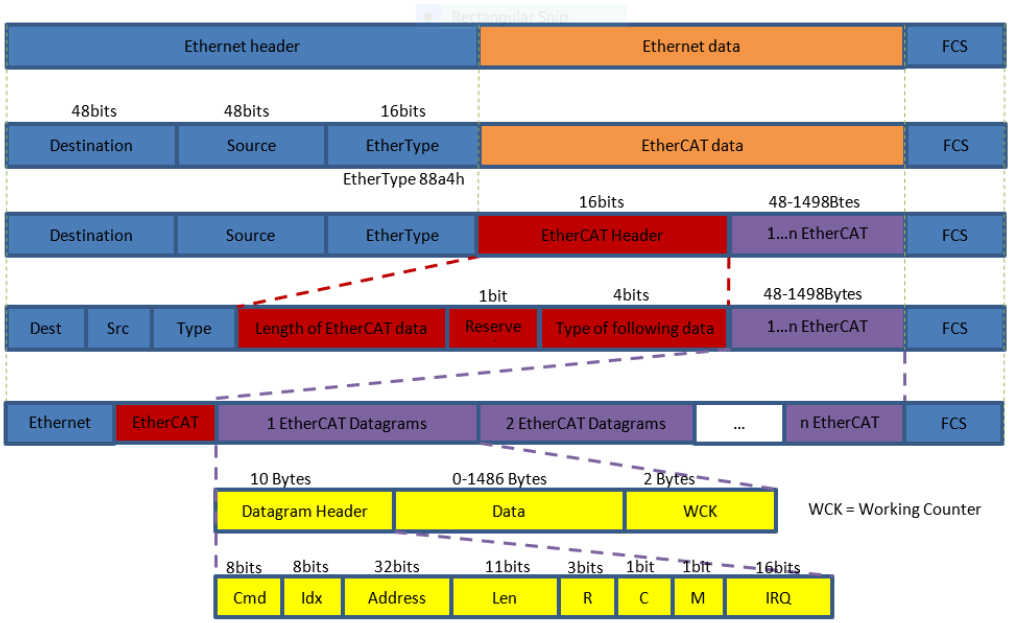


Figure 70. EtherCAT frame structure

5.1.3 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 following figure shows that IGH EtherCAT master architecture.

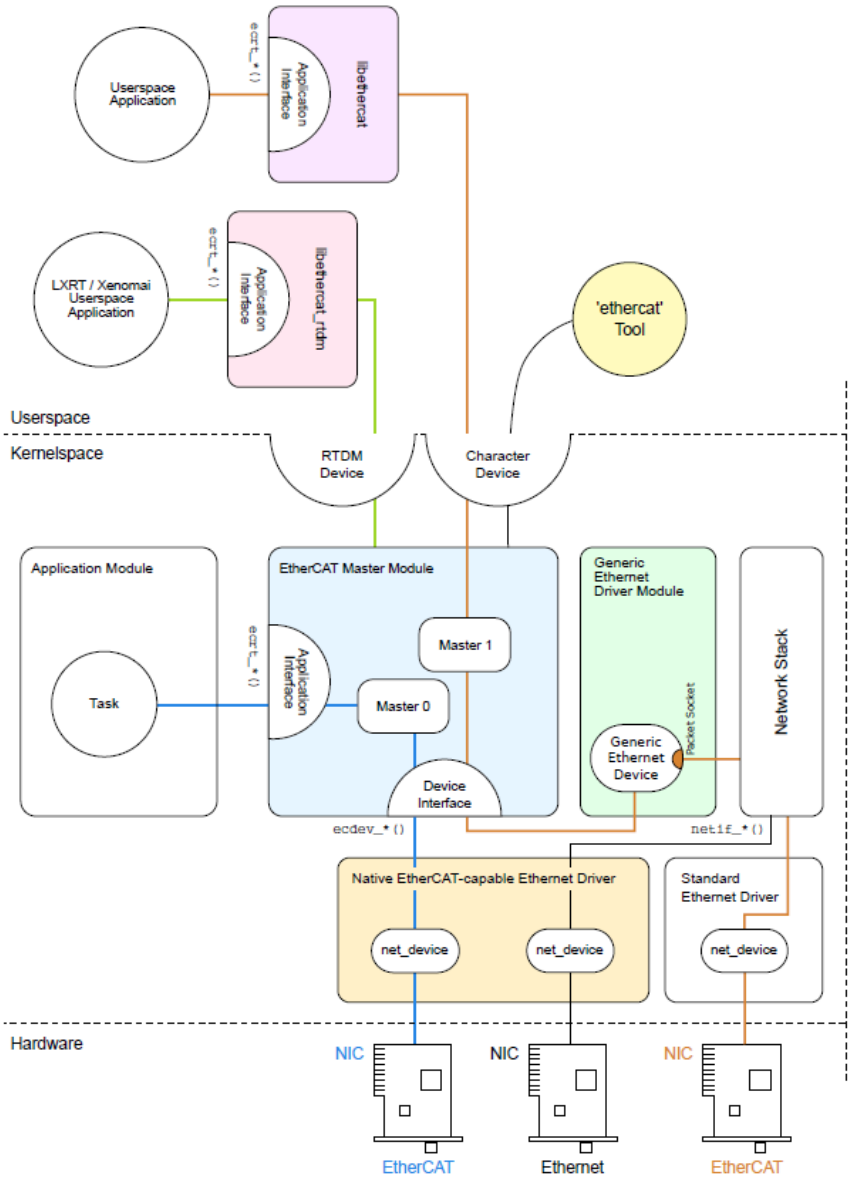


Figure 71. IGH EtherCAT master architecture

5.1.3.1 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 to 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" could be used for the FEC MAC on i.MX 8M Mini LPDDR4 EVK and i.MX 8M Plus LPDDR4 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 to all the platforms Real-time Edge supportes. But the disadvantage is that the performance is a little worse than the native driver, because the Ethernet frame date have to traverse the Linux network stack.

5.1.3.2 IGH EtherCAT Setup

Before the IGH EtherCAT daemon start, the Ethernet device and the Ethernet driver must be specified by setting the "`MASTER0_DEVICE`" and "`DEVICE_MODULES`" variables on "`/etc/ethercat.conf`" file.

5.1.3.2.1 Specify 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.

5.1.3.2.2 Generic Ethernet Driver

The generic Ethernet driver is enabled on Real-time Edge images by default for all platforms. And it can be specified by setting "DEVICE_MODULES" variable to "generic" as below on "/etc/ethercat.conf" file.

```
DEVICE_MODULES="generic"
```

5.1.3.2.3 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.

5.1.3.2.4 Native Ethernet Driver for i.MX 8M Plus LPDDR4 EVK

The native Ethernet driver '`ec_fec`' is not enabled on Real-time Edge images by default for i.MX 8M Plus LPDDR4 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**:
   Like 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.

   ```
   DELTA_KERNEL_DEFCONFIG:append:mx8mp = " linux-fec.config"
   ```

2. **Enable the native Ethernet driver '`ec_fec`' for i.MX 8M Plus LPDDR4 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.

   ```
   IGH_ETHERCAT:imx8mp-lpddr4-evk = " fec "
   ```

Note: *Please note 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`.*

5.1.3.2.5 Native Ethernet Driver for LS1028ARDB

The native Ethernet driver "ec_enetc" is enabled on Real-time Edge images by default for LS1028ARDB. And it can be specified by setting "DEVICE_MODULES" variable to "enetc" as below on "/etc/ethercat.conf" file.

```
DEVICE_MODULES="enetc"
```

5.1.3.2.6 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.

Different from other native drivers mentioned above, DPAA is a network co-processer that is more complex than ordinary network cards. So the native driver "ec_dpaa1" must be configured by "ethercat_port" variable in U-Boot to notify the DPAA co-processer 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 below:

Table 68. LS1046ARDB chassis

Port name on chassis	Port name in Linux
RGMII1	fm1-mac3
RGMII2	fm1-mac4
SGMII1	fm1-mac5
SGMII2	fm1-mac6
10G Copper	fm1-mac9
10G SEP+	fm1-mac10

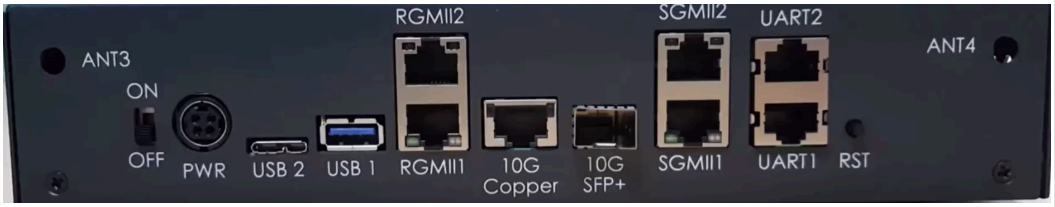


Figure 72. LS1046ARDB Ethernet chassis

The relationship between ethernet port name on LS1043ARDB chassis and the name in the Linux is in the table below:

Table 69. LS1043ARDB chassis

Port name on chassis	Port name in Linux
QSGMII.P0	fm1-mac1
QSGMII.P1	fm1-mac2
QSGMII.P2	fm1-mac5
QSGMII.P3	fm1-mac6
RGMII1	fm1-mac3
RGMII2	fm1-mac4
10G Copper	fm1-mac9

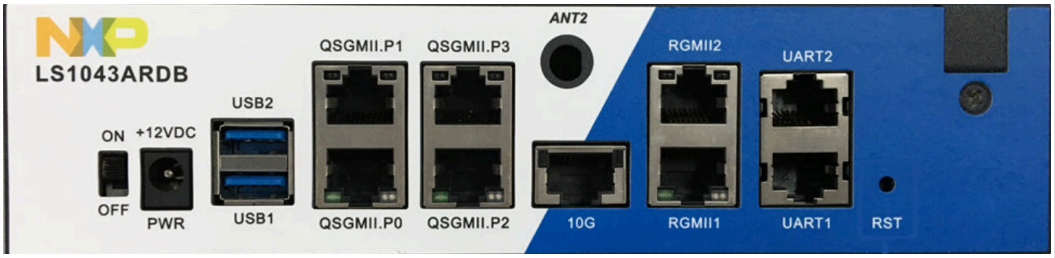


Figure 73. LS1043ARDB ethernet chassis

There are 3 instances of ec_dpaa1 configuration for LS1046ARDB.

1. There is only one network topologies, 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.

5.1.3.2.7 IGH EtherCAT Start

Use the below command to start 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 is 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 statictics.
      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.
```

```
      sdos       List SDO dictionaries.
      sii_read   Output a slave's SII contents.
      sii_write  Write SII contents to a slave.
      slaves     Display slaves on the bus.
      soe_read   Read an SoE IDN from a slave.
      soe_write  Write an SoE IDN to a slave.
      states     Request application-layer states.
      upload     Read an SDO entry from a slave.
      version    Show version information.
      xml        Generate slave information XML.
```

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
```

5.1.3.3 real-time-edge-servo stack

real-time-edge-servo is a CiA402 (also referred to as DS402) profile framework based on IgH CoE interface (An EtherCAT Master stack, see Section 5.1 section for details). It abstracts the CiA 402 profile and provides an easily-usable 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 and SV-X3EB, Just Motion Control 2HSS458-EC and INOVANCE InoSV680N.

5.1.3.3.1 CoE network

The figure below illustrates a typical CoE network.

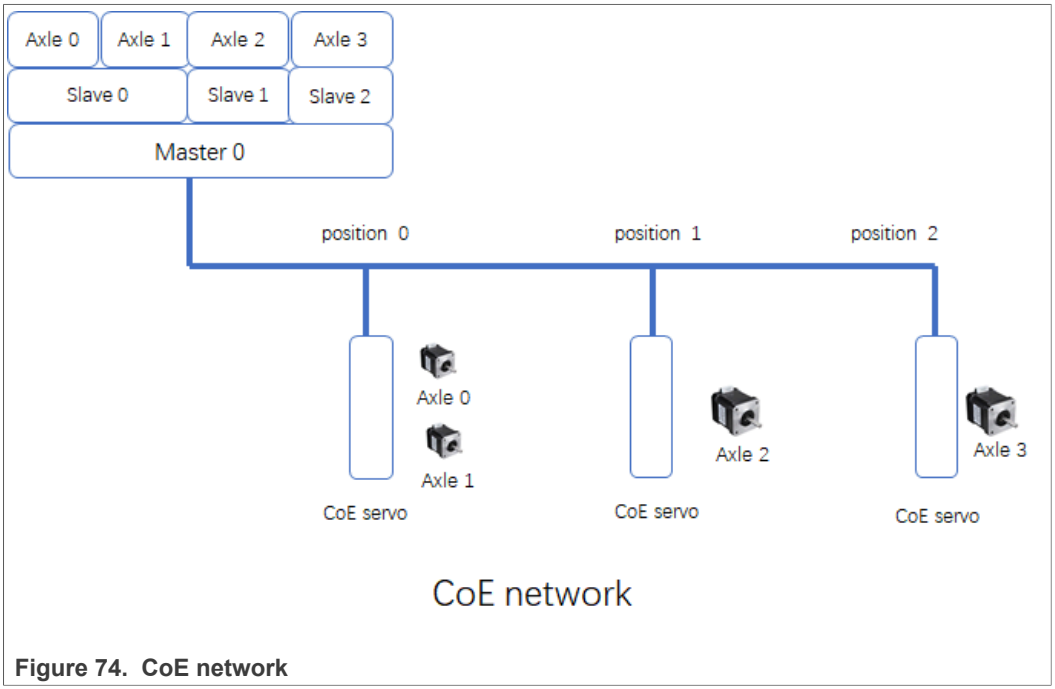


Figure 74. CoE network

There are three CoE servos on this network and we name them slave ***x*** as the position they are. Each CoE servo could have more then 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.

5.1.3.3.2 Libnservo architecture

real-time-edge-servo is running on top of *lgh* EtherCAT stack. And the *lgh* stack provides CoE communication mechanisms - Mailbox and Process Data. Using these mechanisms, real-time-edge-servo could access the CiA Object Dictionary located on CoE servo.

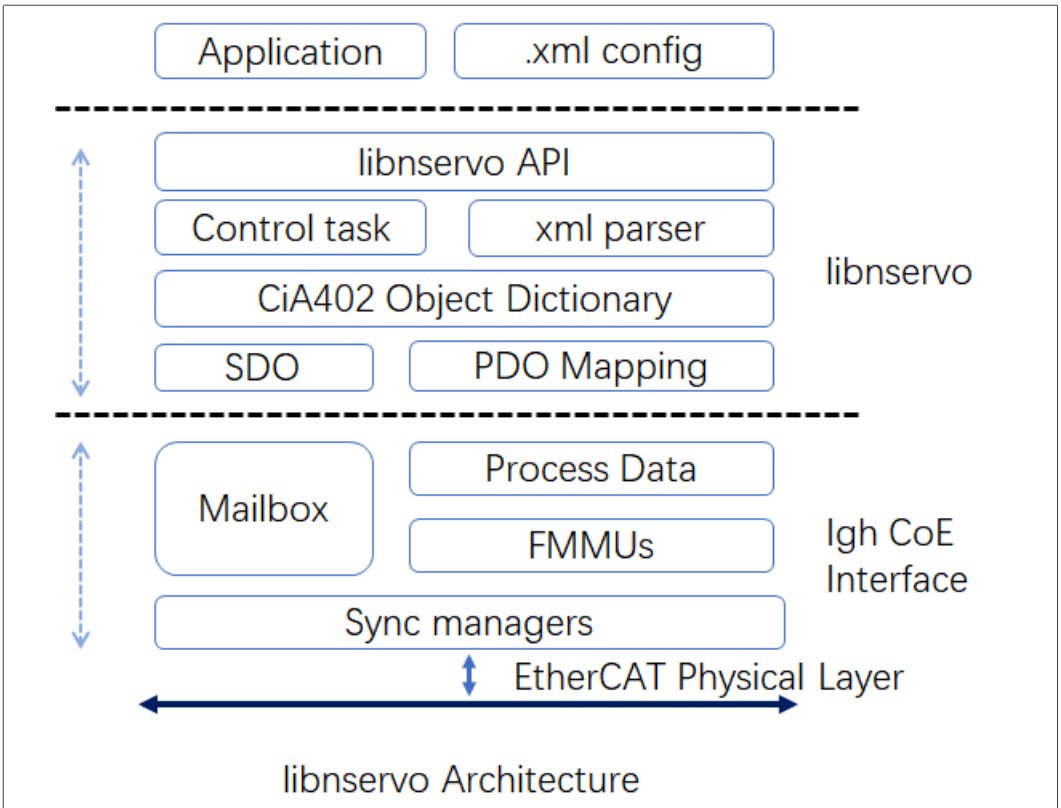


Figure 75. Libnservo architecture

Control task initiates the master, all slaves on the CoE network and registers all PDOs to Igh stack, then constructs a data structure to describe each axle. Finally, the control task creates a task to run the user task periodically.

5.1.3.3.3 real-time-edge-servo Xml configuration

This section focuses on how the xml config file describes a CoE network.

The basic framework of the XML configuration is shown as in code below:

```
<?xml version="1.0" encoding="utf-8"?>
<Config  Version="1.2">
  <PeriodTime>#10000000</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>
```

```
  <Axles>
      <Axle>
        ...
      <\Axle>
      <Axle>
        ...
      <\Axle>
    <\Axles>
</Config>
```

- 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 #x 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.

5.1.3.3.3.1 Master element

As *CoE network* section shown, the Master could has many slaves, so the Master element will consist of some *Slave* elements.

```
<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 means 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 is used to indicate which slave will be used the reference clock.
- <Slave> element means 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>
  <Watchdog>ENABLE</Watchdog>
```

```
  <PdoNum>#1</PdoNum>
  <Pdo SubIndex="#1">
    <Index>#x1600</Index>
    <Name>RxPdo 1</Name>
    <Entry SubIndex="#1">
      ...
    </Entry>
    <Entry SubIndex="#2">
      ...
    </Entry>
  </Pdo>
</SyncManager>
```

- <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="#1> element contains the object dictionary entry we want to mapped.
  - <Index> PDO address.
  - <Name> PDO name
  - <Entry> the object dictionary we want to mapped.

The Entry element is used to describe a object dictionary we want to mapped.

```
<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.

```
<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.

5.1.3.3.3.2 Axle element

```
<Axle master_index='#0' slave_position="#0" AxleIndex="#0"
 AxleOffset="#0">
    <Mode>pp</Mode>
    <Name>x-axle</Name>
    <reg_pdo>
  ...
```

```
    </reg_pdo>
    <reg_pdo>
  ...
    </reg_pdo>
</Axle>
```

- *master_index* attribute indicates which *master* this *axle* belong to.
- *slave_position* attribute indicates which *slave* this *axle* belong to.
- *AxleOffset* attribute indicates which *axle* this *axle* is on the slave. As mentioned above, a CoE slave could have more then on *axle* . If this axle is the second axle on the slave, set *AxleOffset="#1"* .
- <Mode> means which 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

```
<reg_pdo>
  <Index>#x606c</Index>
  <Subindex>#x0</Subindex>
  <Name></Name>
</reg_pdo>
```

5.1.3.3.4 Testing a CoE servo system

5.1.3.3.4.1 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.

Figure 76. '2HSS458-EC' servo system

Figure 77. ASDA-B3-E Servo System

5.1.3.3.4.2 Software preparation

Make sure the below config options are selected when configuring Real-time Edge.

- igh-ethercat
- libxml2
- real-time-edge-servo

5.1.3.3.4.3 CoE network detection

- Igh configuration
  - Configure the `MASTER0_DEVICE` field of the */etc/ethercat.conf*
    Set `MASTER0_DEVICE` 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
```

5.1.3.3.4.4 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
get_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
```

5.1.3.3.4.5 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 &
```

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 Position Mode
```

b. Get 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 will turn 10 rounds. (target_position: 167772160 - current_position:0) / 16777216 = 10

f. Get current position of axle 0.

```
[root]# nservo_client -a 0 -c get_current_position
get_current_position of the axis 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
```

5. 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 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 will turn 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
 get_mode of the axle 0 : Cyclic sync Position Mode
```

   b. Load trajectory planning information from 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, it means motor will come back the first point of TpArrays and running again.*
   - *Scale*: this filed is used to set the resolution of per unit. For eample, the unit of position point in TpArrays is degree. So the Scale should to be set 167772160/360 = *46603.*
     167772160 is the resolution per motor revolution.
   - *Bias: this filed is used to set bias value for the position 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: 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. 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; TpArrays=[(0:2000),(45:1000),(45:2000),
(90:1000),(90:2000),(270:1000),(270:2000),(0:1000)];
Axis=1; Cyclic=1; Scale=364; Bias=0; Accel=8; Decel=8;
 Max_speed=3600; TpArrays=[(0:2000),(45:1000),(45:2000),
(90:1000),(90:2000),(270:1000),(270:2000),(0:1000)];
```

- *Axis*: the index of the axle.
- *Cyclic*: if this field is set, it means motor will come back the first point of TpArrays and run again.
- *Scale*: this field is used to set the resolution of per unit. For eample, the unit of position point in TpArrays is degree. So the Scale should to be set 167772160/360 = *46603.*
  167772160 is the resolution per motor revolution.
- *Bias: this field is used to set bias value for the position 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, it means motor will come back the first point of TpArrays and running again.*
- *Scale*: this filed is used to set the resolution of per unit. For eample, the unit of position point in TpArrays is degree. So the Scale should to be set 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 rotating to the this point from the last point. The unit of the time field is ms. set axle 0 to start running.

```
[root]# nservo_client -a 0 -c set_start
set_start of the axis 0
```

- Get *current_position* of axle 0.

```
[root]# nservo_client -a 0 -c get_current_position
get_current_position of the axis 0 : 2815922
```

- Set axle 0 to stop running.

```
[root]# nservo_client -a 0 -c set_stop
set_stop of the axis 0
```

- Set all axles in CSP mode to start running.

```
[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.

