REALTIMEEDGEUG

Real-time Edge Software User Guide

Rev. 3.3 — 15 December 2025 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.MX 93 EVK, i.MX RT1180 EVK, i.MX 93 9x9 QSB, i.MX 93 14x14 EVK, i.MX 91 9x9 LPDDR4 QSB, i.MX 91 11x11 LPDDR4 EVK, i.MX 95 15x15 LPDDR4 EVK, i.MX 95 19x19 EVK, i.MX 943 19x19 LPDDR4 EVK, i.MX 943 19x19 LPDDR5 EVK, LX2160ARDB Rev2, 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 System, Real-time Networking, Heterogeneous Multicore Framework, Heterogeneous Multi-SoC Framework, and Protocols.

1 Introduction

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

- The Real-time System component includes Preempt-RT Linux, Native RTOS on Cortex-A, Jailhouse, and U-Boot-based BareMetal framework. It also includes RTOS, BareMetal on Cortex-M, and different combinations of these systems.
- Heterogeneous Multicore Framework provides a general software framework to support Heterogeneous AMP (asymmetric multiprocessing). It enables AMP to be interconnected and provides a unified resource management and life-cycle management.
- Heterogeneous Multi-SoC Framework enables the usage of a combination of MPU and MCU. It extends the hardware components for both the MCU and MPU.
- Real-time Networking includes time sensitive network (TSN) technology, TSN standards, management, configuration, and applications. This component also supports the networking and redundancy features.
- The Protocols component includes support for industry standard protocols, such as EtherCAT, CoE, FlexCan, OPC-UA, and others.

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

1.1 Feature support matrix

Table 1 shows the features that are supported in this release.

Table 1. 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	i.MX 93 9x9 LPDDR4 QSB	i.MX 93 14x14 LPDDR4x EVK	i.MX 943 19x19 LPDDR4/5 EVK	i.MX 95 19x19 LPDDR5 EVK	i.MX 95 15x15 LPDDR4 EVK	i.MX 91 11x11 LPDDR4 EVK	i.MX 91 9x9 LPDDR4 QSB	i.MX RT1180 EVK	i.MX 8MP LPDDR4 FRDM	i.MX 91 11x11 LPDDR4 FRDM	i.MX 91 11x11 LPDDR FRDM IMX91s	i.MX93 11x11 LPDDR4X FRDM	LS1028A RDB	LS1043A RDB	LS1046A RDB	LX2160A RDB
Boot mode	SD			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y
	eMMC					Y	Y	Y	Y	Y	Y	Y	Y	Y	Y			Y		Y	Y		Y	
Real-time System	Preempt-RT Linux			Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y	Y
	BareMetal	ICC				Y	Y	Y	Y												Y	Y	Y	Y
		PCIe																			Y	Y	Y	
		Ethernet				Y	Y	Y													Y	Y	Y	
		GPIO				Y	Y																Y	
		IPI				Y	Y	Y	Y												Y	Y	Y	Y
		UART				Y	Y	Y	Y												Y	Y	Y	Y
		USB																				Y	Y	
		SAI																			Y			
		CAN																						
		I2C																			Y	Y	Y	
		QSPI																					Y	
		IFC																					Y	
		Flextimer																					Y	
		Linux (communication with BareMetal)	ICC			Y	Y	Y	Y												Y	Y	Y	Y
			IPI			Y	Y	Y	Y												Y	Y	Y	Y
		Single HW Interrupt to multiple cores																					Y	
		Newlib Math library				Y	Y	Y	Y												Y	Y	Y	Y
		All Cortex-A cores running under Baremetal																			Y		Y	
	Native RTOS on Cortex-A	FreeRTOS				Y	Y	Y		Y		Y	Y	Y	Y									
		Zephyr				Y	Y	Y		Y		Y	Y	Y	Y									
	Jailhouse					Y	Y	Y	Y			Y	Y								Y	Y	Y	
	Harpoon RTOS	FreeRTOS				Y	Y	Y				Y	Y											
		Zephyr				Y	Y	Y				Y	Y											
Heterogeneous Multicore Framework	Flexible Real-time System	Flexible Real-time System	Free RTOS			Y	Y	Y		Y		Y	Y											
			Zephyr			Y	Y	Y		Y		Y	Y											
		RAM Console	Free RTOS			Y	Y	Y		Y		Y	Y											
			Zephyr			Y	Y	Y		Y		Y	Y											
		SMP RTOS	Zephyr				Y	Y		Y														