```
[root]# nservo_client -c set_stop_all
```

All CSP axis in running status start to stop

- Exit

```
[root]# nservo_client -c exit
```

5.1.3.3.5 EtherCAT multiple axes control system

5.1.3.3.5.1 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 below figure, this system could 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.

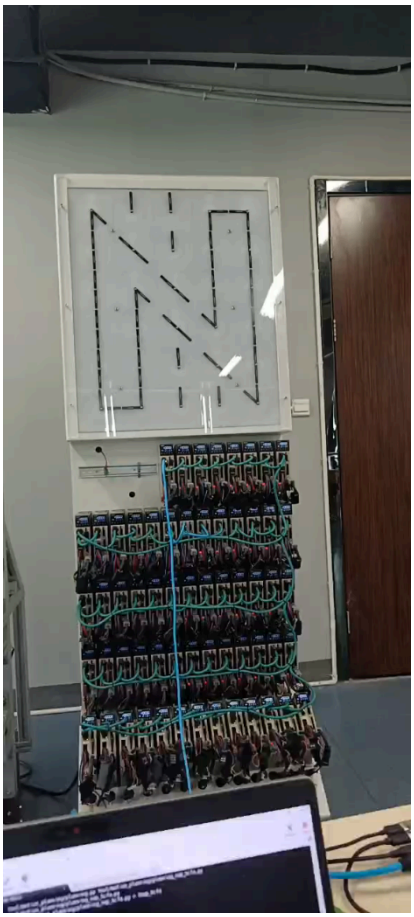


Figure 78. 60-axes servo system built by HCFA

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 rev 2.4 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"e 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 motor
  After the trajectory planning information file is loaded, use the below command to start the system.

```
[root]# nservo_client -c set_start_all
```

5.1.4 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 IOmap
- Detecting errors reported by SOEM
- Managing errors reported by SOEM

Refer http://openethercatsociety.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 SOME runs on FreeRTOS or without an operating system (bare metal).

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).

5.1.4.1 SOEM for i.MX 8M Plus LPDDR4 EVK platform

5.1.4.1.1 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

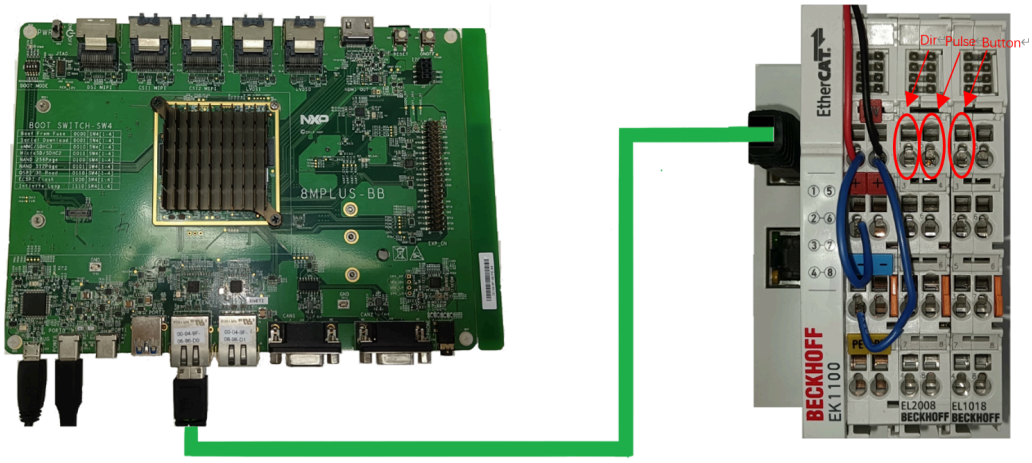


Figure 79. SOEM hardware setup for i.MX 8M Plus LPDDR4 EVK platform

The channel-1 of the EL1018 is connected to a button to generate a 24 V pulse as a 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.

5.1.4.1.2 Build Demo Images

The demo images "freertos_soem_gpio_pules" and "soem_gpio_pules" have been compiled by default with the i.MX 8M Plus LPDDR4 EVK target image compiling, and they will be installed into the directory "/example" of the target rootfs. For JTAG Download, these images also could be found on directory "<image-build-dir>/tmp/deploy/images/imx8mpevk/examples/" on building host.

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mpevk/examples/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/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 Plus LPDDR4 EVK build directory:

```
bitbake soem-gpio-pulse
```

or

```
bitbake freertos-soem-gpio-pulse
```

5.1.4.1.3 Running SOEM demo images using J-Link GDB Server

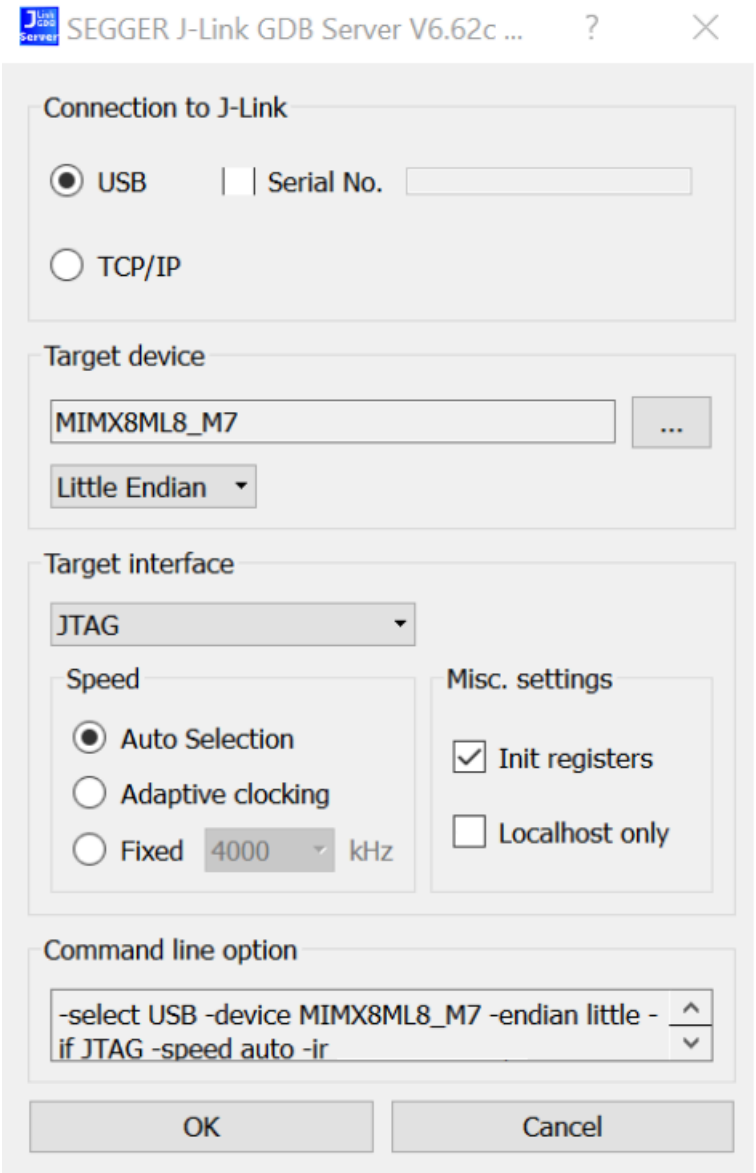
On i.MX 8M Plus LPDDR4 EVK platform, images only support JTAG Download mode currently. This section describes the steps to run a demo application using J-Link GDB Server application. After the J-Link interface configured and connected, please follow these steps to download and run the demo applications:

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:

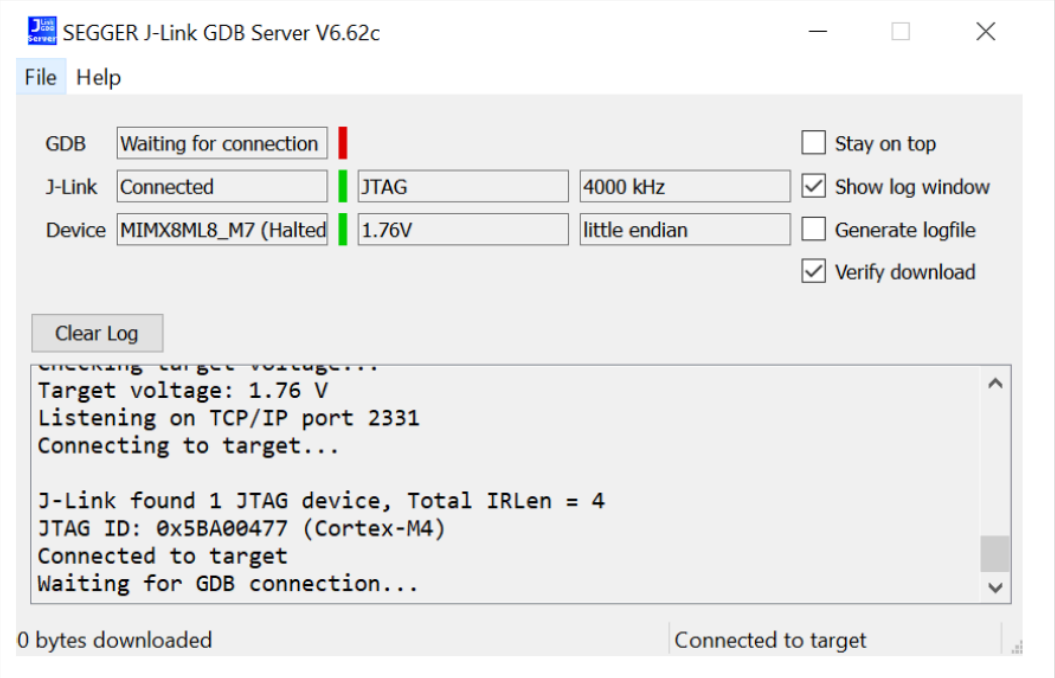


Figure 80. GDB server settings and log

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 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
```

5.1.4.2 SOEM for i.MX 8M Mini LPDDR4 EVK platform

5.1.4.2.1 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 diaital output, 24V DC
- EL1018 - EtherCAT Terminal, 8-channel digital input, 24 V DC
- 24V DC Power Supply

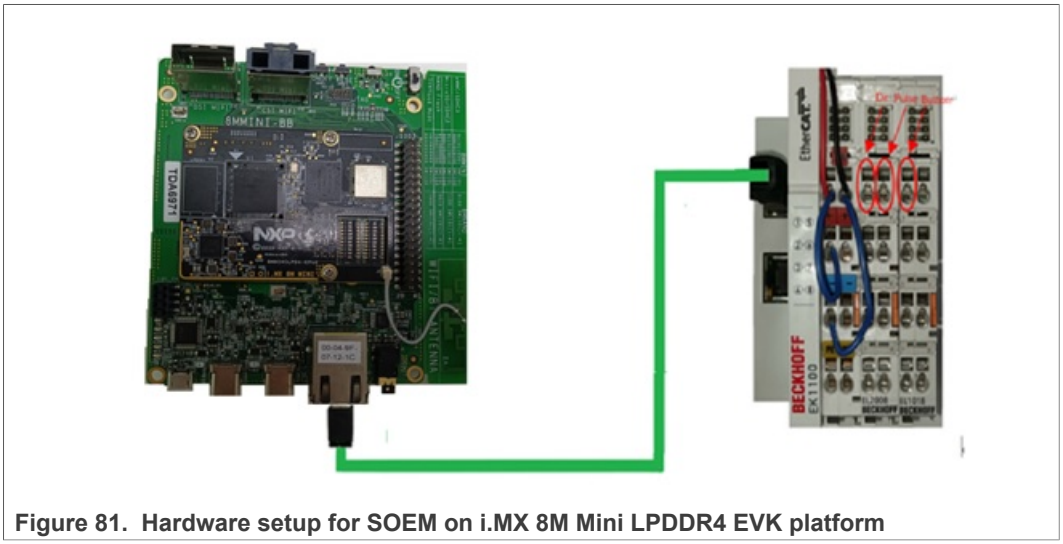


Figure 81. Hardware setup for SOEM on i.MX 8M Mini LPDDR4 EVK platform

The channel-1 of the EL1018 is connected to a button to generate a 24 V pulse as a 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.

5.1.4.2.2 Build Demo Images

The demo images "freertos_soem_gpio_pules" and "soem_gpio_pules" have been compiled by default with the i.MX 8M Mini LPDDR4 EVK target image compiling, and they will be installed into the directory "/example" of the target rootfs. For JTAG

Download, these images also could be found on directory "<image-build-dir>/tmp/deploy/images/imx8mmevk/examples/" on building host.

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mmevk/examples/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/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
```

5.1.4.2.3 Run SOEM demo images

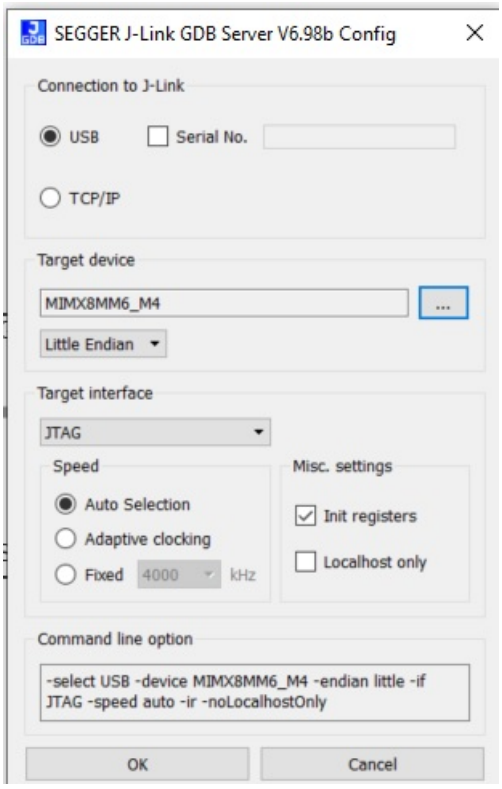
This section describes the steps to run a demo application using J-Link GDB Server application. After the J-Link interface configured and connected, please follow these steps to download and run the demo applications:

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 MIMX8ML_M7 device:

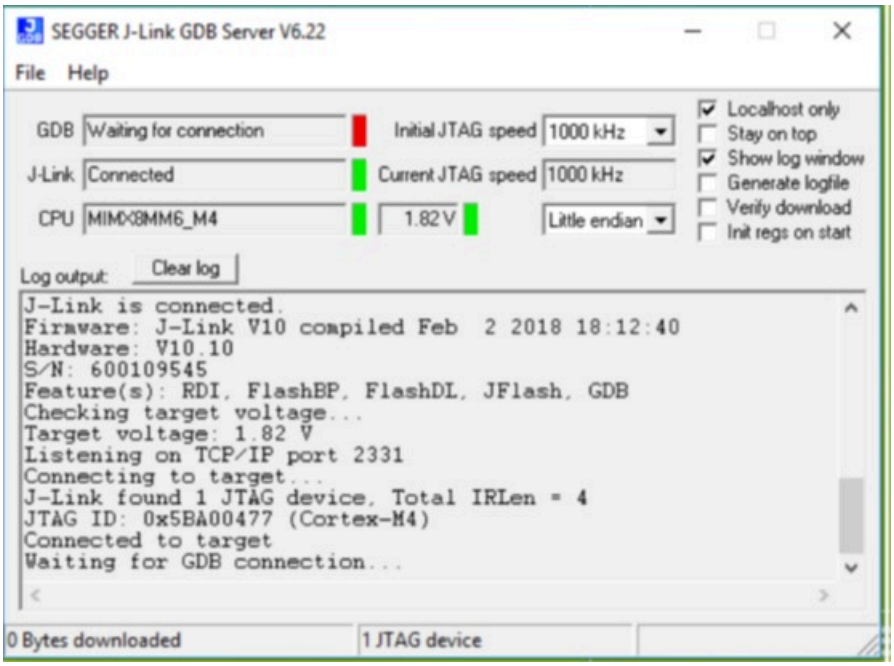
   ```
   $ JLinkGDBServer -if JTAG -device
   SEGGER J-Link GDB Server Command Line Version
   JLinkARM.dll
   Command line: -if JTAG -device MIMX8ML8_M7
   -----GDB Server start settings-----
   GDBInit file: none
   GDB Server Listening port: 2331
   ```

```
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 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 MIMX8ML8_M7 .



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
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 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
```

5.1.4.2.4 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 program 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 dipswitch configuration.
4. Open your preferred serial terminals for the serial devices, setting 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 terminals, then enter to 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/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/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.

5.2 FlexCAN and CAN Open

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.

5.2.1 Introduction

Both the LS1021A and LS1028A boards 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.

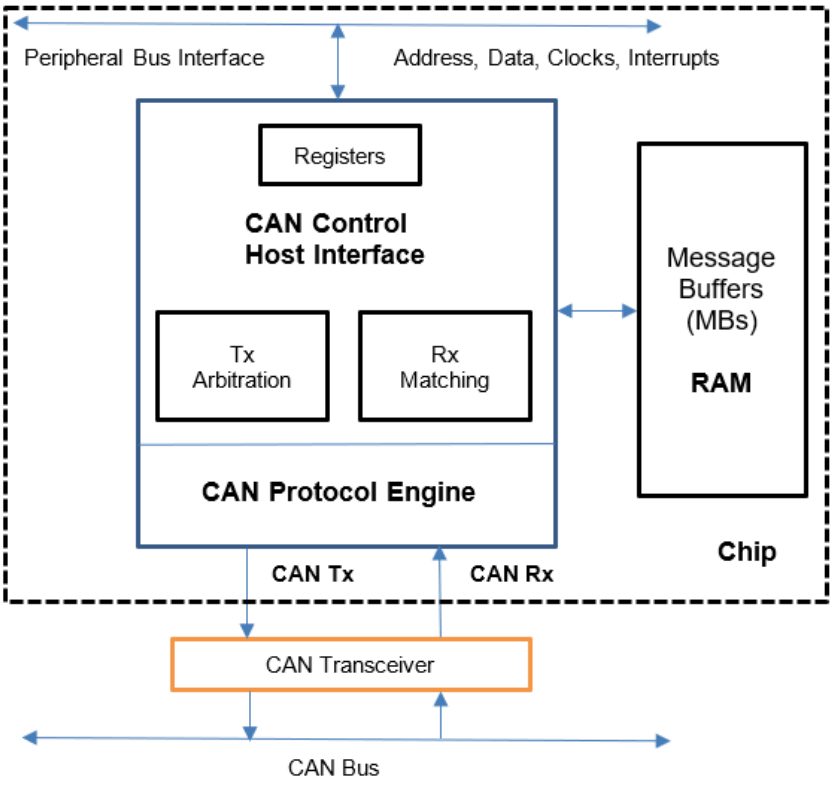


Figure 82. FlexCAN block diagram

5.2.1.1 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.

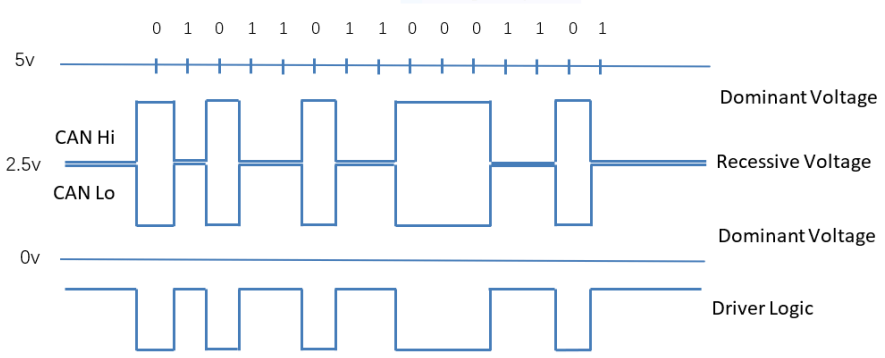


Figure 83. High speed CAN signaling

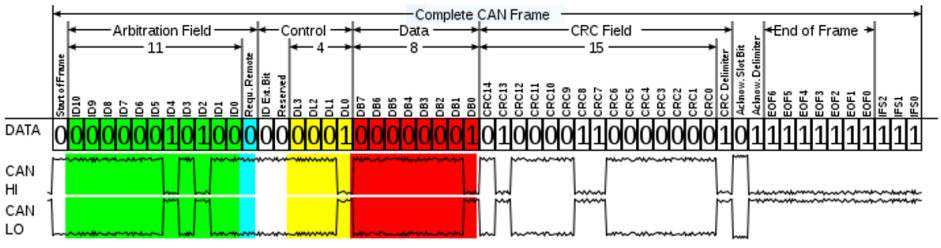


Figure 84. Base frame format

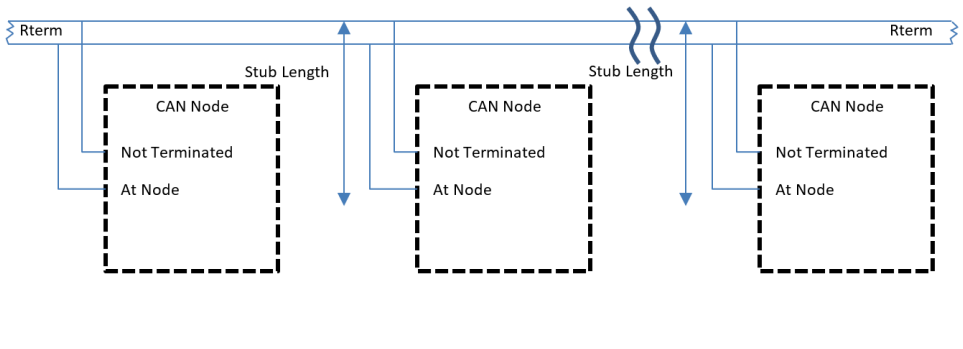


Figure 85. High speed CAN network

5.2.1.2 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.

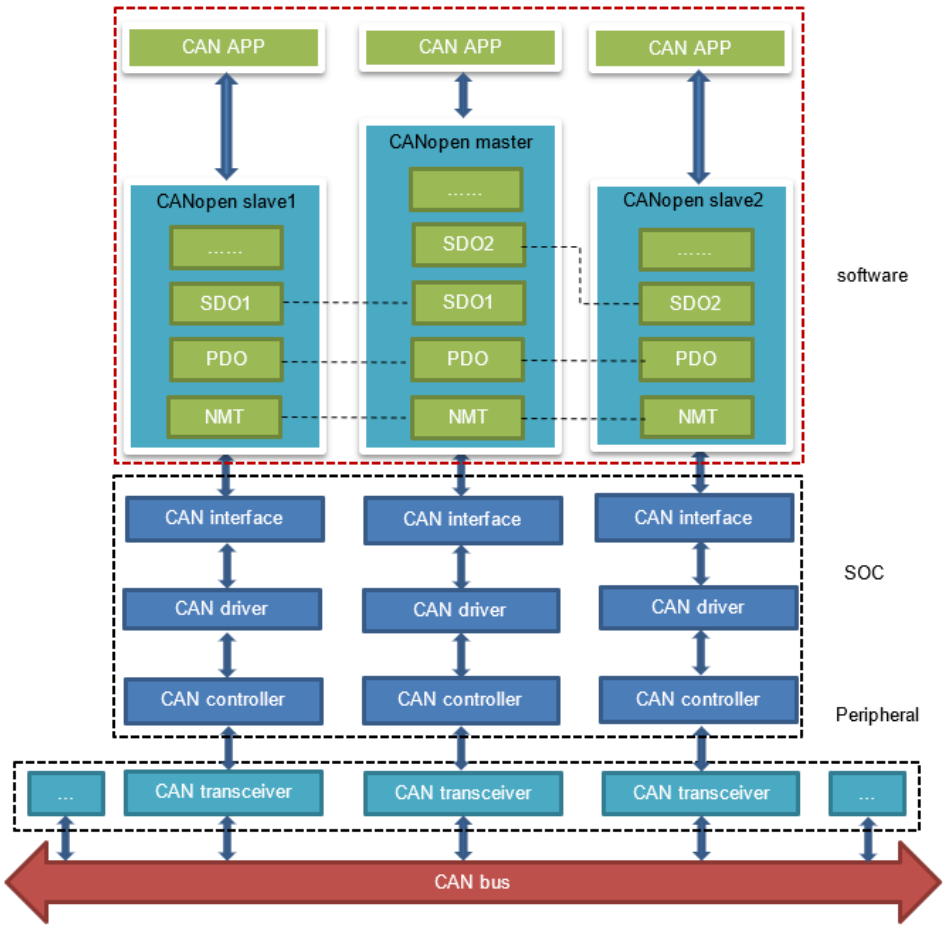


Figure 86. CANopen architecture

5.2.2 FlexCAN integration in Real-time Edge

For LS1021A, there are four CAN controllers. Two CAN controllers (CAN3 and CAN4) are used to communicate with each other. CAN4 is assigned to core0, which runs Linux and CANOpen as master node, whereas CAN3 is assigned to core1, which runs the BareMetal and CANOpen as slave node. For LS1028A, there are two CAN controllers, CAN1 and CAN2, and both of them are used in LS1028ARDB board.

5.2.2.1 LS1021AIOT CAN resource allocation

This section describes steps for assigning CAN4 to Linux and CAN3 to BareMetal core, and how to change or configure it. These examples assume that CAN1 and CAN2 are not enabled, and the pins of CAN1 and CAN2 are used by other IPs.

1. **Assigning CAN4 to Linux**
   In Linux, the port is allocated through the DTS file. DTS file path is `industry-linux/arch/arm/boot/dts/ls1021a-iot.dts`. Content related to CAN ports is as follows:

```
/* CAN3 port */
    &can2
```

```
        {
           status = " disabled ";
        };
/* CAN4 port */
    &can3
        {
            status = "okay";
        };
```

2. **Assigning CAN3 to BareMetal**
   In BareMetal, the port is allocated through the `flexcan.c` file. The flexcan.c path is `industry-uboot/drivers/flexcan/flexcan.c`. In this file, users should define the following variables:
   a. `struct can_bittiming_t flexcan3_bittiming = CAN_BITTIM_INIT(CAN_500K);`
      Note: *Set bit timing and baud rate (500K) of the CAN port.*
   b. struct can_ctrlmode_t flexcan3_ctrlmode

```
struct can_ctrlmode_t flexcan3_ctrlmode =
{
    .loopmode = 0, /* Indicates whether the loop mode is
 enabled */
    .listenonly = 0, /* Indicates whether the only-listen
 mode is enabled */
    .samples = 0,
    .err_report = 1,
};
```

   c. struct can_init_t flexcan3

```
struct can_init_t flexcan3 =
{
    .canx = CAN3, /* Specify CAN port */
    .bt = &flexcan3_bittiming,
    .ctrlmode = &flexcan3_ctrlmode,
    .reg_ctrl_default = 0,
    .reg_esr = 0
};
```

   d. Optional parameters
      - **CAN port**

```
#define CAN3 ((struct can_module *)CAN3_BASE)
#define CAN4 ((struct can_module *)CAN4_BASE)
```

      - **Baud rate**

```
#define CAN_1000K 10
#define CAN_500K 20
#define CAN_250K 40
#define CAN_200K 50
#define CAN_125K 80
#define CAN_100K 100
#define CAN_50K 200
#define CAN_20K 500
#define CAN_10K 1000
#define CAN_5K 2000
```

5.2.2.2 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 70. CAN Net APIs and their description

API name (type)	Description
UNS8 canReceive_driver (CAN_HANDLE fd0, Message * m)	SocketCAN receives CAN messages • fd0 – SocketCAN handle • m – Receive buffer
UNS8 canSend_driver (CAN_HANDLE fd0, Message const * m)	SocketCAN sends CAN messages • fd0 – SocketCAN handle • m – CAN message to be sent
void setNodeId(CO_Data* d, UNS8 nodeId)	Set this node id value. • d – object dictionary • nodeId – id value (up to 127)
UNS8 setState(CO_Data* d, e_nodeState newState)	Set node state • d – object dictionary • newState – The state that needs to be set Returns 0 if OK, > 0 on error
void canDispatch(CO_Data* d, Message *m)	CANopen handles data frames that CAN receive. • d – object dictionary • m – Received CAN message
void timerForCan(void)	CANopen virtual clock counter.
UNS8 sendPDOrequest (CO_Data * d, UNS16 RPDOIndex)	Master node requests slave node to feedback specified data. • d – object dictionary • RPDOIndex – index value of specified data
UNS8 readNetworkDictCallback (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS8 dataType, SDOCallback_t Callback, UNS8 useBlockMode)	The master node gets the specified data from the slave node. • d – object dictionary • nodeId – the id value of slave node • index – the index value of the specified data • subIndex – the subindex value of the specified data • dataType – the data type of the specified data • Callback – callback function • useBlockMode – specifies whether it is a block transmission

Table 70. CAN Net APIs and their description*...continued*

API name (type)	Description
UNS8 writeNetworkDictCallBack (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS32 count, UNS8 dataType, void *data, SDOCallback_t Callback, UNS8 useBlockMode)	The master node sets the specified data to the slave node. • d – object dictionary • nodeId – the id value of slave node • index – the index value of the specified data • subIndex – the subindex value of the specified data • count – the length of the specified data • dataType – the data type of the specified data • Callback – callback function • useBlockMode – specifies whether it is a block transmission

5.2.3 Running a CAN application

The following sections describe the hardware and software preparation steps for running a CAN application. The hardware preparation is described separately for the LS1021A-IoT and LS1028ARDB, but the sections Section 5.2.3.3, Section 5.2.3.4, and Section 5.2.3.5 are applicable to both LS1021A-IoT and LS1028A platforms.

5.2.3.1 Hardware preparation for LS1021-IoT

For LS1021-IoT, the list of hardware required for implementing the FlexCAN demo is as follows:

- LS1021A-IoT boards
- Two CAN hardware interfaces (for example, CAN3 and CAN4 for LS1021A-IoT)
- Two CAN transceivers (for example: TJA1050)

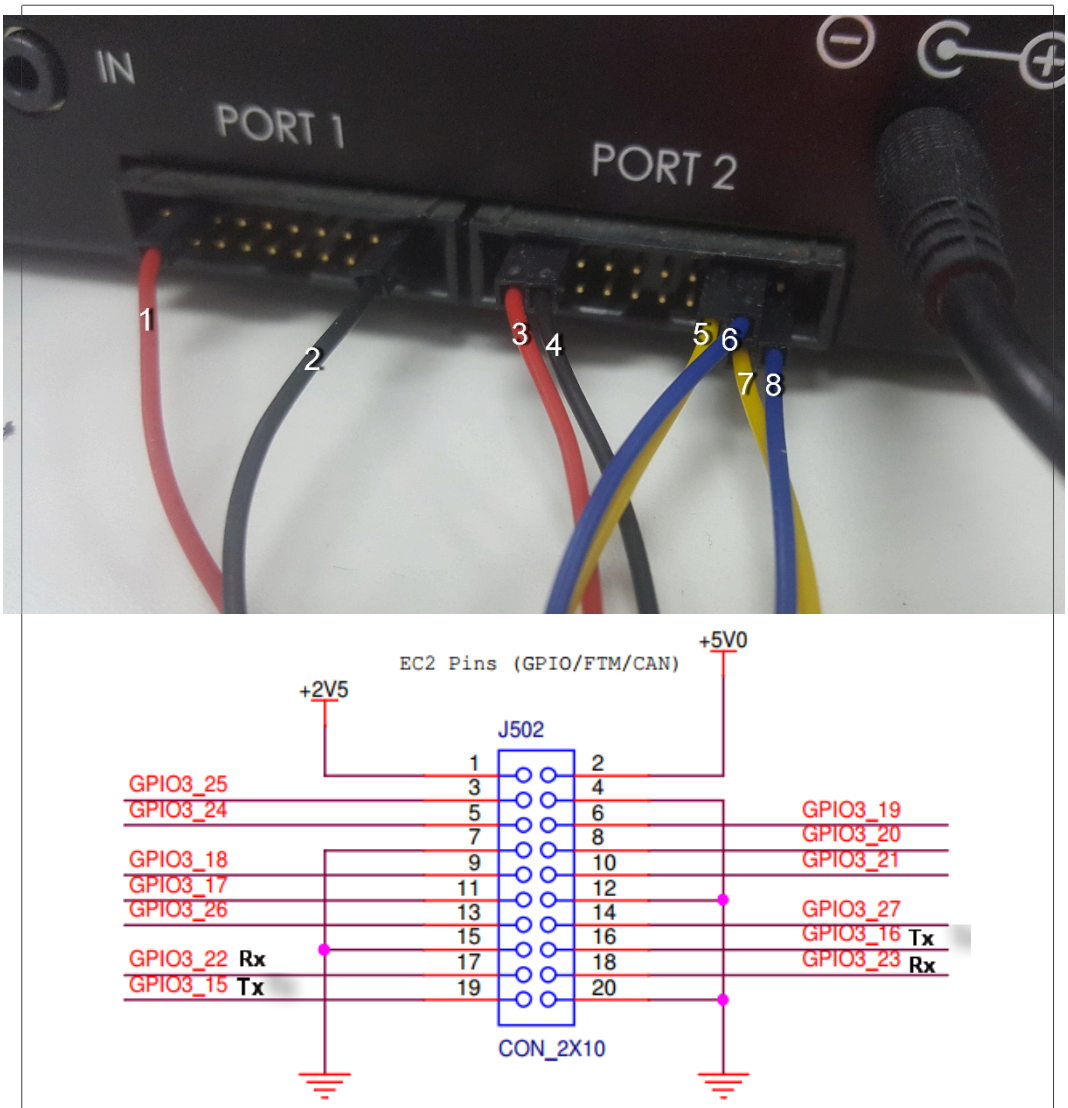


Figure 87. Hardware diagram for the FlexCan demo

Note:

– Line1 and line3 are 5.0 V.
– Line2 and line4 are GND.
– Line5 is CAN3 TX.
– Line6 is CAN3 RX.
– Line7 is CAN4 RX.
– Line8 is CAN4 TX.

5.2.3.2 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.

Figure 88. Physical connection for CAN using LS1028ARDB

5.2.3.3 Compiling the CANopen-app binary for the master node

This section describes the procedure for compiling the CANopen-app binary for the master node, for both LS1021A and LS1028A platforms.

CANopen-app is the CANopen application name. Perform the steps listed below to compile Canopen-app as Linux command to the `target/usr/bin` directory.

1. Configure cross-toolchain on user host environment.
2. Use the commands below:

```
$ cd yocto-real-time-edge/meta-real-time-edge
# open file: ./conf/distro/include/qoriq-baremetalenv.inc
# replace "ls1021aiot_baremetal_defconfig" with
 "ls1021aiot_baremetal_can_defconfig"
$ bitbake nxp-real-time-edge-baremetal
```

3. The generated Real-time Edge image file is in the `tmp/deploy/images/ls1021aiot/` directory.
4. Download the image `nxp-image-real-time-edge-ls1021aiot.wic.bz2` and decompress it and flash it to the SD card:

In U-Boot mode, first run the `tftp` command for downloading `nxp-image-real-time-edge-ls1021aiot.wic` to the buffer. Then, run the `mmc` command for downloading the `nxp-image-real-time-edge-ls1021aiot.wic` to SD card.

Note:

- *The following options are displayed only when the `canfestival` option is set to Y.*

- *Linux uses the SocketCAN interface, so the `driver`option selects the socket.*
- *Setting the `install examples` option to Y Installs binary application to the filesystem.*
- *The following `additional configure options` can be configured in the `config.h` file of CANopen:*
  Parameter description:
  – *--SDO_MAX_LENGTH_TRANSFER: Sets buffer size of SDO protocol.*
  – *-- SDO_BLOCK_SIZE: Sets the maximum number of frames that can be sent by SDO block transport protocol.*
  – *--SDO_MAX_SIMULTANEOUS_TRANSFERS: Sets the number of SDO modules.*
  .

5.2.3.4 Running the CANopen application

This section describes the procedure for running the CANopen-app application. Only the LS1021A platforms support this application.

1. First, boot the LS1021A-IoT board.
2. Waiting for the BareMetal core to output below information:

```
Note: the CANopen protocol starts to run!
=>
```

3. Then, run the `CANopen-app` command in any directory in Linux prompt. While executing this command, first run the test code.
4. After the test code is completed, user can implement the required instructions. The command `CANopen-app` execution process steps are described below:
   a. First, indicate whether the CAN interface has opened successfully. All commands are dynamically registered. Then, indicate whether the command was registered successfully.
      - **Command registration log**

```
Command Registration Log:
[root@]# CANopen-app
[   80.899975] IPv6: ADDRCONF(NETDEV_CHANGE): can0: link
 becomes ready
Note: open the CAN interface successfully!
"can_quit" command: register OK!
"setState" command: register OK!
"showPdo" command: register OK!
"requestPdo" command: register OK!
"sdo" command: register OK!
"" command: register OK!
"test_startM" command: register OK!
"test_sdoSingle" command: register OK!
"test_sdoSingleW" command: register OK!
"test_sdoBlock" command: register OK!
"test_showPdoCyc" command: register OK!
"test_showpdoreq" command: register OK!
"test_requestpdo" command: register OK!
```

   b. There are nine test code in total, tests 1 to 9. Test code details are shown in the test log.
      - **Test code log** “---test---” indicates that the test code begins.
      - Firstly, the execution rights of the SDO and PDO protocol are explained.

- The **tests 1~4** are SDO protocol test code. After starting the CANopen master node, it automatically enters into initialization and pre-operation mode.
- The **test5** is a test code that master node enters the operation mode and starts all slave nodes.
- The **tests 6~9** are PDO protocol test code.

```
Test Code Log:
----------------------- test --------------------------
Note: Test code start execute...
      SDO protocol is valid in preoperation mode, but PDO
 protocol is invalid!
      SDO and PDO protocol are both valid in operation mode!
      Console is invalid when testing!
-----------------------------------------------------------
Note: test1--Read slave node single data by SDO.
Note: master node initialization is complete!
Note: master node entry into the preOperation mode!
Note: Alarm timer is running!
Note: slave node "0x02" entry into "Initialisation" state!
-----------------------------------------------------------
Note: test2--Write 0x2CD5 to slave node by SDO.
Note: Master write a data to 0x02 node successfully.
-----------------------------------------------------------
Note: test3--Read slave node single data by SDO again.
Note: reveived data is 0x2CD5
-----------------------------------------------------------
Note: test4--Read slave node block data by SDO.
---------------- text ------------------
Note: reveived string ==>
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.
It was designed originally for motion-oriented machine control
 systems, such as handling systems.
Today it is used in various application fields, such as medical
 equipment, off-road vehicles, maritime electronics, railway
 applications, or building automation.
---------------------------------------
-----------------------------------------------------------
Note: test5--Master node entry operation mode, and start slave
 nodes!
Note: master node entry into the operation mode,and start all
 slave nodes!
-----------------------------------------------------------
Note: test6--Master node show requested PDO data.
Note: Rpdo4 data is "       "
-----------------------------------------------------------
Note: test7--Master node request PDO data.
-----------------------------------------------------------
Note: test8--Master node show requested PDO data.
Note: Rpdo4 data is "require"
Note: slave node "0x02" entry into "Operational" state!
-----------------------------------------------------------
Note: test9--Master node show received cycle PDO data.
Note: Rpdo2 data is "   cycle"
-----------------------------------------------------------
```

Note: *tests 1 to 9 are not commands.*

c. After the test code is executed, it automatically displays the list of commands. Num00~06 are normal commands. After executing these instructions without parameters, the instruction usage is displayed. Num08~14 are test commands.

All test commands except num10 have no parameters. Argument of Num10 is a 16-bit integer.

- Now the user can execute any command in the command list.

Command List

```
Command List:
-------------------------------------------------------------------
 num    |       command        |            introduction
-------------------------------------------------------------------
 00     | ctrl_quit            |  console thread exit!
-------------------------------------------------------------------
 01     |  help                |  command list
-------------------------------------------------------------------
 02     | can_quit           |  exit CANopen thread
-------------------------------------------------------------------
 03     |  setState          |  set the CANopen node state
-------------------------------------------------------------------
 04     |  showPdo           |  show the data of RPDO
-------------------------------------------------------------------
 05     |  requestPdo       |  request the data of RPDO
-------------------------------------------------------------------
 06     | sdo           |  read/write one entry by SDO protocol
-------------------------------------------------------------------
 07     |                      |
-------------------------------------------------------------------
 08     |  test_startM      |  test -- Start master
-------------------------------------------------------------------
 09     |  test_sdoSingle      |  test -- Read slave node single
 data
-------------------------------------------------------------------
 10     |  test_sdoSingleW     |  test -- Write slave node
 single data
-------------------------------------------------------------------
 11     |  test_sdoBlock       |  test -- Read slave node block
 data
-------------------------------------------------------------------
 12     |  test_showPdoCyc     |  test -- Show cycle PDO data
-------------------------------------------------------------------
 13     |  test_showpdoreq     |  test -- Show requested PDO
 data
-------------------------------------------------------------------
 14     |  test_requestpdo     |  test -- Request PDO data
-------------------------------------------------------------------
Note: User can send command by console!
Note: Test code execution is complete!
```

Example: The following example shows the usage log after running the `sdo` command without any parameters.

```
SDO Command:
sdo
usage: sdo -type index subindex nodeid data
       type = "r"(read), "w"(write), "b"(block)
       index = 0~0xFFFF,unsigned short
       subindex = 0~0xFF,unsigned char
       nodeid = 1~127,unsigned char
       data = 0 ~ 0xFFFFFFFF
```

5.2.3.5 Running the SocketCAN commands

This section describes the steps for running SocketCAN commands that can be performed on either of the boards (LS1021A-IoT or 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.

```
$ ip -details link show can0
```

Note: *The third and fourth commands are valid when the state of can0 port is closed.*

5.2.3.6 Testing CAN bus

Below is the sample code for testing the CAN bus on LS1028ARDB.