Table 1. 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	i.MX 93 9x9 LPDDR4 QSB	i.MX 93 14x14 LPDDR4x EVK	i.MX 943 19x19 LPDDR4/5 EVK	i.MX 95 19x19 LPDDR5 EVK	i.MX 95 15x15 LPDDR4 EVK	i.MX 91 11x11 LPDDR4 EVK	i.MX 91 9x9 LPDDR4 QSB	i.MX RT1180 EVK	i.MX 8MP LPDDR4 FRDM	i.MX 91 11x11 LPDDR4 FRDM	i.MX 91 11x11 LPDDR FRDM IMX91s	i.MX93 11x11 LPDDR4X FRDM	LS1028A RDB	LS1043A RDB	LS1046A RDB	LX2160A RDB
		Networking stack on A-Core RTOS	Free RTOS			Y	Y	Y		Y														
	Unified Life Cycle Management	U-Boot booting Native RTOS A-Core Image	Free RTOS			Y	Y	Y		Y		Y	Y	Y	Y									
			Zephyr			Y	Y	Y		Y		Y	Y	Y	Y									
		Linux booting Native RTOS A-Core Image	Free RTOS			Y	Y	Y		Y		Y	Y											
			Zephyr			Y	Y	Y		Y		Y	Y											
		U-Boot booting Native RTOS M-Core Image	Free RTOS			Y	Y	Y	Y	Y														
		Linux booting Native RTOS M-Core Image	Free RTOS			Y	Y	Y	Y	Y														
	RPMSG	RPMSG between Linux and M-Core RTOS	Free RTOS			Y	Y	Y	Y	Y														
		RPMSG between Linux and A-Core RTOS	Free RTOS			Y	Y	Y		Y		Y	Y											
			Zephyr				Y	Y		Y														
		RPMSG between 2 A-Core RTOS	Free RTOS				Y	Y		Y														
		RPMSG between Linux and M-core RTOS with enhanced 8 MB buffer	Free RTOS			Y																		
		RPMSG Perfor-mance Evaluation	Free RTOS				Y																	
		UART Sharing based on RPMSG	Free RTOS			Y		Y	Y	Y														
	Hetero-geneous Multicore VirtIO	Hetero-geneous Multicore VirtIO Perfor-mance Evaluation	Free RTOS			Y	Y																	

Table 1. 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	i.MX 93 9x9 LPDDR4 QSB	i.MX 93 14x14 LPDDR4x EVK	i.MX 943 19x19 LPDDR4/5 EVK	i.MX 95 19x19 LPDDR5 EVK	i.MX 95 15x15 LPDDR4 EVK	i.MX 91 11x11 LPDDR4 EVK	i.MX 91 9x9 LPDDR4 QSB	i.MX RT1180 EVK	i.MX 8MP LPDDR4 FRDM	i.MX 91 11x11 LPDDR4 FRDM	i.MX 91 11x11 LPDDR FRDM IMX91s	i.MX93 11x11 LPDDR4X FRDM	LS1028A RDB	LS1043A RDB	LS1046A RDB	LX2160A RDB
		Hetero-geneous Multicore VirtIO Network Sharing	Free RTOS			Y	Y	Y		Y														
	Industrial Applications	SOEM running on A-core or M-core	Free RTOS			Y	Y	Y																
Heteroge-neous Multi-SoC Framework	NETC TSN Switch	DSA single port mode					Y	Y			Y					Y								
		DSA bridge mode					Y	Y			Y					Y								
		MTU configuration					Y	Y			Y					Y								
		VLAN configuration					Y	Y			Y					Y								
		FDB configuration					Y	Y			Y					Y								
		Port statistics					Y	Y			Y					Y								
		PTP stack					Y	Y			Y					Y								
		802.1CB					Y	Y			Y					Y								
		Qci					Y	Y			Y					Y								
		Qbv					Y	Y			Y					Y								
		Qbu					Y	Y			Y					Y								
		HSR					Y	Y			Y					Y								
	Virtual Switch						Y	Y			Y					Y								
Real-time Networking	TSN Standards	Qbv			Y		Y	Y	Y	Y	Y										Y			
		Qbu			Y		Y	Y	Y	Y	Y										Y			
		Qci									Y										Y			
		Qav			Y	Y	Y	Y	Y	Y	Y										Y			
		802.1AS			Y	Y	Y	Y	Y	Y	Y										Y	Y	Y	Y
		802.1CB																			Y			
		VCAP chain mode																			Y			
		802.1 Q-in-Q																			Y			
	TSN Config-urations	Linux tc command			Y		Y	Y	Y	Y											Y			
		TSN tool																			Y			
		NETCONF /YANG	Qbv		Y		Y	Y	Y	Y	Y										Y			
			Qbu		Y		Y	Y	Y	Y	Y										Y			
			Qci																		Y			
			IP		Y		Y	Y	Y	Y	Y										Y			
			MAC		Y		Y	Y	Y	Y	Y										Y			
			VLAN config		Y		Y	Y	Y	Y	Y										Y			
			PTP		Y		Y	Y	Y	Y	Y													
			LLDP		Y		Y	Y	Y	Y	Y													
		Web-based configuration	Qbv		Y		Y	Y	Y												Y			
			Qbu		Y		Y	Y	Y												Y			

Table 1. 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	i.MX 93 9x9 LPDDR4 QSB	i.MX 93 14x14 LPDDR4x EVK	i.MX 943 19x19 LPDDR4/5 EVK	i.MX 95 19x19 LPDDR5 EVK	i.MX 95 15x15 LPDDR4 EVK	i.MX 91 11x11 LPDDR4 EVK	i.MX 91 9x9 LPDDR4 QSB	i.MX RT1180 EVK	i.MX 8MP LPDDR4 FRDM	i.MX 91 11x11 LPDDR4 FRDM	i.MX 91 11x11 LPDDR FRDM IMX91s	i.MX93 11x11 LPDDR4X FRDM	LS1028A RDB	LS1043A RDB	LS1046A RDB	LX2160A RDB
			Qci																		Y			
		Dynamic topology discovery			Y		Y	Y	Y												Y			
		Dynamic TSN configu-ration	Qci																		Y			
			CQF																		Y			
			Qbv		Y		Y	Y	Y												Y			
	GenAVB / TSN Stack	gPTP 802.1AS-2020 with multi-domain support			Y	Y	Y	Y	Y	Y	Y	Y									Y	Y	Y	Y
		SRP IEEE 802.1Q-2022		Y	Y	Y	Y	Y	Y	Y		Y									Y	Y	Y	Y
		AVB Standards	AVTP IEEE 1722-2016	Y	Y	Y	Y	Y		Y														
			ATDECC 1722.1-2021	Y	Y	Y	Y	Y		Y														
			Milan v1.2	Y	Y	Y	Y	Y		Y														
			MAAP IEEE 1722-2016	Y	Y	Y	Y	Y		Y														
		AVB Endpoint		Y	Y	Y	Y	Y		Y														
		AVB Bridge			Y [1]					Y [1]											Y			
		AVB Hybrid								Y [1]														
		TSN Endpoint			Y		Y	Y	Y	Y	Y	Y												
		TSN Bridge																			Y			
		Media clock recovery		Y	Y	Y [2]	Y	Y [2]		Y [2]														
Industrial Protocol	EtherCAT MainDevice	IGH EtherCAT Main Device stack		Y	Y	Y	Y	Y			Y	Y	Y								Y	Y	Y	Y
		IGH native Ethernet device driver			Y	Y	Y	Y			Y	Y	Y								Y	Y	Y	
		SOEM				Y	Y	Y			Y													
		CodeSYS EtherCAT MainDevice stack		Y		Y	Y	Y			Y	Y	Y											
	FlexCAN										Y					Y								
	CANopen										Y					Y								
	OPC UA	open62541		Y	Y	Y	Y	Y	Y	Y		Y									Y	Y	Y	Y
		OPC UA PubSub over TSN			Y		Y	Y	Y	Y		Y	Y								Y			
		OPC UA Frame Summation									Y													
	BEE (Mikroe Click board)																				Y			
	Bluetooth Low Energy (Mikroe Click board)																				Y			
	NFC (Mikroe Click board)																				Y			
	Modbus	Modbus-RTU		Y	Y	Y	Y	Y	Y	Y	Y		Y											