```
[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  001   [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 can1 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
```

5.3 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

5.3.1 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.

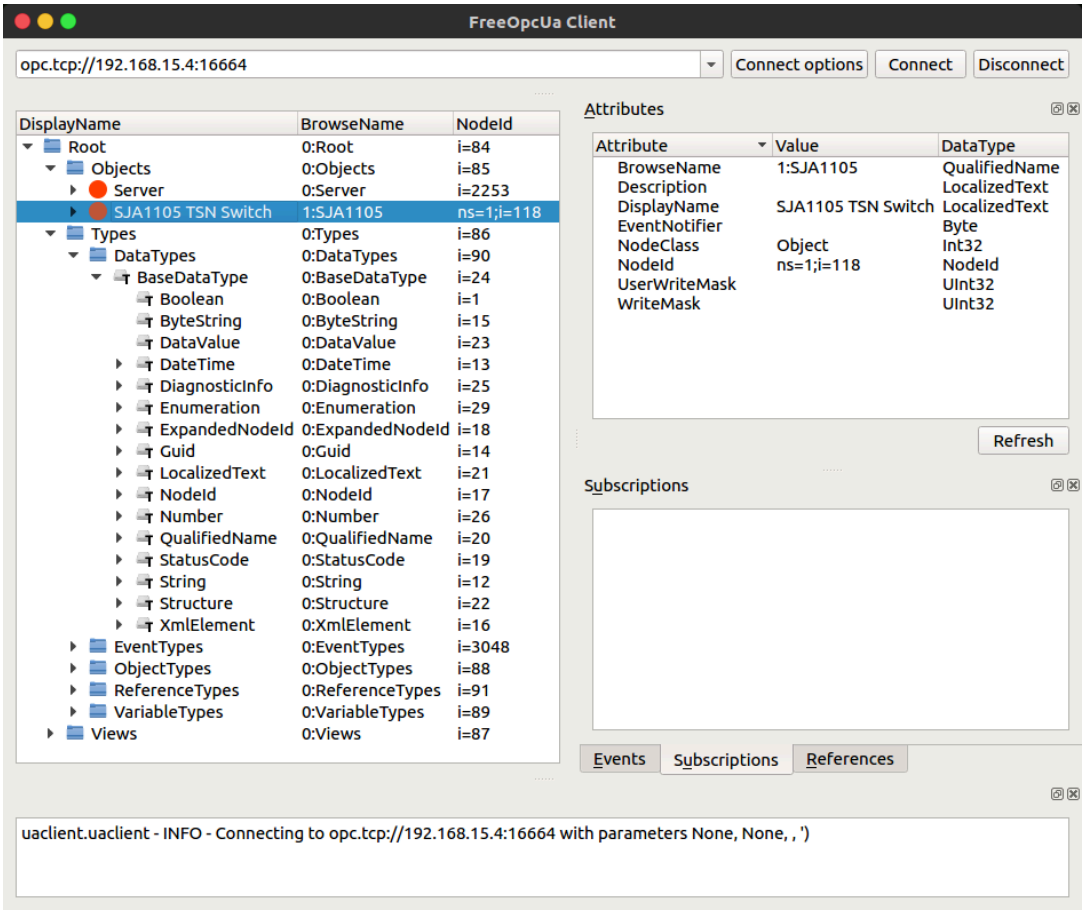
5.3.2 The node model

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 89. 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`.

5.3.3 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*.

5.3.4 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*.

5.3.5 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.

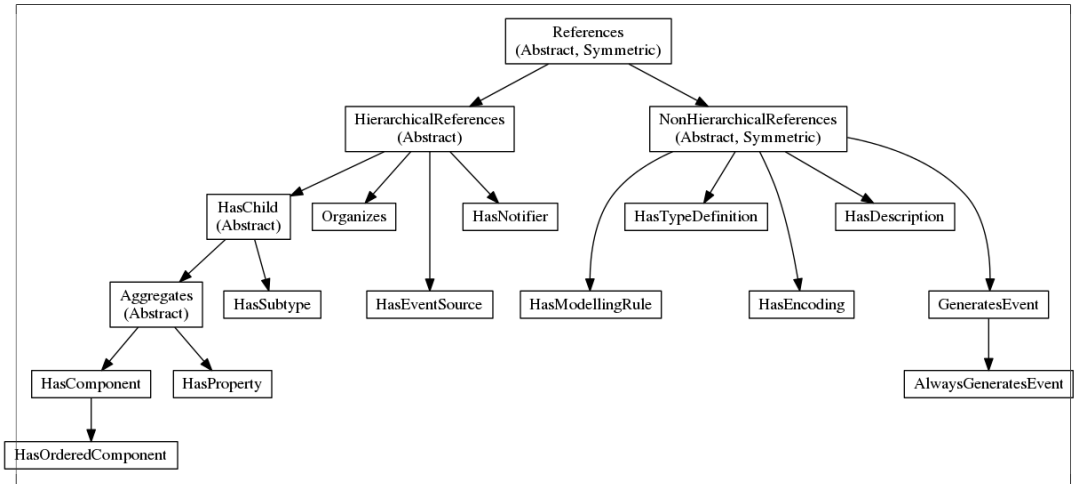


Figure 90. Hierarchy of the standard ReferenceTypes, defined in Part 3 of the OPC UA specification (Image taken from www.open62541.org)

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.

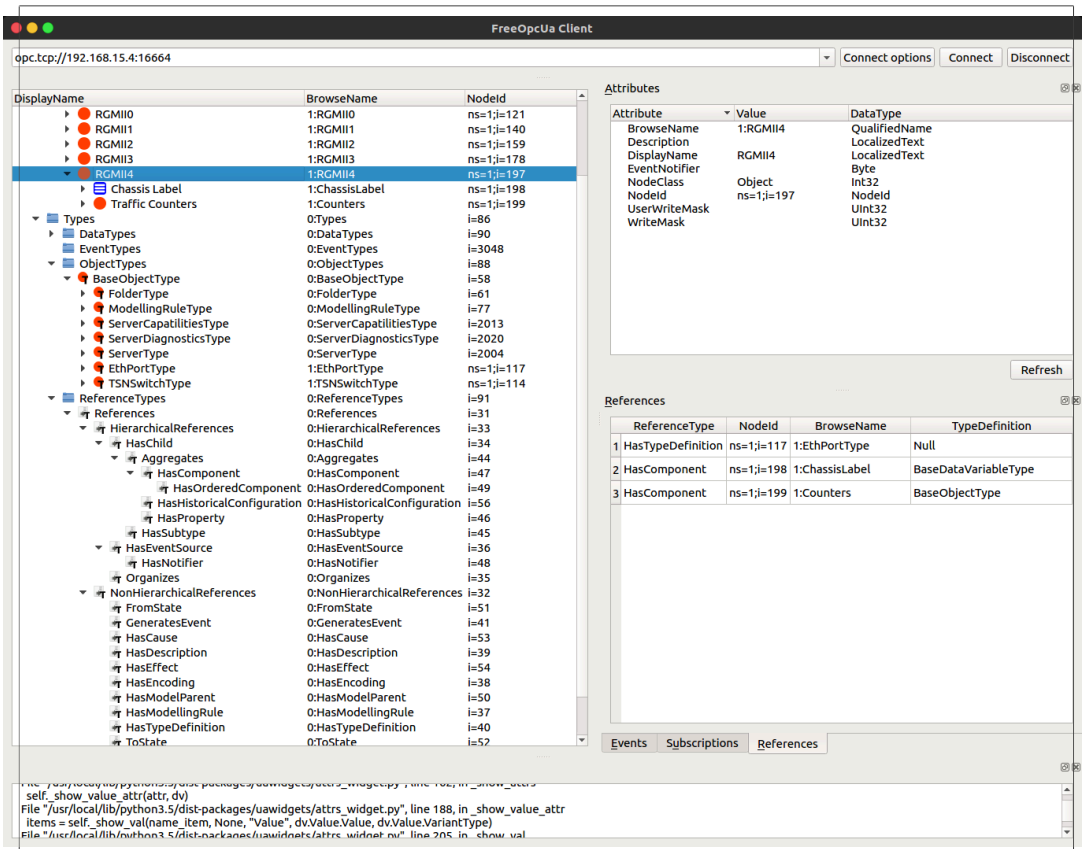


Figure 91. The 'Attributes' and 'References' views of the FreeOpcUa Client populated with details of the RGMII4 node

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`.

5.3.6 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
- open62541_access_control_server
- open62541_client
- open62541_client_async
- open62541_client_connect
- open62541_client_connectivitycheck_loop
- open62541_client_connect_loop
- open62541_client_subscription_loop
- open62541_custom_datatype_client
- open62541_custom_datatype_server
- open62541_server_ctt
- open62541_server_inheritance
- open62541_server_instantiation
- open62541_server_loglevel
- open62541_server_mainloop
- open62541_server_nodeset
- open62541_server_repeated_job
- open62541_tutorial_client_events
- open62541_tutorial_client_firststeps
- open62541_tutorial_datatypes
- open62541_tutorial_server_datasource
- open62541_tutorial_server_firststeps
- open62541_tutorial_server_method
- open62541_tutorial_server_monitoreditems
- open62541_tutorial_server_object
- open62541_tutorial_server_variable
- open62541_tutorial_server_variabletype

5.3.7 OPC UA Pub/Sub over TSN

This section introduces OPC UA PubSub and demonstrates how TSN can be used to make deterministic and reliable transmission of OPC UA PubSub traffic as well as PTP traffic on a network co-existing with best effort traffic.

5.3.7.1 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 which complements the Client/Server communication model.

In PubSub, the participating OPC UA applications can assume the roles 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 like UDP or raw Ethernet as 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.

5.3.7.2 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.

5.3.7.3 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 the DataSets, the message generation and the sending on the Publisher side. These parameters need to be known on the Subscriber side to configure DataSetReaders to filter and process DataSetMessages.

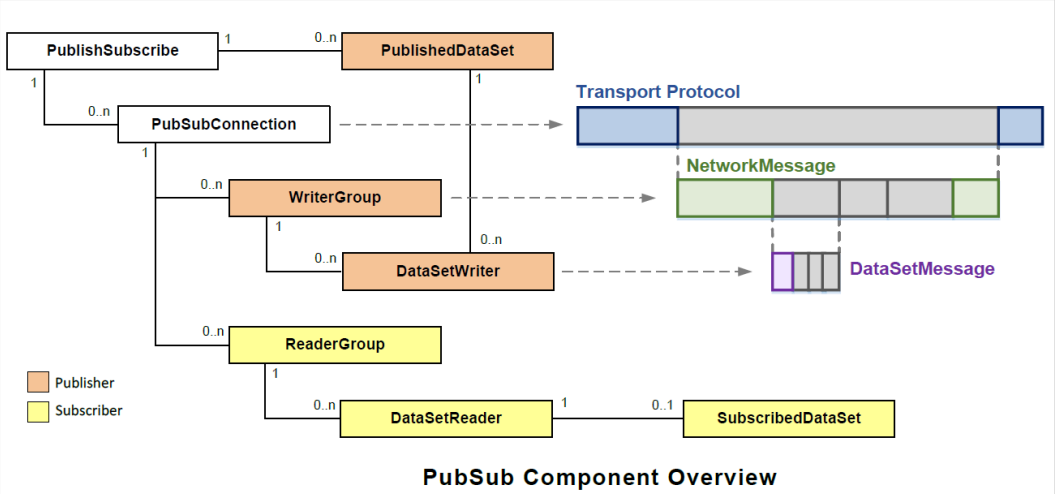


Figure 92. OPC UA PubSub components

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.

5.3.7.4 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 will transmit one packet per second cyclically. Each cycle aligns with whole second using Linux system clock 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, *opc.eth://01-00-5E-00-00-01*) and the Ethernet interface (e.g. 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. Note that the Subscriber also runs cyclically with 1 second cycle time to receive packet. Each cycle aligns with whole second with 500us offset to account for the application delay on the publisher and the path delay from publisher to subscriber. Linux system clock 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 runs a OPC UA server. User 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.

5.3.7.5 OPC UA PubSub Sample Application over TSN

Hardware Requirements:

1. Two or three i.MX8M Plus LPDDR4 EVK boards
2. One LS1028ARDB board

Software Requirements:

1. linuxptp package which provides tools like ptp4l, phc2sys, phc_ctl and hwstamp_ctl.
2. iproute2 package which provides tools like tc.
3. Open source OPC UA stack open62541 compiled as shared library (libopen62541.so).
4. OPC UA PubSub sample application opcua_pubsub_publisher and opcua_pubsub_subscriber under /home/root/open62541_example.

All the above software tools and binaries are already in the rootfs.

5.3.7.5.1 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 back-to-back via **ENET2** as shown in the following block diagram. 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 may change.

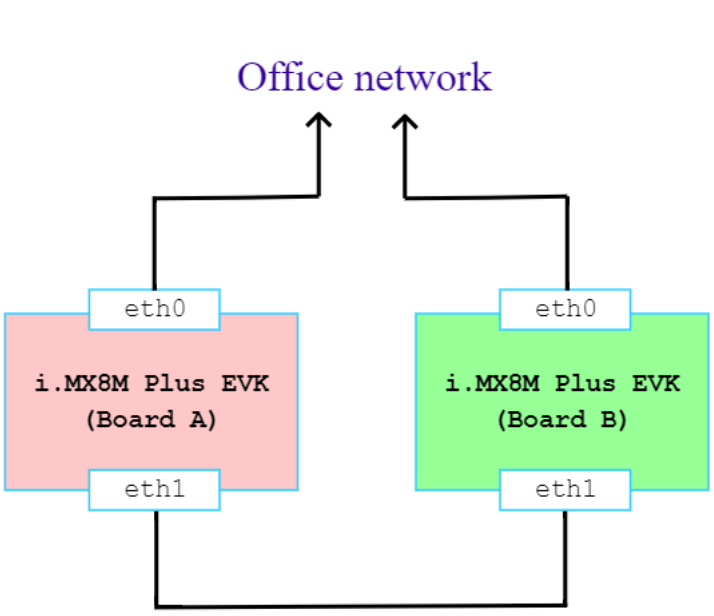


Figure 93. Two i.MX8M Plus LPDDR4 EVK boards connected back-to-back

- On both boards, bring up ENET2 (that is, eth1):

```
# ip link set eth1 up
# ethtool eth1
```

Command *ethtool eth1* should show that *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.

```
# 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):

```
# cp /etc/ptp4l_cfg/gPTP.cfg .
# sed -i 's/priority1.*248/priority1\t\t246/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):

```
# 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 both boards, we can observe the logs of ptp4l and phc2sys to check the time synchronization progress by below commands: (It is advised 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).

```
# 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[121.633]: 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 94. 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. Example phc2sys log below:

```
root@imx8mpevk:~# tail -f /var/log/phc2sys.log
phc2sys[7349.412]: CLOCK_REALTIME phc offset        61 s2 freq     +58 delay    750
phc2sys[7350.413]: CLOCK_REALTIME phc offset       -81 s2 freq     -66 delay    750
phc2sys[7351.413]: CLOCK_REALTIME phc offset       -20 s2 freq     -29 delay    750
phc2sys[7352.413]: CLOCK_REALTIME phc offset        54 s2 freq     +39 delay    750
phc2sys[7353.414]: CLOCK_REALTIME phc offset        20 s2 freq     +21 delay    750
phc2sys[7354.414]: CLOCK_REALTIME phc offset       -31 s2 freq     -24 delay    750
phc2sys[7355.414]: CLOCK_REALTIME phc offset        48 s2 freq     +46 delay    750
phc2sys[7356.415]: CLOCK_REALTIME phc offset        -7 s2 freq      +5 delay    750
phc2sys[7357.415]: CLOCK_REALTIME phc offset       -57 s2 freq     -47 delay    750
phc2sys[7358.416]: CLOCK_REALTIME phc offset        20 s2 freq     +13 delay    750
phc2sys[7359.416]: CLOCK_REALTIME phc offset       -28 s2 freq     -29 delay    750
phc2sys[7360.416]: CLOCK_REALTIME phc offset        26 s2 freq     +16 delay    750
phc2sys[7361.417]: CLOCK_REALTIME phc offset        20 s2 freq     +18 delay    750
phc2sys[7362.417]: CLOCK_REALTIME phc offset       -14 s2 freq     -10 delay    750
^C
```

Figure 95. 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 500us 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 sched-entry S 0x05 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:

```
root@imx8mpevk:~# /home/root/open62541_example/opcua_pubsub_publisher -u opc.eth://01-00-5E-00-00-01 -d eth1
[2020-09-20 11:02:20.030 (UTC+0000)] info/userland      Transport Profile   : http://opcfoundation.org/UA-Profile/Transport/pubsub-eth-uadp
[2020-09-20 11:02:20.030 (UTC+0000)] info/userland      Network Address URL : opc.eth://01-00-5E-00-00-01
[2020-09-20 11:02:20.030 (UTC+0000)] info/userland      Ethernet Interface  : eth1
[2020-09-20 11:02:20.482 (UTC+0000)] warn/server        AccessControl: Unconfigured AccessControl. Users have all permissions.
[2020-09-20 11:02:20.482 (UTC+0000)] info/server        AccessControl: Anonymous login is enabled
[2020-09-20 11:02:20.482 (UTC+0000)] warn/server        Username/Password configured, but no encrypting SecurityPolicy. This can leak credentials on the network.
[2020-09-20 11:02:20.482 (UTC+0000)] warn/userland      AcceptAll Certificate Verification. Any remote certificate will be accepted.
[2020-09-20 11:02:20.483 (UTC+0000)] info/userland      PubSub channel requested
[2020-09-20 11:02:20.483 (UTC+0000)] info/server        Open PubSub ethernet connection.
[2020-09-20 11:02:20.483 (UTC+0000)] info/userland      Publisher socket FD : 3
[2020-09-20 11:02:20.483 (UTC+0000)] info/userland      Publisher Tx HW timestamp : Enabled
[2020-09-20 11:02:20.484 (UTC+0000)] info/userland      Publisher cycle time : 1000.000000 ms
[2020-09-20 11:02:20.484 (UTC+0000)] info/userland      Publisher thread priority : 78
[2020-09-20 11:02:20.484 (UTC+0000)] info/userland      Publisher on CPU core : 3
[2020-09-20 11:02:20.484 (UTC+0000)] info/userland      Publisher thread callback Id: 281472837534032
[2020-09-20 11:02:20.484 (UTC+0000)] info/network       TCP network layer listening on opc.tcp://imx8mpevk:4840/
[2020-09-20 11:02:20.485 (UTC+0000)] info/userland      Starting the publisher cycle at 1600599745.000000000
```

Figure 96. Example log on the Publisher

Example log on the Subscriber:

```
root@imx8mpevk:~# /home/root/open62541_example/opcua_pubsub_subscriber -u opc.eth://01-00-5E-00-00-01 -d eth1
[2020-09-20 11:02:22.297 (UTC+0000)] info/userland      Transport Profile   : http://opcfoundation.org/UA-Profile/Transport/pubsub-eth-uadp
[2020-09-20 11:02:22.297 (UTC+0000)] info/userland      Network Address URL : opc.eth://01-00-5E-00-00-01
[2020-09-20 11:02:22.297 (UTC+0000)] info/userland      Ethernet Interface  : eth1
[2020-09-20 11:02:22.747 (UTC+0000)] warn/server        AccessControl: Unconfigured AccessControl. Users have all permissions.
[2020-09-20 11:02:22.747 (UTC+0000)] info/server        AccessControl: Anonymous login is enabled
[2020-09-20 11:02:22.747 (UTC+0000)] warn/server        Username/Password configured, but no encrypting SecurityPolicy. This can leak credentials on the network.
[2020-09-20 11:02:22.747 (UTC+0000)] warn/userland      AcceptAll Certificate Verification. Any remote certificate will be accepted.
[2020-09-20 11:02:22.748 (UTC+0000)] info/userland      PubSub channel requested
[2020-09-20 11:02:22.748 (UTC+0000)] info/server        Open PubSub ethernet connection.
[2020-09-20 11:02:22.748 (UTC+0000)] info/userland      Subscriber socket FD : 3
[2020-09-20 11:02:22.748 (UTC+0000)] info/userland      Subscriber Rx HW timestamp : Enabled
[2020-09-20 11:02:22.748 (UTC+0000)] info/userland      Subscriber thread priority : 81
[2020-09-20 11:02:22.749 (UTC+0000)] info/userland      Publisher on CPU core : 3
[2020-09-20 11:02:22.749 (UTC+0000)] info/userland      Subscriber thread callback Id: 281473844617552
[2020-09-20 11:02:22.749 (UTC+0000)] info/userland      Starting the subscriber cycle at 1600599743.000500000
[2020-09-20 11:02:22.750 (UTC+0000)] info/network       TCP network layer listening on opc.tcp://imx8mpevk:4801/
[2020-09-20 11:02:23.001 (UTC+0000)] info/userland      Packet Sequence Number : 0
[2020-09-20 11:02:24.001 (UTC+0000)] info/userland      Packet Sequence Number : 0
[2020-09-20 11:02:25.000 (UTC+0000)] info/userland      Packet Sequence Number : 0
[2020-09-20 11:02:26.001 (UTC+0000)] info/userland      Packet Sequence Number : 1
[2020-09-20 11:02:27.000 (UTC+0000)] info/userland      Packet Sequence Number : 2
[2020-09-20 11:02:28.000 (UTC+0000)] info/userland      Packet Sequence Number : 3
[2020-09-20 11:02:29.000 (UTC+0000)] info/userland      Packet Sequence Number : 4
[2020-09-20 11:02:30.001 (UTC+0000)] info/userland      Packet Sequence Number : 5
[2020-09-20 11:02:31.000 (UTC+0000)] info/userland      Packet Sequence Number : 6
[2020-09-20 11:02:32.001 (UTC+0000)] info/userland      Packet Sequence Number : 7
[2020-09-20 11:02:33.001 (UTC+0000)] info/userland      Packet Sequence Number : 8
```

Figure 97. Example log on the Subscriber

- 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:

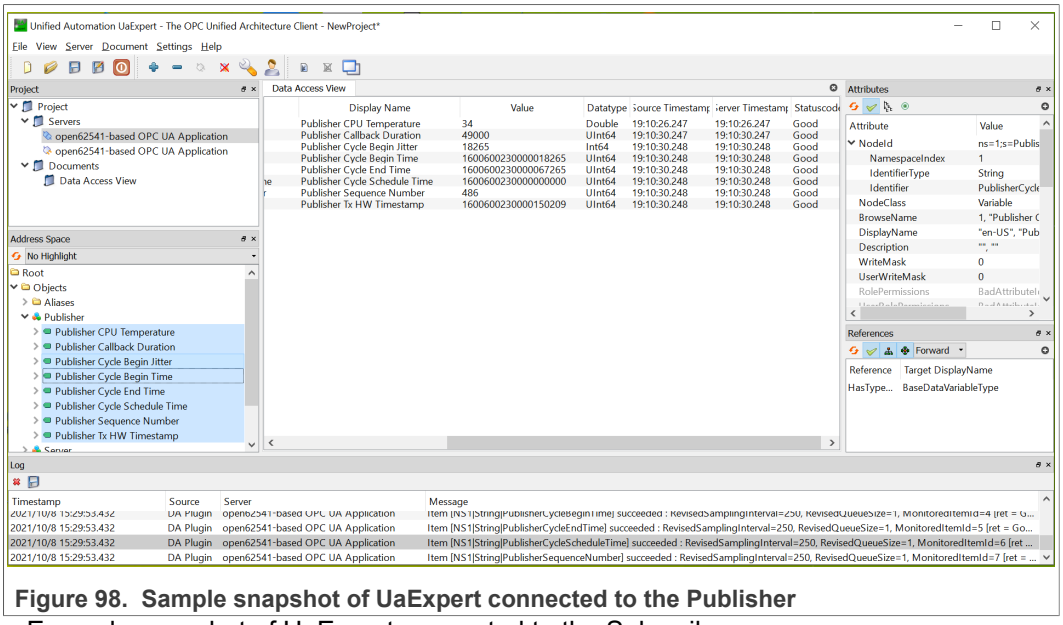


Figure 98. Sample snapshot of UaExpert connected to the Publisher

Example snapshot of UaExpert connected to the Subscriber:

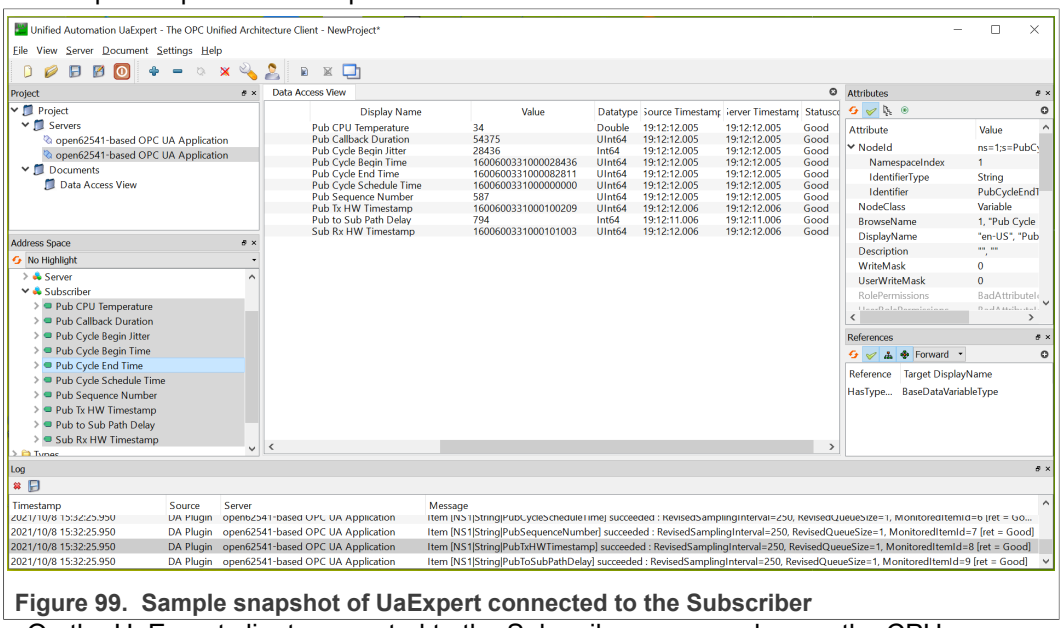


Figure 99. Sample snapshot of UaExpert connected to the Subscriber

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:

Figure 100. Sample error log of ptp4l on the Publisher