Table 1. 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	i.MX 93 9x9 LPDDR4 QSB	i.MX 93 14x14 LPDDR4x EVK	i.MX 943 19x19 LPDDR4/5 EVK	i.MX 95 19x19 LPDDR5 EVK	i.MX 95 15x15 LPDDR4 EVK	i.MX 91 11x11 LPDDR4 EVK	i.MX 91 9x9 LPDDR4 QSB	i.MX RT1180 EVK	i.MX 8MP LPDDR4 FRDM	i.MX 91 11x11 LPDDR4 FRDM	i.MX 91 11x11 LPDDR FRDM IMX91s	i.MX93 11x11 LPDDR4X FRDM	LS1028A RDB	LS1043A RDB	LS1046A RDB	LX2160A RDB
		Modbus-TCP	Y	Y	Y	Y	Y	Y	Y	Y		Y								Y	Y	Y	Y
	Digital Encoder	BiSS								Y													
		EnDat2.2								Y													
		EnDat3.0								Y													
		HIPERFACE DSL								Y													
		Nikon A-Format								Y					Y								
		Tamagawa T-Format								Y					Y								

[1] Using a SJA1105Q Ethernet Interface Evaluation Board (EVB).
[2] Media clock recovery is implemented through a software-based sampling.

1.2 Supported NXP platforms

The Table 2 lists the NXP hardware SoCs and boards that support the Real-time Edge software.

Table 2. Supported NXP platforms

Platform	Architecture	Boot
i.MX 6ULL EVK	Arm v7	SD
i.MX 8DXL LPDDR4 EVK	Arm v8	SD, eMMC
i.MX 8M Mini LPDDR4 EVK	Arm v8	SD, eMMC
i.MX 8M Plus LPDDR4 EVK	Arm v8	SD, eMMC
i.MX 8M Plus LPDDR4 FRDM	Arm v8	SD
i.MX 93 EVK	Arm v8	SD, eMMC
i.MX 93 9x9 QSB	Arm v8	SD, eMMC
i.MX 93 11x11 LPDDR4x FRDM	Arm v8	SD, eMMC
i.MX 93 14x14 EVK	Arm v8	SD, eMMC
i.MX 95 19x19 LPDDR5 EVK	Arm v8	SD, eMMC
i.MX 95 15x15 LPDDR4X EVK	Arm v8	SD, eMMC
i.MX 91 11x11 LPDDR4X EVK	Arm v8	SD, eMMC
i.MX 91 11x11 LPDDR4 FRDM	Arm v8	SD, eMMC
i.MX 91 11x11 LPDDR4X FRDM IMX91S	Arm v8	SD
i.MX 91 9x9 LPDDR4 QSB	Arm v8	SD, eMMC
i.MX 943 19x19 LPDDR5 EVK	Arm v8	SD, eMMC
i.MX 943 19x19 LPDDR4 EVK	Arm v8	SD, eMMC
i.MX 943 15x15 LPDDR4 EVK	Arm v8	SD, eMMC
LS1028ARDB	Arm v8	SD, eMMC
LS1043ARDB	Arm v8	SD
LS1046ARDB	Arm v8	SD, eMMC
LX2160ARDB Rev 2	Arm v8	SD

1.2.1 Switch settings

The Table 3 lists and describes the switch configuration for the platforms supported by Real-time Edge software.

Table 3. 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, eMMC	• SD: SW1[1:4] = 0b'1100 • eMMC: SW1[1:4] = 0b'0100
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 8M Plus LPDDR4 FRDM	MicroSD / SDHC2	SW4[1:4] = 0b’0011
i.MX 93 EVK	MicroSD / uSDHC2	SW1301[1:4] = 0b'0100
i.MX 93 9x9 QSB	MicroSD / uSDHC2	SW601[1:4] = 0b'0011
i.MX 93 11x11 FRDM	SD, eMMC	• SD: SW1[1:4] = 0b1100 • eMMC: SW1[1:4] = 0b0100
i.MX 93 14x14 EVK	SD, eMMC	• SD: SW5[1:4] = 0b'0100 • eMMC: SW5[1:4] = 0b'0000
i.MX 95 19x19 LPDDR5 EVK	SD, eMMC	• SD: SW7[1:4] = 0b'1011 • eMMC: SW7[1:4] = 0b'1010
i.MX 95 15x15 LPDDR4X EVK	SD, eMMC	• SD: SW7[1:4] = 0b1011 • eMMC: SW7[1:4] = 0b1010
i.MX 91 11x11 LPDDR4X EVK	SD, eMMC	• SD: SW1301[1:4] = 0b0100 • eMMC: SW1301[1:4] = 0b0000
i.MX 91 11x11 LPDDR4 FRDM	SD, eMMC	• SD: SW1[1:4] = 0b1100 • eMMC: SW1[1:4] = 0b0100
i.MX 91 11x11 LPDDR4 FRDM-IMX91S	SD	• SD: SW1[1:4] = 0b1100
i.MX 91 9x9 LPDDR4 QSB	SD, eMMC	• SD: SW3[1:4] = 0b0011 • eMMC: SW3[1:4] = 0b0010
i.MX 943 19x19 LPDDR4 EVK	SD, eMMC	• SD: SW3[1:4] = 0b1100 • eMMC: SW3[1:4] = 0b0010
i.MX 943 19x19 LPDDR5 EVK	SD, eMMC	• SD: SW3[1:4] = 0b1100 • eMMC: SW3[1:4] = 0b0010
i.MX 943 15x15 LPDDR4 EVK	SD, eMMC	• SD: SW3[1:4] = 0b1100 • eMMC: SW3[1:4] = 0b0010
LS1028ARDB	SD, eMMC	• SD: SW2[1:8] = 0b’10001000 • eMMC: SW2[1:8] = 0b’10011000
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)