```
ptp4l[436.575]: timed out while polling for tx timestamp
ptp4l[436.575]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a driver bug
ptp4l[436.575]: port 1: send sync failed
ptp4l[436.575]: port 1: MASTER to FAULTY on FAULT_DETECTED (FT_UNSPECIFIED)
ptp4l[436.600]: selected local clock 00049f.fffe.06fe8b as best master
ptp4l[436.600]: port 1: assuming the grand master role
```

Example error log of ptp4l on the Subscriber

Figure 101. Sample error log of ptp4l on the Subscriber

```
ptp4l[431.404]: rms   11 max   16 freq   -152 +/-  14 delay   777 +/-   0
ptp4l[432.406]: rms    8 max   20 freq   -155 +/-  10 delay   777 +/-   0
ptp4l[433.408]: rms    4 max    8 freq   -146 +/-   4 delay   777 +/-   0
ptp4l[433.784]: port 1: SLAVE to MASTER on ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[433.784]: selected local clock 00049f.fffe.06fe83 as best master
ptp4l[433.784]: port 1: assuming the grand master role
ptp4l[452.679]: selected best master clock 00049f.fffe.06fe8b
ptp4l[452.680]: port 1: MASTER to SLAVE on RS_SLAVE
ptp4l[453.680]: rms  731 max 1281 freq  +1135 +/- 357 delay   777 +/-   0
ptp4l[454.682]: rms  220 max  304 freq   +157 +/- 200 delay   778 +/-   0
ptp4l[455.683]: rms  285 max  320 freq   -252 +/-  54 delay   778 +/-   0
ptp4l[456.684]: rms  155 max  220 freq   -299 +/-  18 delay   778 +/-   0
ptp4l[457.687]: rms   43 max   79 freq   -231 +/-  27 delay   777 +/-   0
```

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.

5.3.7.5.2 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.MX8M 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.

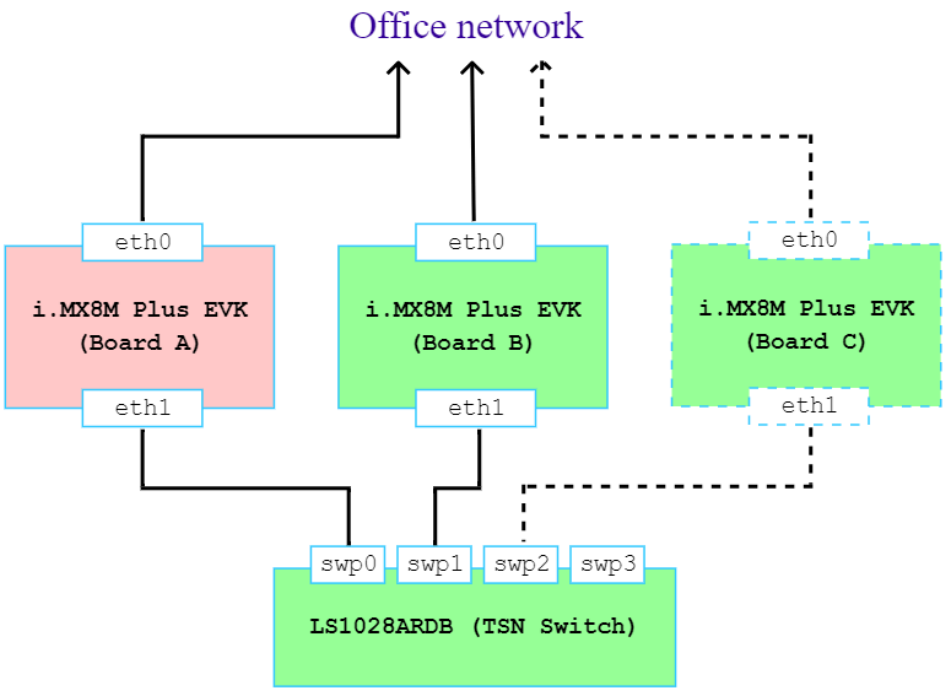


Figure 102. Two i.MX 8M Plus LPDDR4 EVK boards connected to LS1028ARDB TSN switch

The following steps assumes 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 will 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.

```
# 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.MX8M Plus LPDDR4 EVK - Board A) and the Subscriber (i.MX8M Plus LPDDR4 EVK - Board B), bring up ENET2 (that is, eth1):

```
# ip link set eth1 up
```

- On the Publisher (i.MX8M 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.

```
# 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:

```
# 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.MX8M Plus LPDDR4 EVK - Board A):

```
# cp /etc/ptp4l_cfg/gPTP.cfg .
# sed -i 's/priority1.*248/priority1\t\t246/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):

```
# 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, we can observe the logs of ptp4l and phc2sys to check the time synchronization progress by below commands: (It is advised to SSH to each board to check the logs of ptp4l and phc2sys so that we can continue to execute other commands on the serial console).

```
# 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, 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 ENET2to 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 queue2 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 500000 sched-entry S 0x05 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, 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 (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 1ms so that the schedule is aligned to 1ms as 1ms. 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. It is optional in this use case and used as 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 500000 sched-entry S 0x05 500000
 flags 0x2
# tc -g qdisc show dev swp1
```

With the above TSN Qbv configuration on egress switch port swp1, OPC UA PubSub traffic will go into TX hardware queue 2 (We will add VLAN header with PCP field set to 2 for OPC UA PubSub packet). The best effort traffic will go into TX hardware queue 4 (We will 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 we won't miss any packet sent by the Publisher.
  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.

```
# /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:

Figure 103. Example log on the Publisher

```
root@imx8mpevk:~# /home/root/open62541_example/opcua_pubsub_publisher -u opc.eth://01-00-5E-00-00-01:100.2 -d eth1
[2020-09-20 10:18:51.436 (UTC+0000)] info/userland      Transport Profile    : http://opcfoundation.org/UA-Profile/Transport/pubsub-
eth-uadp
[2020-09-20 10:18:51.441 (UTC+0000)] info/userland      Network Address URL : opc.eth://01-00-5E-00-00-01:100.2
[2020-09-20 10:18:51.441 (UTC+0000)] info/userland      Ethernet Interface  : eth1
[2020-09-20 10:18:51.920 (UTC+0000)] warn/server        AccessControl: Unconfigured AccessControl. Users have all permissions.
[2020-09-20 10:18:51.920 (UTC+0000)] info/server        AccessControl: Anonymous login is enabled
[2020-09-20 10:18:51.920 (UTC+0000)] warn/server        Username/Password configured, but no encrypting SecurityPolicy. This can le
ak credentials on the network.
[2020-09-20 10:18:51.921 (UTC+0000)] warn/userland      AcceptAll Certificate Verification. Any remote certificate will be accepted
.
[2020-09-20 10:18:51.921 (UTC+0000)] info/userland      PubSub channel requested
[2020-09-20 10:18:51.921 (UTC+0000)] info/server        Open PubSub ethernet connection.
[2020-09-20 10:18:51.922 (UTC+0000)] info/userland      Publisher socket FD : 3
[2020-09-20 10:18:51.922 (UTC+0000)] info/userland      Publisher Tx HW timestamp : Enabled
[2020-09-20 10:18:51.922 (UTC+0000)] info/userland      Publisher cycle time : 1000.000000 ms
[2020-09-20 10:18:51.922 (UTC+0000)] info/userland      Publisher thread priority : 78
[2020-09-20 10:18:51.922 (UTC+0000)] info/userland      Publisher on CPU core : 3
[2020-09-20 10:18:51.922 (UTC+0000)] info/userland      Publisher thread callback Id: 281473694241104
[2020-09-20 10:18:51.922 (UTC+0000)] info/userland      Starting the publisher cycle at 1600597136.000000000
[2020-09-20 10:18:51.923 (UTC+0000)] info/network       TCP network layer listening on opc.tcp://imx8mpevk:4840/
```

Example log on the Subscriber:

```
root@imx8mpevk:~# /home/root/open62541_example/opcua_pubsub_subscriber -u opc.eth://01-00-5E-00-00-01:100.2 -d eth1
[2020-09-20 10:19:43.151 (UTC+0000)] info/userland      Transport Profile    : http://opcfoundation.org/UA-Profile/Transport/pubsub-
eth-uadp
[2020-09-20 10:19:43.151 (UTC+0000)] info/userland      Network Address URL : opc.eth://01-00-5E-00-00-01:100.2
[2020-09-20 10:19:43.151 (UTC+0000)] info/userland      Ethernet Interface  : eth1
[2020-09-20 10:19:43.598 (UTC+0000)] warn/server        AccessControl: Unconfigured AccessControl. Users have all permissions.
[2020-09-20 10:19:43.599 (UTC+0000)] info/server        AccessControl: Anonymous login is enabled
[2020-09-20 10:19:43.599 (UTC+0000)] warn/server        Username/Password configured, but no encrypting SecurityPolicy. This can le
ak credentials on the network.
[2020-09-20 10:19:43.599 (UTC+0000)] warn/userland      AcceptAll Certificate Verification. Any remote certificate will be accepted
.
[2020-09-20 10:19:43.600 (UTC+0000)] info/userland      PubSub channel requested
[2020-09-20 10:19:43.600 (UTC+0000)] info/server        Open PubSub ethernet connection.
[2020-09-20 10:19:43.600 (UTC+0000)] info/userland      Subscriber socket FD : 3
[2020-09-20 10:19:43.600 (UTC+0000)] info/userland      Subscriber Rx HW timestamp : Enabled
[2020-09-20 10:19:43.600 (UTC+0000)] info/userland      Subscriber thread priority : 81
[2020-09-20 10:19:43.600 (UTC+0000)] info/userland      Publisher on CPU core : 3
[2020-09-20 10:19:43.600 (UTC+0000)] info/userland      Subscriber thread callback Id: 281472940425552
[2020-09-20 10:19:43.600 (UTC+0000)] info/userland      Starting the subscriber cycle at 1600597184.000500000
[2020-09-20 10:19:43.602 (UTC+0000)] info/network       TCP network layer listening on opc.tcp://imx8mpevk:4801/
[2020-09-20 10:19:44.001 (UTC+0000)] info/userland      Packet Sequence Number : 0
[2020-09-20 10:19:45.000 (UTC+0000)] info/userland      Packet Sequence Number : 0
[2020-09-20 10:19:46.000 (UTC+0000)] info/userland      Packet Sequence Number : 1
[2020-09-20 10:19:47.000 (UTC+0000)] info/userland      Packet Sequence Number : 2
[2020-09-20 10:19:48.000 (UTC+0000)] info/userland      Packet Sequence Number : 3
[2020-09-20 10:19:49.001 (UTC+0000)] info/userland      Packet Sequence Number : 4
[2020-09-20 10:19:50.000 (UTC+0000)] info/userland      Packet Sequence Number : 5
[2020-09-20 10:19:51.000 (UTC+0000)] info/userland      Packet Sequence Number : 6
[2020-09-20 10:19:52.000 (UTC+0000)] info/userland      Packet Sequence Number : 7
[2020-09-20 10:19:53.000 (UTC+0000)] info/userland      Packet Sequence Number : 8
```

Figure 104. Example log on the Subscriber

- 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:
  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/
  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/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 that 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 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 stats of TX packets of swp1 by using the below command:

```
# 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
root@ls1028ardb:~#
```

Figure 105. Sample log after checking the stats of TX packets of swp1

5.3.8 OPC UA Client Installation and Usage

5.3.8.1 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.

https://www.unified-automation.com/downloads/opc-ua-clients.html

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.

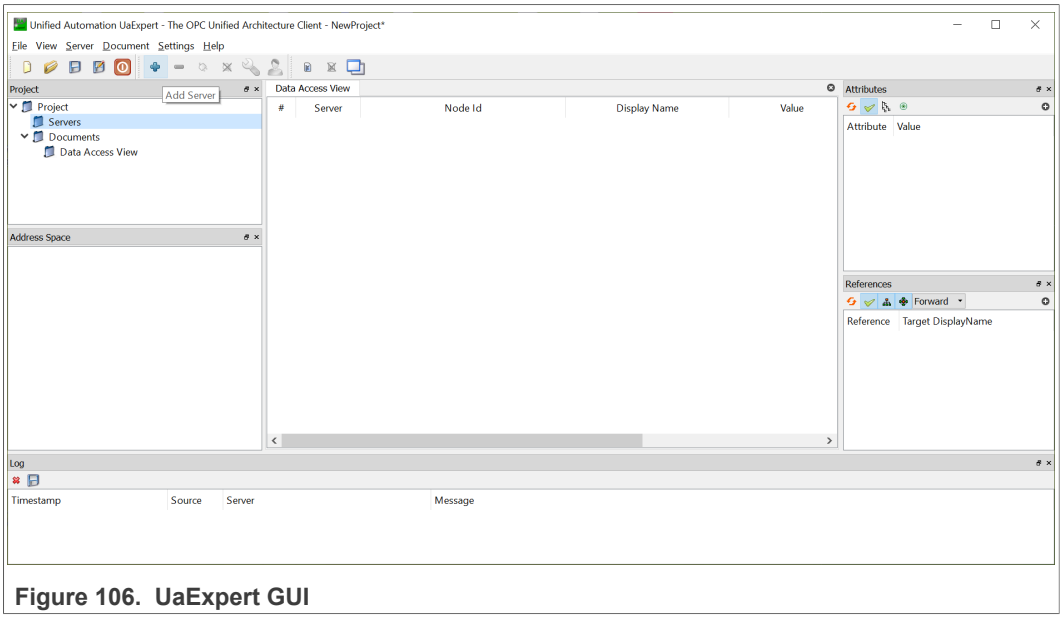


Figure 106. UaExpert GUI

- 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**.

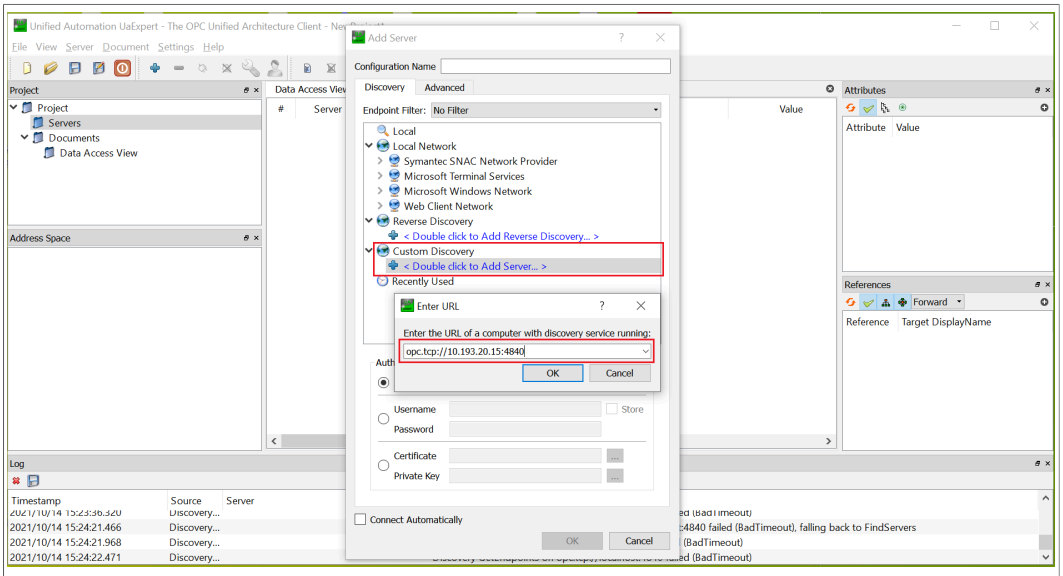


Figure 107. Adding the OPC UA server

- 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**'.

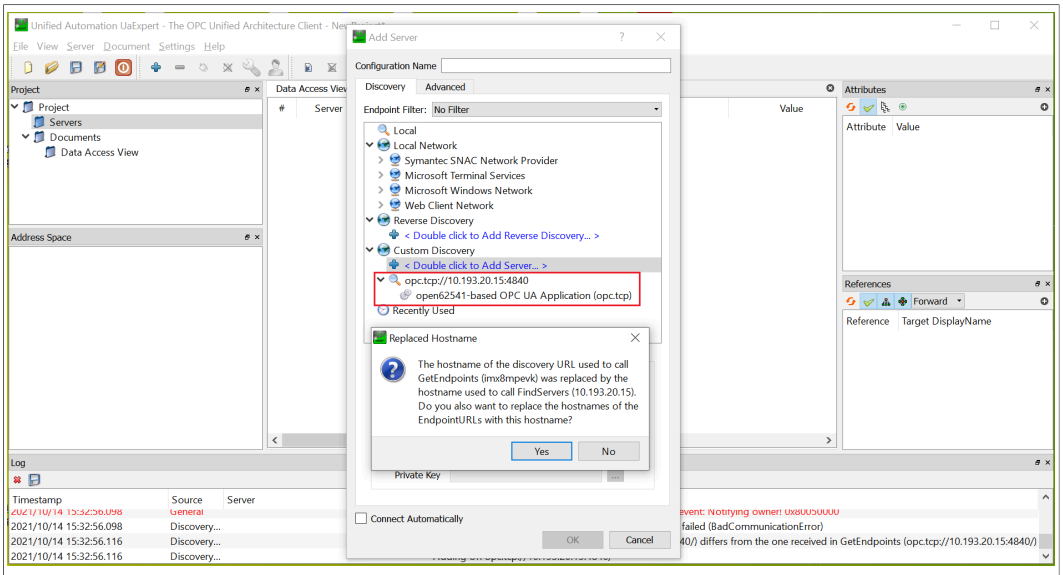


Figure 108. Server listed under Custom Discovery

- Click to select '**None – None (...)**' and click **OK**.

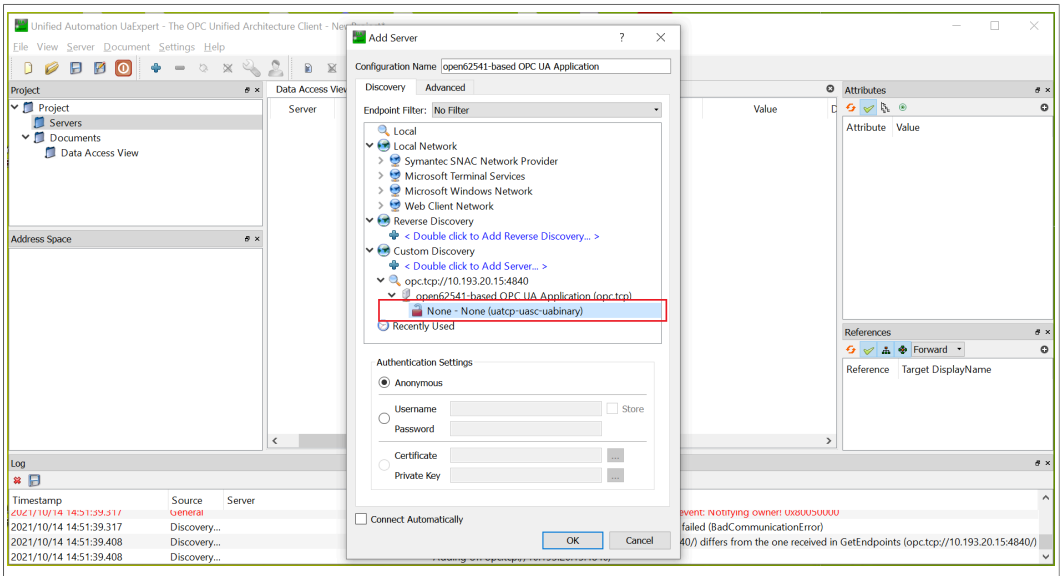


Figure 109. Selecting the hostname

- Right click on the server listed under ‘Servers’ and click ‘Connect’.

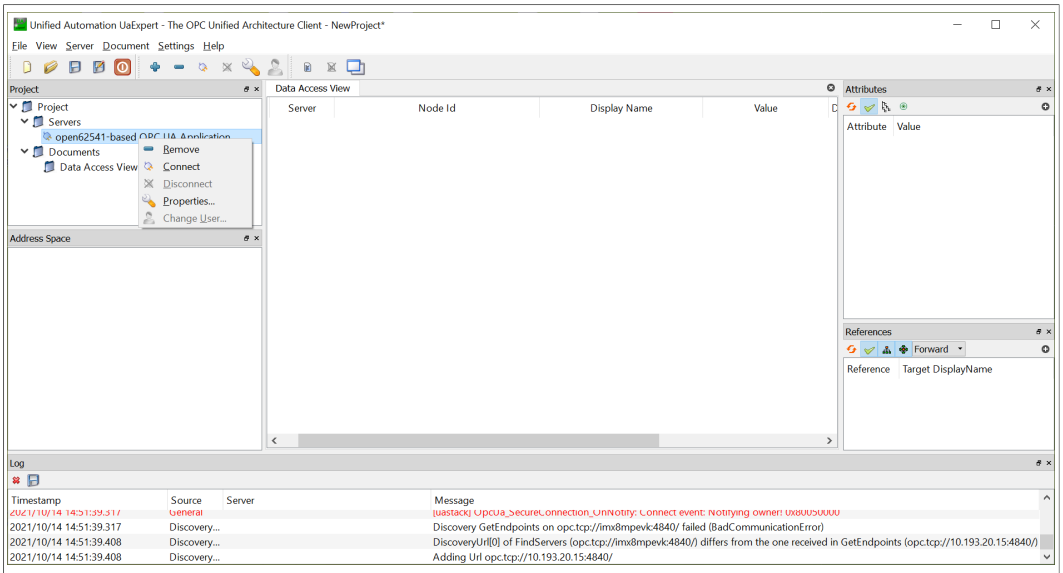


Figure 110. Connecting to the server

- 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.

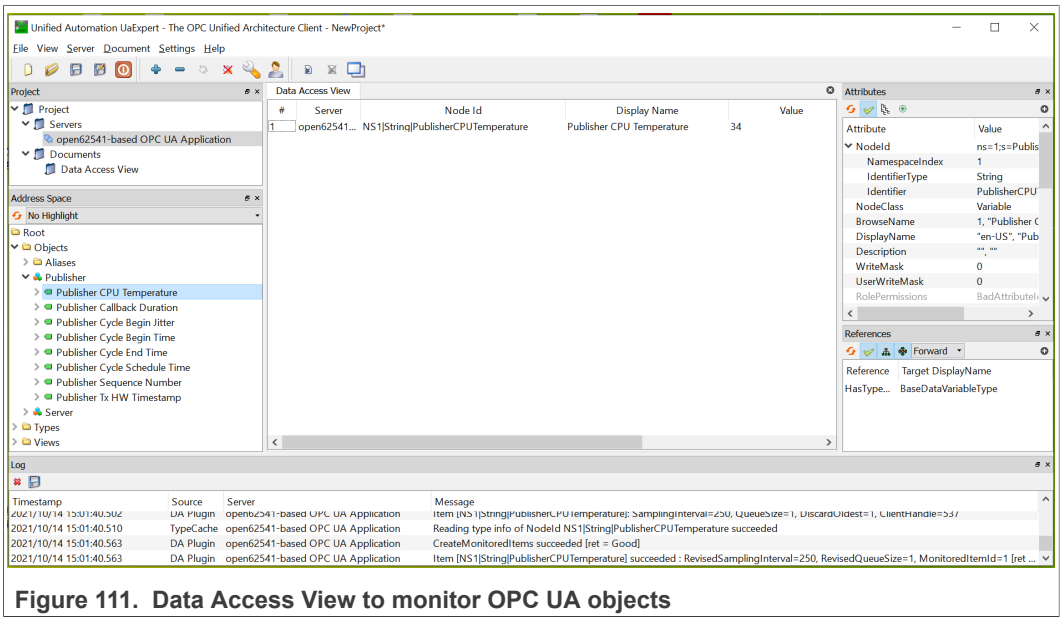


Figure 111. Data Access View to monitor OPC UA objects

5.3.8.2 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 freeopcua python api and pyqt. Use below command to install it on a Linux PC using pip3. Make sure python3 and python3-pip is installed.

```
$ 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

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:

```
$ 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.

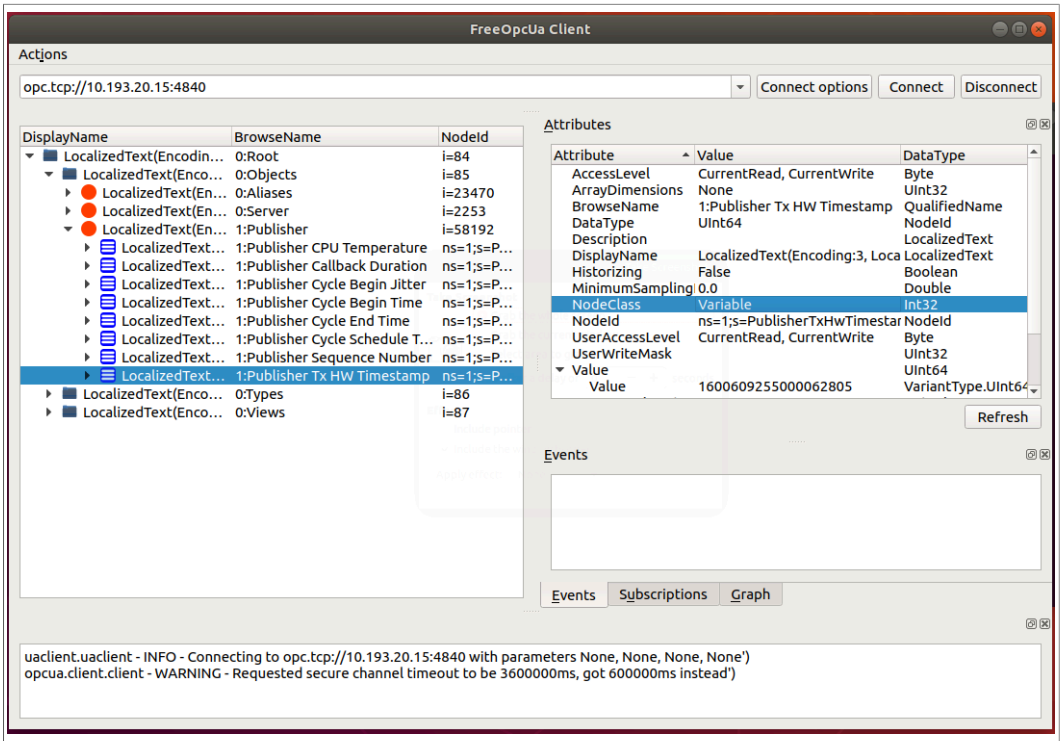


Figure 112. FreeOpcUa GUI client Connect options

5.4 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.

5.4.1 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 71. The NETCONF layers

Layer	Purpose	Example
1	Content	Configuration data, Notification data
2	Operations	<edit-config>
3	Messages	<rpc>, <rpc-reply>, <notification>
4	Secure	Transport SSH

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.

5.4.2 Netopeer2

5.4.2.1 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.

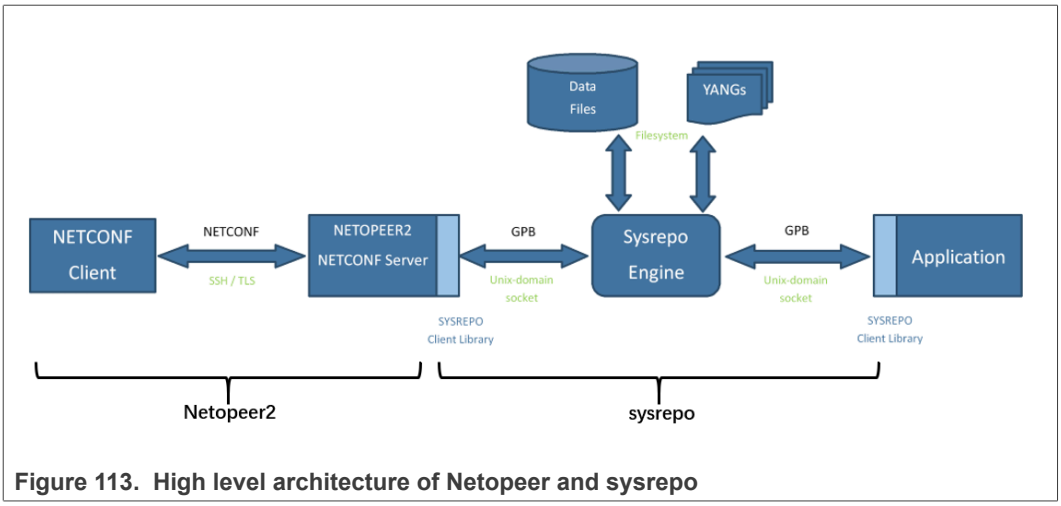


Figure 113. High level architecture of Netopeer and sysrepo

5.4.2.2 Installing Netopeer2-cli on Ubuntu18.04

Use the following steps for installing Netopeer2-cli on Ubuntu18.04 operating systems.

1. Install the following packages:

```
$ 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:

```
$ 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:PATH=/usr ..
$ make
$ sudo make install
```

3. Install sysrepo (v0.7.8):

```
$ 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 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
```

5.4.2.3 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 programing language that supports GPB is possible).

For information about sysrepo, see:

http://www.sysrepo.org/static/doc/html/index.html

5.4.2.4 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 RFCs2 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.

5.4.2.5 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.

5.4.2.6 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.

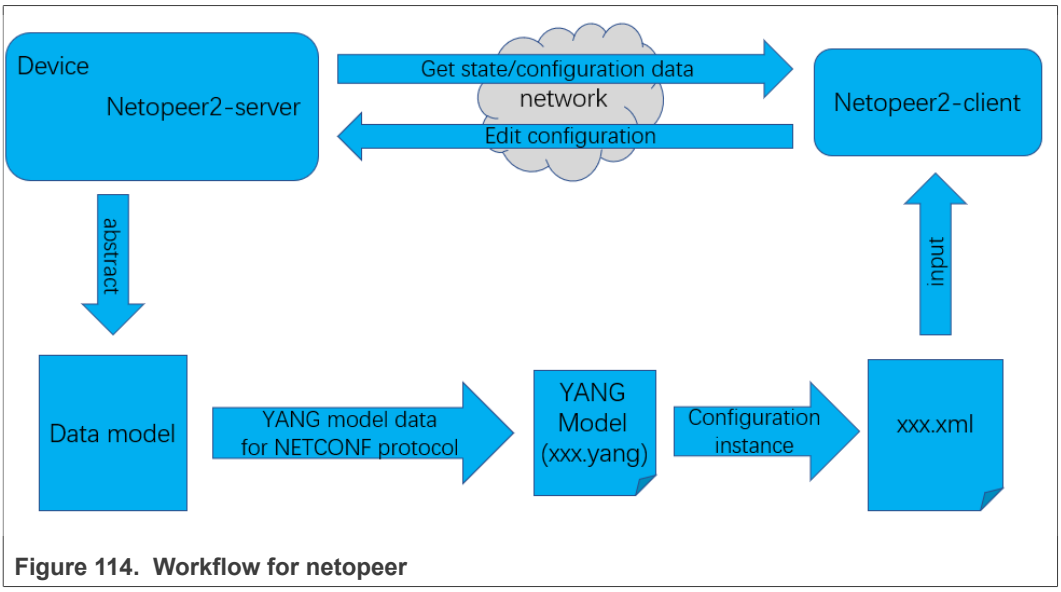


Figure 114. Workflow for netopeer

5.4.3 Configuration

5.4.3.1 Enabling NETCONF feature

This feature is enabled by default in Real-time Edge software, build the image using the below command

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

or

```
DISTRO=nxp-real-time-edge MACHINE=ls1021atsn source real-time-
edge-setup-env.sh -b build-ls1021atsn
```

Below packages are enabled by default in Real-time Edge software:

```
netopeer2-keystored netopeer2-server real-time-edge-sysrepo
```

sysrepo-tsn is daemon application to implement tsn configuration based on **sysrepo**. It was enabled for **LS1028ARDB**, **LS1021A-TSN** and **i.MX 8M Plus LPDDR4 EVK**.

Note:

- *For LS1028ARDB board, Qbv, Qbu, Qci, stream identification in CB, IP, MAC, and VLAN are supported.*
- *For LS1021ATSN board, Qbv, IP, MAC and VLAN are supported.*
- *real-time-edge-sysrepo was only verified in environment build by **LS1028ARDB**, **LS1021ATSN** and **i.MX 8M Plus LPDDR4 EVK**.*

5.4.3.2 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).

5.4.3.3 Netopeer2-cli

The netopeer2-cli is 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.

5.4.3.3.1 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.

   ```
   connect [--help] [--ssh] [--host <hostname>] [--port <num>]
    [--login <username>]
   ```

   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.

```
copy-config [--help] --target running|startup|candidate|
url:<url> (--source running|startup|candidate|url:<url> | --
src-config[=<file>])
    [--defaults report-all|report-all-tagged|trim|explicit]
```

   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 5.4.3.3.2.
   - **--source** datastore: Specifies the source datastore for the `copy-config` operation. For description of the datastore parameter, refer to Section 5.4.3.3.2.
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 5.4.3.3.2. 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:

```
get [--help] [--filter-subtree[=<file>] | --filter-xpath
 <XPath>] [--defaults report-all|report-all-tagged|trim|
explicit] [--out <file>]
```

   - **--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.

```
get-config [--help] --source running|startup|candidate [--
filter-subtree[=<file>] | --filter-xpath <XPath>]
```

```
    [--defaults report-all|report-all-tagged|trim|explicit]
 [--out <file>]
```

10. **--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.
11. **--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.
12. **--target**
    - Target datastore to retrieve. For description of possible values, refer to Section 5.4.3.3.2. Note, that the url configuration datastore cannot be retrieved.
13. **lock**
    Performs the NETCONF `lock` operation to lock the entire configuration datastore of a server. For details, see RFC 6241 section 7.5.

    ```
    lock [--help] --target running|startup|candidate
    ```

    Where the
    - **--target**: specifies the target datastore to lock. For description of possible values, refer to Section 5.4.3.3.2. Note, that the url configuration datastore cannot be locked.
14. **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.

    ```
    unlock [--help] --target running|startup|candidate
    ```

    where
    - **--target**: specifies the target datastore to unlock. For description of possible values, refer to Section 5.4.3.3.2. Note, that the url configuration datastore cannot be unlocked.
15. **verb**
    - Enables/disables verbose messages.
16. **quit**
    - Quits the program.

5.4.3.3.2 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.

5.4.3.4 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

5.4.3.5 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 \fBsysrepocfg\fP and can immediately take the new configuration into account.

5.4.3.6 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
```

5.4.3.7 List of yang models

Table 72. Revision of yang models

YANG models name	Official revision	List of the developer changes	Draft or Published
ieee802-types	2020-10-23		DRAFT
ieee802-dot1q-types	2020-10-23		DRAFT
ietf-interfaces	2018-02-20		DRAFT
iana-if-type	2020-01-10	- Delete duplicate revision	PUBLISHED

Table 72. Revision of yang models*...continued*

YANG models name	Official revision	List of the developer changes	Draft or Published
ietf-yang-types	2013-07-15		PUBLISHED
ieee802-dot1q-bridge	2020-11-24	- Add prefix of bridge-type and bridge-component	DRAFT
ieee802-dot1q-sched	2020-07-07	- Add prefix of gate operation-name - Allow gate-control-entry to be empty - Delete cycle-time limited	DRAFT
ieee802-dot1q-preemption	2020-07-07		DRAFT
ieee802-dot1cb-stream-identification-types	2021-06-14		DRAFT
ieee802-dot1cb-stream-identification	2021-05-06		DRAFT
ieee802-dot1q-stream-filters-gates	2020-11-06		PUBLISHED
ieee802-dot1q-psfp	2020-07-07	- Add prefix of set-gate-and-ipv - Delete cycle-time limited	DRAFT
ietf-ip	2018-02-22		RFC
ieee802-dot1q-pb	2020-11-24		DRAFT
ieee802-dot1q-qci-augment	2019-05-20	- Add by NXP	PROPRIETARY
ieee802-dot1cb-frer-types	2021-05-06		DRAFT
ieee802-dot1cb-frer	2021-05-06		DRAFT
nxp-bridge-vlan-tc-flower	2020-04-02	- Add by NXP	PROPRIETARY

5.4.3.8 Operation examples

The following figure describes the steps to configure device via netopeer2:

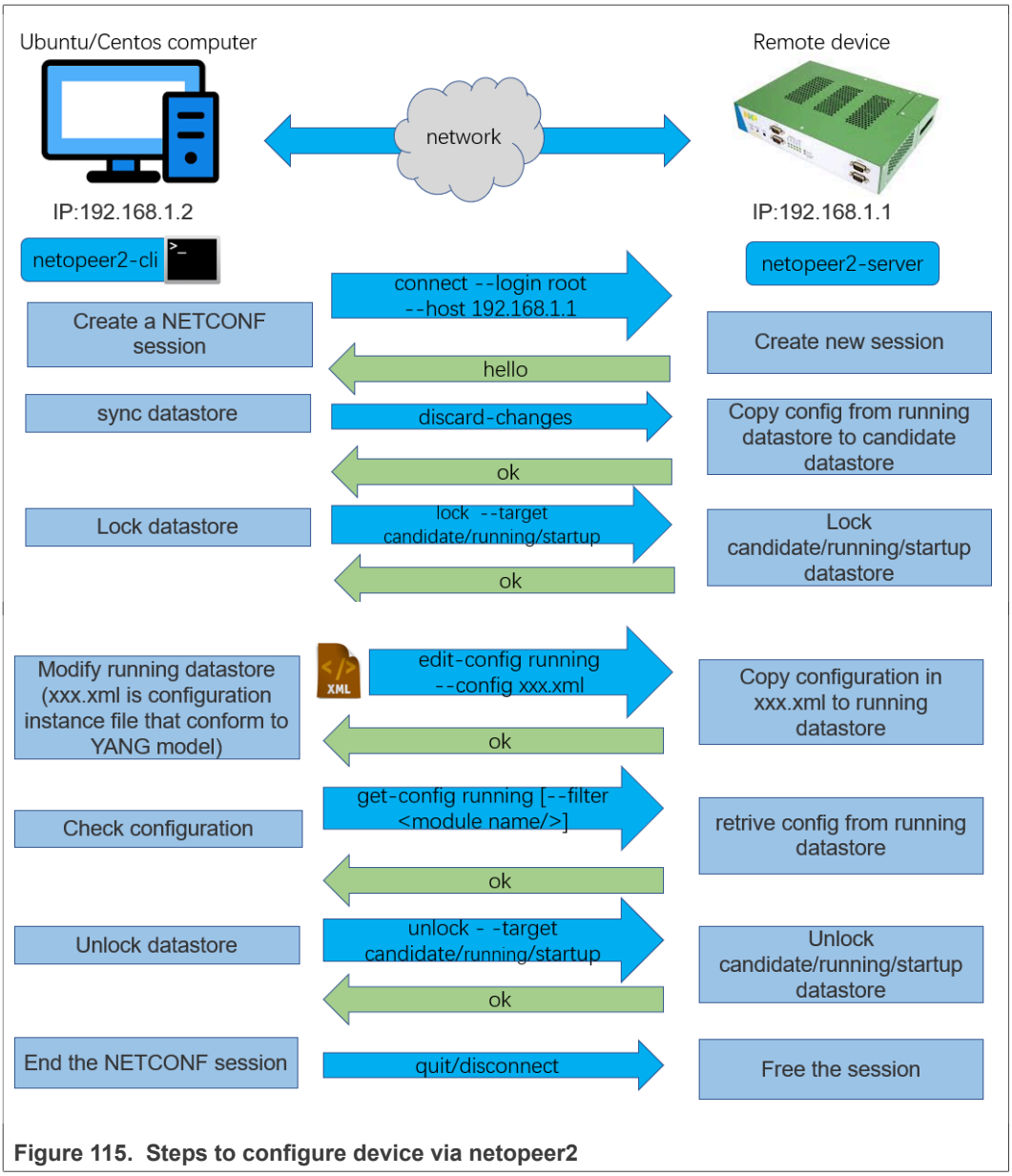


Figure 115. Steps to configure device via netopeer2

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 **sysrepod**, **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
```

5.4.3.9 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.*

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 notopeer2-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.
   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 SJA1105 Qbv via tc
   a. Edit and send configuration file

```
> edit-config --target running --config=qbv-swp5-tc.xml
```

   b. Show the result:

```
# tc qdisc show dev swp5
```

   Note: *Due to hardware limitation of SJA1105, if you want to config `qbv` again, you should delete the former `qdisc` using the following command:*

```
# tc qdisc del dev swp5 parent root handle 100
```

10. Configure SJA1105 Qci gate via tc
    a. Create 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-gate-
swp2-enable.xml
```

c. Show the result:

```
# tc filter show dev swp2 ingress
```

Note: *If you want to test Qci flow meter, you must send* `switch-qci-fm-swp2-enable.xml`.

Note:

The engine used to schedule the ingress gate operations is the same that the one used for the tc-taprio offload. Therefore, the restrictions regarding the fact that no two gate actions (either tc-gate or tc-taprio gates) may fire at the same time (during the same 200 ns slot) still apply.

11. Configure i.MX 8M Plus 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
```

12. Configure i.MX 8M Plus 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
```

5.4.4 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.

5.5 Graphics on LS1028A

This chapter is applicable to LS1028A. For i.MX 8M Plus and 8M Mini, refer to i.MX Graphics User's Guide for verification of features.

5.5.1 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 monitor.
- For i.MX 8M Plus LPDDR4 EVK, connect the monitor and i.MX 8M PlusLPDDR4 EVK with HDMI cable.

Insert the USB mouse into USB port in the keyboard.

Step 3: Running GPU demo

OpenCL demo (example A and B) just suites to LS1028ARDB and i.MX 8M Plus LPDDr4 EVK. i.MX 8M Mini LPDDR4 EVK does 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_DEVICE_PROFILE:
 FULL_PROFILE
                 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
                         CL_DEVICE_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...
log2(fft size) = log2(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

kmscube is used to test OpenGL ES, it supports HDMI and eDP interface.

For eDP interface, 4K resolution is not supported due to firmware limitation.