Table 3. Switch setting for various NXP platforms*...continued*

Platform	Boot source	Switch setting
LS1046ARDB	SD, eMMC	SW5[1:8] + SW4[1] = 0b'00100000_0
		UART1 output select • SW4[4] = 0b’0: RJ45 • SW4[4] = 0b’1: CMSIS-DAP (MicroUSB)
LX2160ARDB Rev2	SD	SW1[1:8] = 0b'10001000

1.2.2 Flashing prebuilt images

Prebuilt images for hardware platforms supported by Real-time Edge software can be downloaded from the below URL:

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

Download the image required and extract it by using the commands below: (The code below shows the commands used for i.MX 93 EVK as an example.)

```
$ unzip Real-time_Edge_v3.3_IMX93EVK.zip
$ cd Real-time_Edge_v3.3_IMX93EVK/real-time-edge/
$ ls
Image-imx93evk.bin                        imx93-11x11-evk-igh.dtb
imx93-11x11-evk-avb.dtb                   imx93-11x11-evk-inmate.dtb
imx93-11x11-evk-dpdk.dtb                  imx93-11x11-evk-multicore-rtos.dtb
imx93-11x11-evk-dsa.dtb                   imx93-11x11-evk-root.dtb
imx93-11x11-evk-dsa-enetc.dtb             imx93-11x11-evk-rpmsg.dtb
imx93-11x11-evk.dtb                       imx93-11x11-evk-uart-sharing-cm33.dtb
imx93-11x11-evk-ecat.dtb                  imx93-11x11-evk-virtio-net-ca55.dtb
imx93-11x11-evk-ecat-userspace.dtb        imx93-11x11-evk-virtio-net-cm33.dtb
imx93-11x11-evk-harpoon-avb.dtb           nxp-image-real-time-edge-
imx93evk.rootfs.manifest
imx93-11x11-evk-harpoon.dtb               nxp-image-real-time-edge-
imx93evk.rootfs.tar.zst
imx93-11x11-evk-harpoon-industrial.dtb  nxp-image-real-time-edge-
imx93evk.rootfs.wic.zst
imx93-11x11-evk-harpoon-virtio-net.dtb
```

Extract the compressed wic image:

```
$ zstd -d nxp-image-real-time-edge-imx93evk.rootfs.wic.zst
```

Insert the SD card. A device node “`sdx`” (for example: `sdc`) is created in directory “`/dev/`” with USB reader. Flash the file “`nxp-image-real-time-edge-imx93evk.rootfs.wic`” to SD card:

```
$ sudo dd if=./nxp-image-real-time-edge-imx93evk.rootfs.wic of=/dev/sdc bs=1M
 conv=fsync
```