```
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"
  vendor: "Vivante Corporation"
  renderer: "Vivante GC7000UL"
  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.

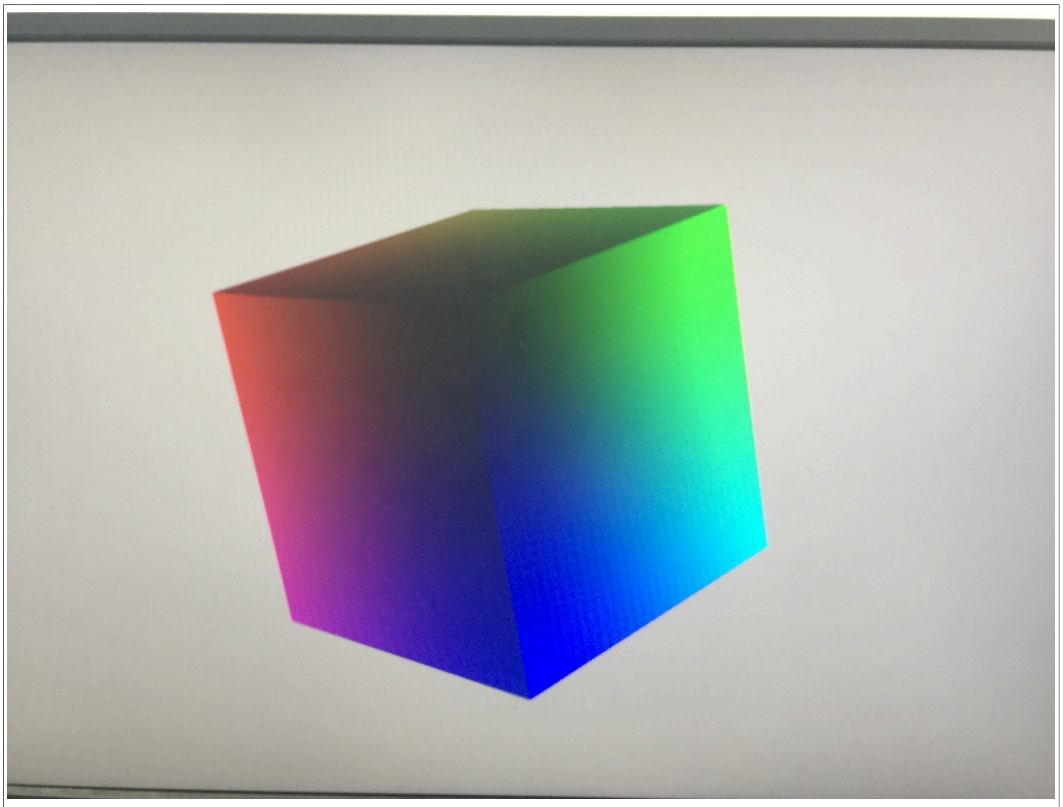


Figure 116. OpenGL ES demo with kmscube

5.5.2 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

```
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/
               Build: 8.0.0
[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_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
[14:38:00.224] EGL_KHR_surfaceless_context available
[14:38:00.310] GL version: OpenGL ES 3.1 V6.4.0.p2.234062
[14:38:00.311] 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_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
[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 'DELL 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@60.3, 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
[14:39:23.341] libinput: configuring device "Logitech USB
 Optical Mouse".
[14:39:23.342] associating input device event0 with output
 DP-1 (none by udev)
could not load cursor 'dnd-move'
could not load cursor 'dnd-copy'
could not load cursor 'dnd-none'
[14:39:33.459] event0  - Logitech USB Optical Mouse: device
 removed
[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] libinput: configuring device "Dell Dell USB
 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] libinput: configuring device "Logitech USB
 Optical Mouse".
[14:40:11.758] associating input device event0 with output
 DP-1 (none by udev)
```

```
[14:40:19.403] event0  - Logitech USB Optical Mouse: device
 removed
[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] libinput: configuring device "Logitech USB
 Optical Mouse".
[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)
```

Below is the snapshot.

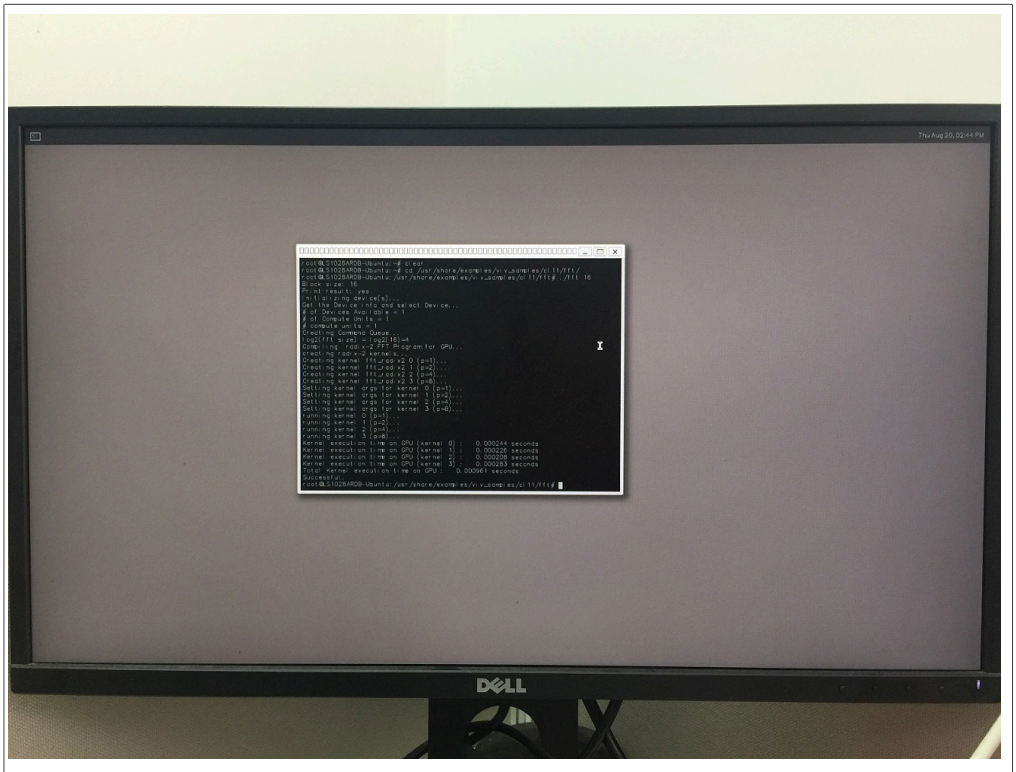


Figure 117. Weston demo

5.5.3 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 5.5.2 to enable them. Users should follow the steps below to enable `gstreamer` on a target board.

1. **Software setting and configuration**
   `Gstream` 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:

Figure 118. MIPI-DSI to HDMI interface

3. MIPI-CSI camera module: The following figure shows the MIPI-CSI camera module:

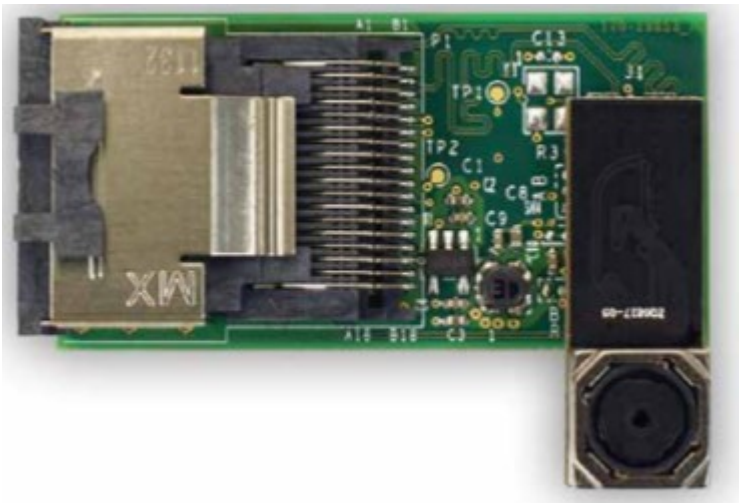


Figure 119. MIPI-CSI camera module

4. Run `gstreamer` for camera.
   After entering Linux prompt, run `gstreamer` command as shown in the below steps:

```
[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`**:

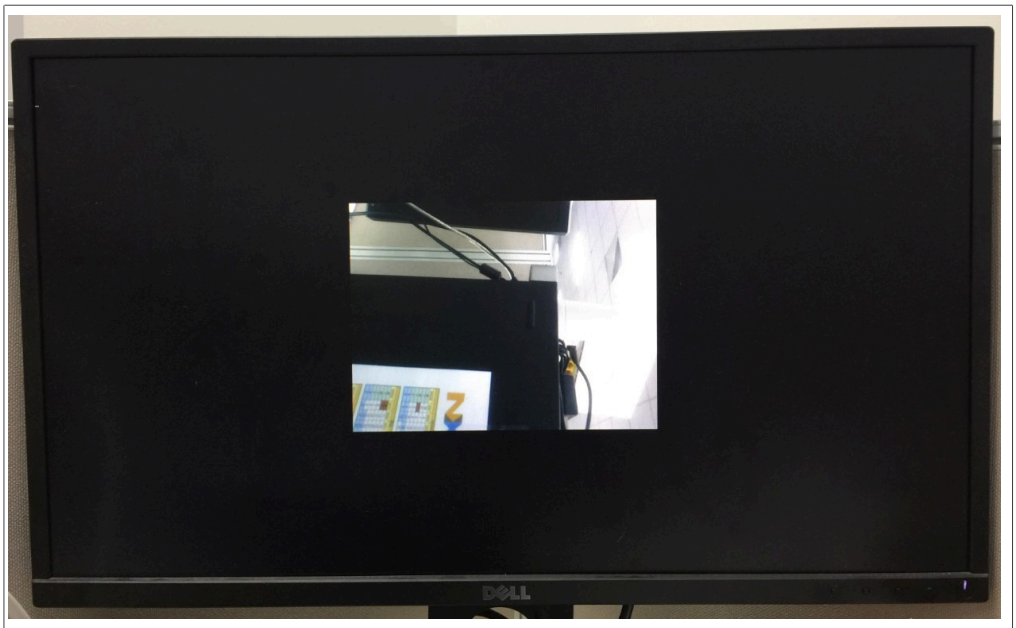


Figure 120. `fbdevsink`

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:

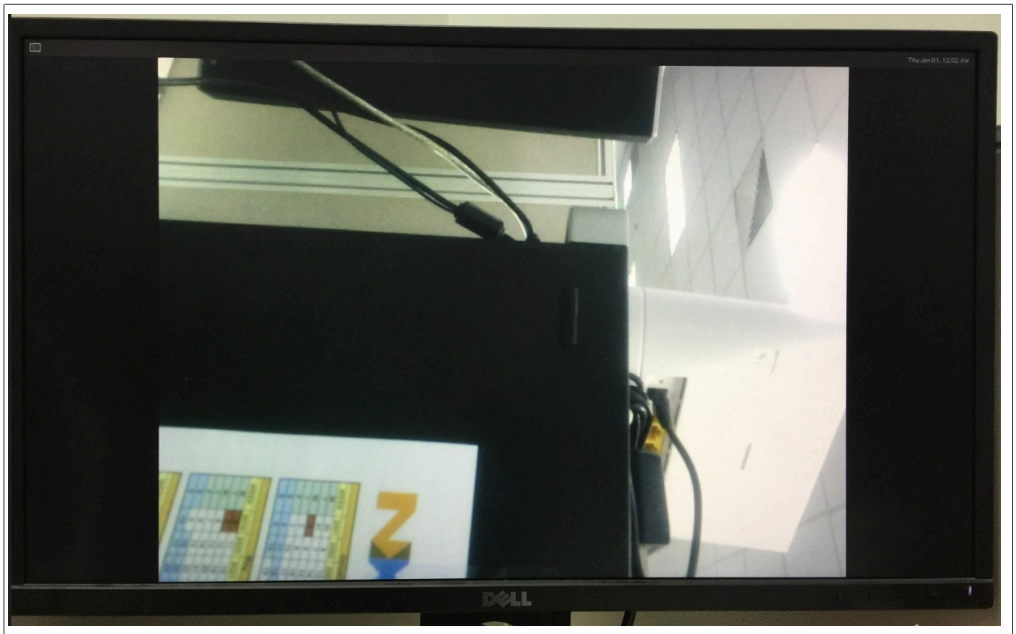


Figure 121. Waylandsink implementation

5.5.4 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.

```
camera.set(cv2.CAP_PROP_FPS, 30)
```

5.6 Wireless on LS1028A

5.6.1 NFC

NFC click board is a mikroBUS add-on board with a versatile near field communications controller from NXP — the PN7120 IC. 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

5.6.1.1 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.

5.6.1.2 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.

5.6.1.3 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.*

5.6.1.4 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.

5.6.1.5 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.

```
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/[  195.547601] random: crng init
done
[  195.551016] random: 5 urandom warning(s) missed due to ratelimiting
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/drivers/misc/nxp-pn5xx
/pn5xx_i2c.ko
[  777.503246] pn54x_dev_init
[  777.506048] pn54x_probe
[  777.508523] pn544 7-0028: FIRM GPIO <OPTIONAL> error getting from OF node
[  777.515344] pn544 7-0028: CLKREQ GPIO <OPTIONAL> error getting from OF node
[  777.522347] pn544 7-0028: 7-0028 supply nxp,pn54x-pvdd not found, using dummy regulator
[  777.530424] pn544 7-0028: 7-0028 supply nxp,pn54x-vbat not found, using dummy regulator
[  777.538490] pn544 7-0028: 7-0028 supply nxp,pn54x-pmuvcc not found, using dummy regulato
r
[  777.546723] pn544 7-0028: 7-0028 supply nxp,pn54x-sevdd not found, using dummy regulator
```

3. Run the **nfcDemoApp** application:

```
[root]# nfcDemoApp poll
```

```
[root@OpenIL:~]# nfcDemoApp poll
################################################################################
##                                    NFC demo                                  ##
################################################################################
##                                Poll mode activated                           [ 1251.20807
1] pn54x_dev_open : 10,55
        ##
####[ 1251.212807] pn54x_dev_ioctl, cmd=1074063617, arg=1
################[ 1251.219006] pn544_enable power on
#####################################################################
                            ... press enter to quit ...

[ 1251.431597] pn54x_dev_ioctl, cmd=1074063617, arg=0
[ 1251.436416] pn544_disable power off
[ 1251.647586] pn54x_dev_ioctl, cmd=1074063617, arg=1
[ 1251.652401] pn544_enable power on
NfcHcpX:8103
NfcHcpR:8180
NfcHcpX:810103020304
NfcHcpR:8180
NfcHcpX:81010143da67663bda6766
NfcHcpR:8180
NfcHcpX:810204
NfcHcpR:818000
Waiting for a Tag/Device...
```

4. Put the NFC Sample Card (tag) on top of the NFC click board:

```
Waiting for a Tag/Device...

        NFC Tag Found

        Type :          'Type A - Mifare Ul'
        NFCID1 :         '04 67 66 D2 9C 39 81 '
                Record Found :
                                NDEF Content Max size :     '868 bytes'
                                NDEF Actual Content size :     '29 bytes'
                                ReadOnly :                      'FALSE'
                Read NDEF URL Error

                29 bytes of NDEF data received :
                D1 01 19 55 01 6E 78 70 2E 63 6F 6D 2F 64 65 6D 6F 62 6F 61 72 64 2F 4F 4D
35 35 37 38

        NFC Tag Lost

Waiting for a Tag/Device...
```

Display of the above information indicates successful card reading.

5.6.2 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).

5.6.2.1 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.

5.6.2.2 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.

5.6.2.3 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

5.6.2.4 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:

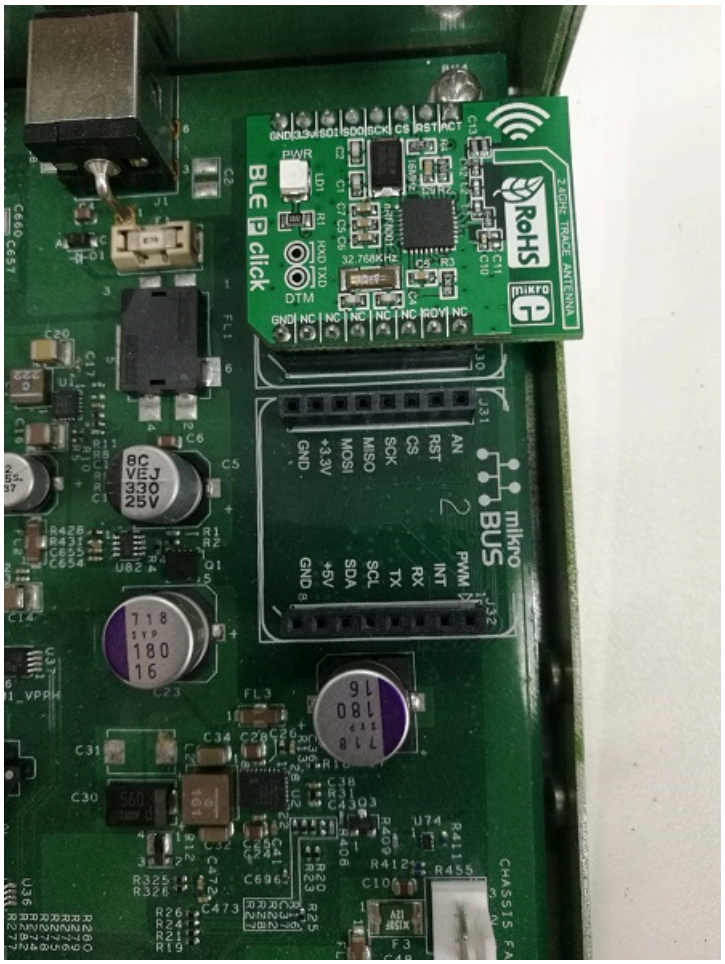


Figure 122. BLE P Click board hardware setup

5.6.2.5 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:

```
Device Drivers --->
    SPI support --->
        <*>   Freescale DSPI controller
        <*>   User mode SPI device driver support
```

  3. Use the `make` command to create the images.

5.6.2.6 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**”.

```
root@OpenIL-Ubuntu:~# blep_demo
spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 KHz)
Please input a command!
 Event device started: Setup
Error:no
Start setup command
.............................
Setup complete
 Event device started: Standby
 Advertising started : Tap Connect on the nRF UART app
Error:no
Send broadcast command successfully
```

Figure 123. Initialization log

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.

```
Evt Connected
Evt Pipe Status
Evt link connection interval changed
ConnectionInterval:0x0006
SlaveLatency:0x0000
SupervisionTimeout:0x01F4
Evt Pipe Status
Evt link connection interval changed
ConnectionInterval:0x0027
SlaveLatency:0x0000
SupervisionTimeout:0x01F4
```

Figure 124. Connection log

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:

```
Evt Disconnected
Advertising started : Tap Connect on the nRF UART app Send broadcast command successfully
```

Figure 125. Disconnection log

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
Please input a command!
Device address:DC:E2:6C:17:07:45
```

Figure 126. devaddr command log

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.

```
name=ble_demo
Name set. New name: ble_demo, 8
Please input a command!
Unknow event:0x00
Set local data successfully
```

Figure 127. name log

c. **version**

This command is used to obtain the version of the BLE P click board. User can run this command at any time.

```
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)
```

Figure 128. version log

d. **echo**

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:

```
echo hi
Please input a command!
Unknow event:0x00
 ACI Evt Pipe Error: Pipe #9
 Pipe Error Code: 0x83
 Pipe Error Data: 0x00
 Please connect the device before sending data
```

Figure 129. User input log (no connection)

The below log is displayed when user sends a string after a connection is established:

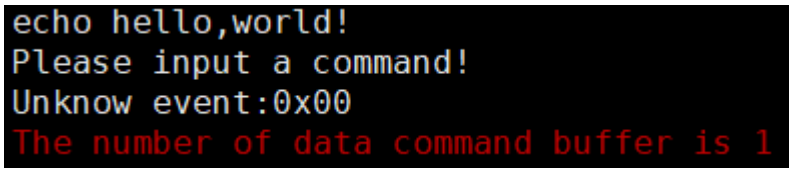


Figure 130. User input log (after a connection is established)

5. Receiving data
   When the Android app sends a string:

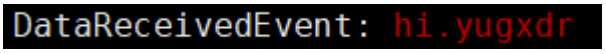


Figure 131. Received data log

5.6.3 BEE

This chapter introduces the features of the BEE Click Board and how to use it on LS1028ARDB.

5.6.3.1 BEE/ZigBEE

LS1028ARDB supports BEE click board, which can implement the MRF24J40MA 2.4 GHz IEEE 802.15.4 radio transceiver module from Microchip.

5.6.3.2 Introduction

The BEE Click Board features the MRF24J40MA 2.4 GHz IEEE 802.15.4 radio transceiver module from Microchip. The click is designed to run on 3.3 V power supply only. It communicates with the target controller over an SPI interface.

5.6.3.3 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

5.6.3.4 Hardware preparation

Use the following hardware items for the BEE Click Board demo setup:

- Two LS1028ARDB Boards
- Two BEE Click Boards

The figure below describes the hardware setup for the BEE Click Board.

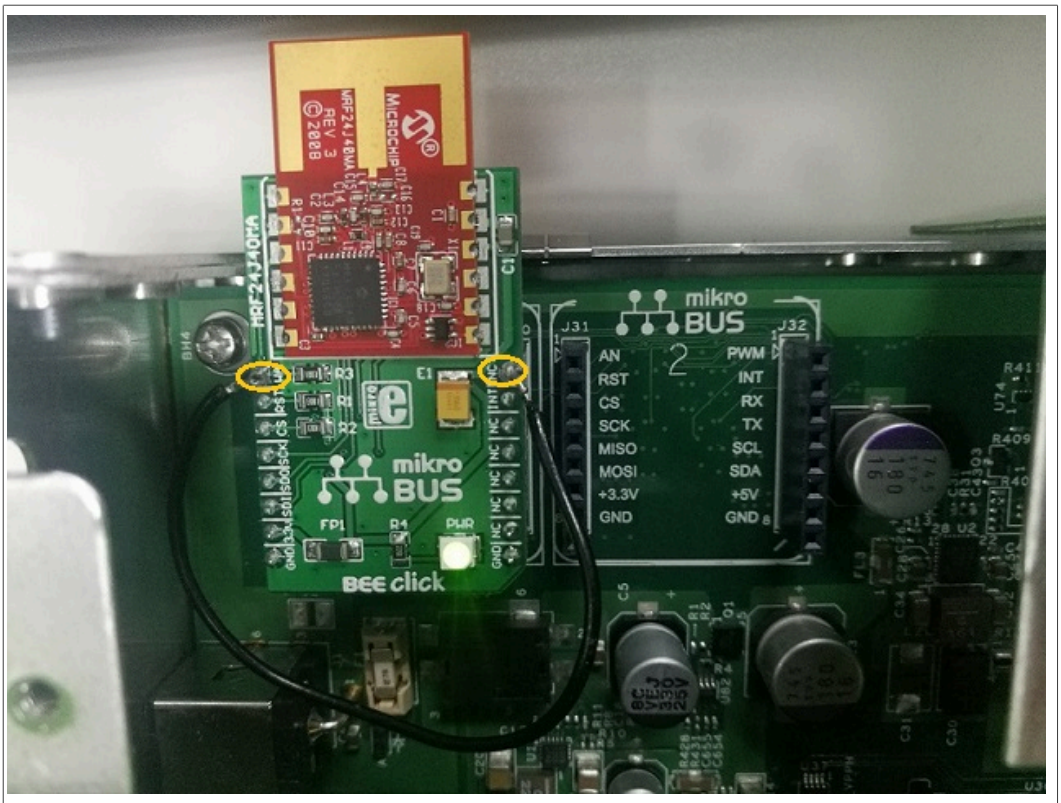


Figure 132. BEE Click Board hardware setup

Note: *The WA pin of BEE Click Board connects with the NC pin.*

5.6.3.5 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:

```
Device Drivers --->
  SPI support --->
    <*>   Freescale DSPI controller
    <*> User mode SPI device driver support
  -*- GPIO Support  --->
    [*]   /sys/class/gpio/... (sysfs interface)
          Memory mapped GPIO drivers  --->
            [*] MPC512x/MPC8xxx/QorIQ GPIO support
```

3. Use the `make` command to create the images.

5.6.3.6 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.
                    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.
```

5.7 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-ls1028ardb
$ 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
...
```

5.8 Wi-Fi on i.MX 8DXL EVK

5.8.1 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.

Table 73. Wi-Fi card

Feature	Description
Manufacturer	Embedded Artists
Part Number	EAR00387
Module	Murata LBEE5ZZ1XL (also known as 1XL)
Chipset	NXP 88W9098
Wi-Fi Standards	802.11 a/b/g/n/ac/ax 2x2 MU-MIMO, Wi-Fi 6

Table 73. Wi-Fi card*...continued*

Feature	Description
Frequency	2.4 GHz and 5 GHz
Wi-Fi Interface	PCIe 3.0 (in M.2 form factor)

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.

5.8.2 Hardware Setup

Install the Wi-Fi card '**1XL M.2 Module**' to the M.2 connector (J17) on i.MX 8DXL EVK.

5.8.3 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.

```
[   44.855282] mlan: loading out-of-tree module taints
 kernel.
[   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
[   44.952178] rx_work=1 cpu_num=2
```

```
[   44.955403] Attach mlan adapter operations.card_type is
 0x206.
[   44.967149] Request firmware: nxp/
pcieuart9098_combo_v1.bin
[   45.247953] FW download over, size 682784 bytes
[   45.510922] WLAN FW is active
[   45.513922] on_time is 79996054375
[   45.531982] VDLL image: len=160672
[   45.535888] fw_cap_info=0xc8fcffa3,
 dev_cap_mask=0xffffffff
[   45.541571] max_p2p_conn = 8, max_sta_conn = 64
[   45.549005] wlan: mlan0 set max_mtu 2000
[   45.569666] wlan: uap0 set max_mtu 2000
[   45.584434] wlan: wfd0 set max_mtu 2000
[   45.604604] wlan: version = PCIE9098--17.92.1.p116.1-
MM5X17344.p3-GPL-(FP92)
[   45.617407] wlan_pcie 0000:01:00.1: enabling device (0000
 -> 0002)
[   45.623935] Attach moal handle ops, card interface type:
 0x206
[   45.630063] PCIE9098: init module param from usr cfg
[   45.635290] card_type: PCIE9098, config block: 1
[   45.640039] cfg80211_wext=0xf
[   45.643271] max_vir_bss=1
[   45.645925] cal_data_cfg=none
[   45.650328] ps_mode = 1
[   45.653099] auto_ds = 1
[   45.655658] host_mlme=enable
[   45.662576] fw_name=nxp/pcieuart9098_combo_v1.bin
[   45.667565] rx_work=1 cpu_num=2
[   45.670767] Attach mlan adapter operations.card_type is
 0x206.
[   45.687250] Request firmware: nxp/
pcieuart9098_combo_v1.bin
[   45.694421] WLAN FW already running! Skip FW download
[   45.701756] WLAN FW is active
[   45.704867] on_time is 80092507000
[   45.710734] VDLL image: len=160672
[   45.714827] fw_cap_info=0x68fcffa3,
 dev_cap_mask=0xffffffff
[   45.720564] max_p2p_conn = 8, max_sta_conn = 64
[   45.727958] wlan: mmlan0 set max_mtu 2000
[   45.741123] wlan: muap0 set max_mtu 2000
[   45.747316] wlan: mwfd0 set max_mtu 2000
[   45.754666] wlan: version = PCIE9098--17.92.1.p116.1-
MM5X17344.p3-GPL-(FP92)
[   45.762585] wlan: Register to Bus Driver Done
[   45.767017] wlan: Driver loaded successfully
```

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)
        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
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)
        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
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)
        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
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)
        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
```

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)
        TSF: 90867505 usec (0d, 00:01:30)
        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 MPDU 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:

```
# 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.

```
# 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.

```
# 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.

```
# udhcpc -i mlan0
```

11. Check the IP address of AP and ping the AP to check connectivity.

```
# 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
……
```

5.9 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.

5.9.1 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).

Libmodbus is supported on all I.MX series and LayerScape series boards. It can be used to write a:

- client, the application reads/writes data from various devices.
- server, the application provides data to several clients.

The official website that contains the latest version of the documentation for libmodbus is www.libmodbus.org.

5.9.2 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, other boards define a dummy variable.*

5.9.3 Modbus-Simulator Usage

SYNOPSIS

{modbus_device_simulator|modbus_client_simulator} [OPTION]… {RTU-PARAMS|TCP-PARAMS}… {SERIALPORT|HOST}… [WRITE-DATA]…

5.9.3.1 Parameter description

The parameter of [Option] is the general parameter of TCP and RTU startup.

Table 74. Parameter description

Parameter option	Description
--debug,	show debug information.
-m,	connection Type TCP/RTU, optional parameter: {tcp|rtu}.
-a,	slave address.
-c,	read and write data number.

Table 74. Parameter description*...continued*

Parameter option	Description
-t,	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 75. Parameter description

Parameter option	Description
-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 76. Parameter description

Parameter option	Description
-p,	port.

5.9.3.2 EXAMPLES

Examples of TCP `modbus_device_simulator` and `modbus_client_simulator` startup are as follows:

1. Start `modbus_device_simulator` locally with port 1502.

```
# 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.

```
# 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`.

```
# 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.

```
# modbus_client_simulator --debug -m rtu -a 1 -t 0x06 -r 1 -b
 1 -p none /dev/ttymxc2
```

5.9.3.3 Commands for all features

The commands for all features of TCP are as follows:

1. Read the status of the LED light:

```
modbus_client_simulator --debug -m tcp -t 0x01 -r 0 -p 1502
 127.0.0.1
```

2. Change the status of the LED light:

```
# modbus_client_simulator --debug -m tcp -t 0x05 -r 0 -p 1502
 127.0.0.1 {0xFF00|0x0000}
```

3. Read the temperature of CPU

```
# 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:

```
modbus_client_simulator --debug -m tcp -t 0x01 -r 0 -p 1502
 127.0.0.1
```

2. Change the status of the LED light:

```
# 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:

```
 # 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.

```
# 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.

```
# modbus_client_simulator --debug -m rtu -a 6 -t 0x06 -r 1 -b
 3 -p none /dev/ttymxc2 1
```

5.9.4 Modbus-Simulator Test

Use two i.MX8MP boards to test the functions of the modbus-simulator.

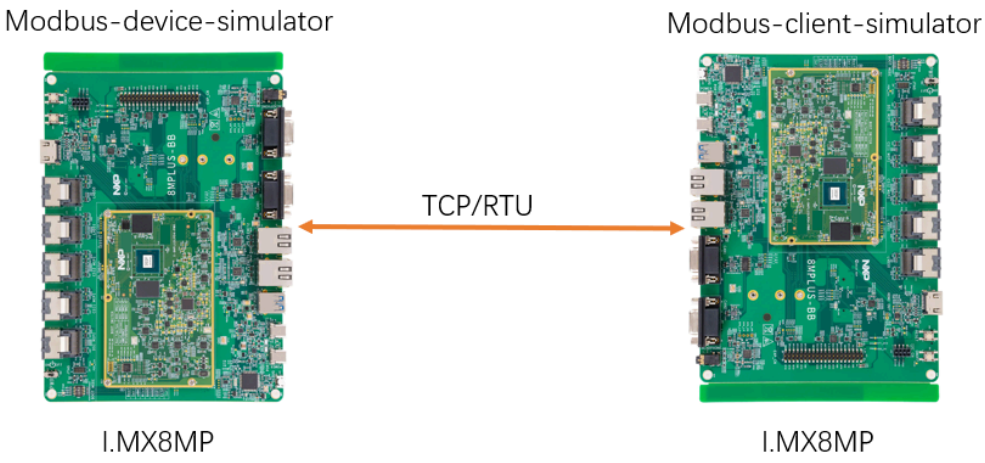


Figure 133. Testing the functions of the modbus-simulator

Note: *Learn about the startup parameters before starting.*

5.9.4.1 Testing TCP functions

Read the status of the LED light on the board1.

1. Enter board1 and start up `modbus_device_simulator`.

```
# modbus_device_simulator --debug -m tcp -p 1502 0.0.0.0
```

2. Enter board2 and start up `modbus_client_simulator`.

```
# modbus_client_simulator --debug -m tcp -t 0x01 -r 0 -p 1502
 10.193.21.104
```

```
root@imx8mp-lpddr4-evk:~# modbus_device_simulator --debug -m tcp -p 1502 0.0.0.0
Debug:
        Coils: 0x0001
        Digital inputs: 0x0001
        Holding registers: 0x0002
        Input registers: 0x0001
        Ip address: 0.0.0.0
        Port: 1502
        New connection from 10.193.21.105:14071 on socket 4
Waiting for an indication...
<00><01><00><00><00><06><01><01><00><00><00><01>
[00][01][00][00][00][04][01][01][01][01]
```

Figure 134. Debug log on I.mx8mp-lpddr4-evk board

5.9.4.2 Testing RTU functions

Preparation: Connect the serial ports on the two boards, as shown below.

Figure 135. Connection of two i.MX8MP boards

The pin connection information of the two boards should be as follows:

- GND <---> GND
- TXD <---> RXD
- RXD <---> TXD

1. Read the temperature of CPU on the board1.
2. Enter board1 and start up `modbus_device_simulator`.

```
# modbus_device_simulator --debug -m rtu -a 1 -b 115200 -p
 none /dev/ttymxc2
```

3. Enter board2 and start up `modbus_client_simulator`.

```
# 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.*

5.10 UART 9-bit Multidrop mode (RS-485) support

5.10.1 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.

5.10.2 Building and running the demo

5.10.2.1 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 bellow 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/
```

5.10.2.2 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.

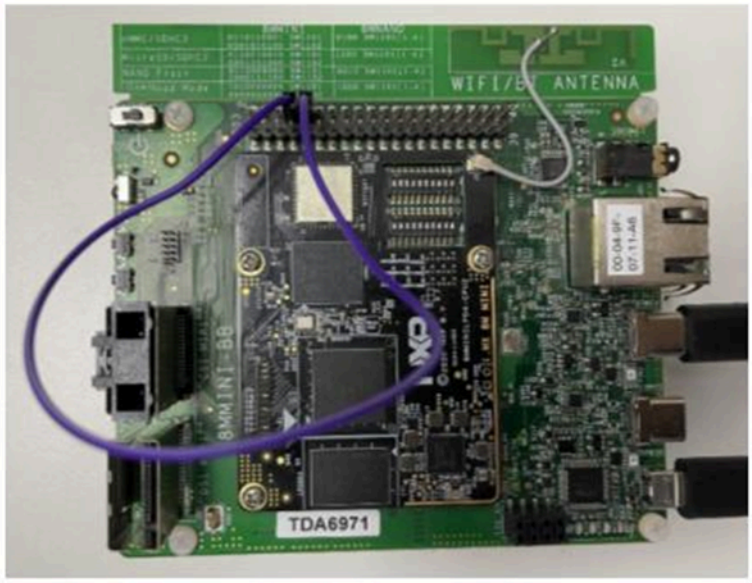


Figure 136. i.MX 8M Mini LPDDR4 EVK UART Connection

5.10.2.3 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

3. Program the full Yocto SD card image in Linux host PC:

Note: *The SD card capacity is required at least 4 GB.*

```
$ bzip2 -d -c nxp-image-real-time-edge-imx8mm-lpddr4-
evk.wic.bz2 | pv | sudo dd of=/dev/sdc bs=1M && sync
```

5.10.2.4 Running the demo

Power up the board and startup M-core firmware under U-Boot by using the following commands:

Use `9bit_iuart_interrupt_transfer` or `9bit_iuart_polling` for different examples to replace “DEMO_NAME” in the following commands.

If you choose to run the binary in DRAM:

```
=> ext4load mmc 1:2 0x80000000 /examples/DEMO_NAME/ddr_release/
DEMO_NAME.bin
=> dcache flush
=> bootaux 0x80000000
```

If you choose to run the binary in TCM:

```
=> ext4load mmc 1:2 0x48000000 /examples/ DEMO_NAME /release/
DEMO_NAME.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:

```
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!
```

6 Revision history

The table below summarizes revisions to this document.

Table 77. Document revision history

Date	Document version	Topic cross-reference	Change description
16/12/2022	2.4	-	Added support for I.MX 8DXL and i.MX93 EVK boards throughout the document
		Configuring Flow Meter policy on stream	Added the section
		Section 4.1.3.4.1	Added the section for preemptive verify
		Section 3.2.4.3	Added the Math library support
		Section 5.8	Added the section
		Section 5.1.3.3.5	Added the section
28/07/2022	2.3	What's new in Real-time Edge software v2.3	Updated the section
		Heterogeneous AMP Architecture	Added the section
		Section 4.2	Updated the section
		-	Other updates throughout the document
29/03/2022	2.2	What's new in Real-time Edge software v2.2	Added the section. Updated for Real-time Edge Software Rev 2.2. Support for TLS protocol removed for NETCONF feature.
15/12/2021	2.1	Section 2.1.4	Added the section. Updated for Real-time Edge Software Rev 2.1
		Section 2.3	Updated the section
29/07/2021	2.0	-	• 'Real-time Edge Software' introduced instead of 'Open Industrial Linux', with real-time feature support • Rearranged the document structure to include the Chapters: Real-time System, Real-time Networking, and Protocols
		-	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*
		Section 2.1	Added the section.
		-	Integrated BareMetal framework in the document
26/04/2021	1.11	Section 2.1	Added the section that describes the new features of each release
		-	Updated the section 'Getting Open IL'
		-	Deleted references to Edgescale, OP-TEE, OTA throughout the document and other minor updates

Table 77. Document revision history *...continued*

Date	Document version	Topic cross-reference	Change description
22/12/2020	1.10	Section 4.2	Added the chapter and related contents.
		-	Added the 'Camera' section and related details.
		-	Added the Host setup for i.MX 8M Plus EVK board details
15/09/2020	1.9	Section 4.3	Added the section
		Section 5.5.1	Added the chapter and related description
		Section 5.5.2	Added the chapter and related description
		Section 5.1.3.3	Made it a part of the chapter "EtherCAT"
29/05/2020	1.8	Section 3.1	Added the section in Section 5
		-	Updated this section Interface naming in Linux for LS1028ARDB.
		-	Updated the section Host system requirements for Open IL
		-	Updated the section Running Selinux demo
20/02/20	1.7.1	Section 5.4.3.8	Updated this section
17/01/20	1.7	Section 5.1.3.3	Added the chapter (nxp servo)
		Section 4.3	Added the chapter
		Getting Open IL	Updated the section
		Section 5.4	Other updates
31/08/19	1.6	Section 4.1.4	• Information related to pcpmap command removed from the section Section 4.1.4.1 and Section 4.1.4.2 • Port names "eno/swp0" changed to "swp0" for few tsntool commands. • Note added in section Section "Stream identification" for usage of `nulltagged` and `streamhandle` parameters. • Added the section Section 4.1.4.2.8. • Other minor updates
		-	Updated the table "Host system mandatory packages". Added `autogen autoconf libtool` and `pkg-config` packages
		Section 5.6.3	Added this Chapter
		-	Updated Section 5.4
		Section 5.4	• Added the section Section 5.4.3.1 and other updates.
01/05/2019	1.5	-	Added the section to describe interface naming for U-Boot and Linux for LS1028ARDB.
		Section 4.1.4	Updated this section in the Chapter Section 4.1
		Section 5.6.2	Added the chapter
		-	Added the chapter 'EdgeScale Client'

Table 77. Document revision history*...continued*

Date	Document version	Topic cross-reference	Change description
		-	Updated the OpenIL version and Git tag in the section 'Getting Open IL'.
01/02/2019	1.4	Section 1.3	Added support for LS1028ARDB (64-bit and Ubuntu). Updated various sections accordingly
		-	Updated the OpenIL version and Git tag in the section 'Getting Open IL'.
		-	Added this Section for LS1028ARDB support
		Section 4.1	Reorganized this Chapter and added separate Section for Section 4.1.4
		Section 5.6.1	Added the Chapter.
		Section 5.2	Minor updates in this Chapter. Also added the section, Section 5.2.3.2 and Section 5.2.3.6.
		-	Added the chapter QT
15/10/2018	1.3.1	-	Updated the OpenIL version and Git tag in the section 'Getting Open IL'
31/08/2018	1.3	EtherCAT	Added the chapter
		FlexCAN	Added the chapter
		i.MX6QSabreSD support.	Added the section in chapter 'NXP OpenIL platforms'. Updated other sections for i.MX6Q Sabre support
		Getting Open IL	Updated the section
		Selinux demo	Added the section for enabling SELinux and updated Basic setup. Updates in other sections.
31/05/2018	1.2	-	Updated the Section, "Hardware requirements" for RTnet
		-	Updated the Section, "Software requirements" for RTnet
18/04/2018	1.1.1	-	Added the Section, "RTnet"
		-	Added a note for LS1043A switch setting
30/03/2018	1.1	-	Added support for industrial IoT BareMetal framework in this section
		-	Added a note for steps to be performed before booting up the board
		Section 1.4	Added the section
22/12/2017	1.0	Section 5.3	Added the chapter
		Section 4.1	Chapters for "1-board TSN demo" and "3-board TSN demo" replaced by a single chapter, "TSN demo"
		Section 5	• Updated the section, 'Industrial Features' • -IEEE 1588 -'sja1105-ptp' support removed
25/08/2017	0.3	-	Set up the OpenIL websitehttp://www.openil.org/.

Table 77. Document revision history *...continued*

Date	Document version	Topic cross-reference	Change description
		-	OTA - Xenomai Cobalt 64-bit and SJA1105 support added
		Section 4.1	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, LS1046ARDB, and LS1012ARDB added
		Section 4	Ubuntu Userland support for 64-bit LS1043ARDB and 64-bit LS1046ARDB added
26/05/2017	0.2	-	Initial public release

7 Legal information

7.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

7.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

i.MX — is a trademark of NXP B.V.

Contents

For more information, please visit: http://www.nxp.com

Date of release: 16 December 2022
Document identifier: REALTIMEEDGEUG