After flashing this image to an SD card, insert this SD card into i.MX 93 EVK board. Then, connect the DBG port and open it. Then, power on the i.MX 93 EVK board. The message below is displayed:

```
U-Boot SPL 2025.04-g7d45dffba048 (Jul 15 2025 - 10:04:29 +0000)
PMIC: PCA9451A
PMIC: Over Drive Voltage Mode
DDR: 3733MTS
M33 prepare ok
Normal Boot
Trying to boot from BOOTROM
Boot Stage: Primary boot
image offset 0x8000, pagesize 0x200, ivt offset 0x0
Load image from 0x57800 by ROM_API
NOTICE:  TRDC init done
NOTICE:  BL31: v2.12.0(release):lf-6.12.20-2.0.0-dirty
NOTICE:  BL31: Built : 08:15:07, May  9 2025
```

```
U-Boot 2025.04-g7d45dffba048 (Jul 15 2025 - 10:04:29 +0000)

Reset Status: POR

CPU:   NXP i.MX93(52) Rev1.1 A55 at 1700 MHz
CPU:   Industrial temperature grade  (-40C to 105C) at 36C
Model: NXP i.MX93 11X11 EVK board
DRAM:  2 GiB
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C2 0x52]
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C2 0x51]
TCPC:  Vendor ID [0x1fc9], Product ID [0x5110], Addr [I2C2 0x50]
Core:  249 devices, 37 uclasses, devicetree: separate
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... Reading from MMC(1)... *** Warning - bad CRC,
 using default environment

[*]-Video Link 0adv7535_mipi2hdmi hdmi@3d: Can't find cec device id=0x3c
fail to probe panel device hdmi@3d
fail to get display timings
probe video device failed, ret -19

        [0] lcd-controller@4ae30000, video
        [1] dsi@4ae10000, video_bridge
        [2] hdmi@3d, panel
adv7535_mipi2hdmi hdmi@3d: Can't find cec device id=0x3c
fail to probe panel device hdmi@3d
fail to get display timings
probe video device failed, ret -19
In:    serial
Out:   serial
Err:   serial

BuildInfo:
  - ELE firmware version 2.0.2-85b63cb9

switch to partitions #0, OK
mmc1 is current device
UID: 7e9483913bca4ffba6a990c8d86d46ba
flash target is MMC:1
Net:   eth0: ethernet@42890000, eth1: ethernet@428a0000 [PRIME]
Fastboot: Normal
......
```

1.3 Real-time Edge Software Yocto Project

For using the 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.

1.4 Related documentation

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

- ***Real-time Edge Release Notes*** (document RN00161*)* provides release information.
- ***Real-time Edge User Guide*** (document REALTIMEEDGEUG) provides description of features and usage of the software.

- ***Real-time Edge Yocto Project User Guide*** *(*document RTEDGEYOCTOUG*)* provides information for using the Yocto build environment.
- ***GenAVB/TSN Stack Evaluation User Guide*** *(*document GENAVBTSNUG*)* provides information on how to set up Audio Video Bridging evaluation experiments of the GenAVB/TSN Stack on NXP platforms.
- ***Harpoon User's Guide*** *(*HRPNUG*)* provides information to build Harpoon Yocto images.
- ***i.MX6ULL EVK GenAVB/TSN Rework Application Note*** (document AN13678)
- ***NXP MPU Cortex-A Core Zephyr User Guide*** (document UG10199)
- For details about the graphics feature available in i.MX 8M Plus and i.MX 8M Mini boards, refer to the i.MX Graphics User's Guide (*https://www.nxp.com/docs/en/user-guide/UG10159.pdf*).

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

- *i.MX 6ULL EVK Quick Start Guide*
- *i.MX 8M Mini LPDDR4 EVK Quick Start Guide*
- *i.MX 8M Plus LPDDR4 EVK Quick Start Guide*
- *i.MX 8XLite EVK Quick Start Guide*
- *LS1028ARDB Quick Start Guide*
- *LS1043ARDB Getting Started Guide*
- *LS1046ARDB Getting Started Guide*
- *LX2160A/LX2160A-Rev2 RDB Quick Start Guide*
- *i.MX 93 EVK Quick Start Guide*
- *i.MX 91 EVK Quick Start Guide*
- *i.MX RT1180 Getting Started Guide*

Note: *For access to these documents, contact your local NXP Field Application Engineer (FAE) or your local NXP support executive.*

2 Real-time system

This section provides an overview of Real-time systems, features, support matrix, and deployment methods using Baremetal and RTOS.

Real-time System addresses several real-time requirements on multicore platforms. Schedule latency, inter-core communication, hardware resource sharing, unified life-cycle management, and unified building and deployment mechanism are some of these requirements that a real-time system addresses.

To support different schedule latency requirements, the Real-time System provides the following features:

- Preempt-RT Linux
- Native RTOS on Cortex-A core and Cortex-M core
- RTOS on Cortex-A core with Jailhouse
- BareMetal framework

To meet the requirements for different use cases, Real-time System provides a flexible combination of different cores running Preempt-RT Linux and RTOS/BareMetal.

The heterogeneous multicore framework feature of Real-time Edge software facilitates intercore communication and hardware-resource sharing between the different CPU cores and operating systems. This feature covers high-performance communication and real-time requirements. Unified CPU core life cycle management feature provides a unified mechanism to bootstrap the Cortex-A core and Cortex-M core on the heterogeneous MPU system. Refer to chapter Section 3 for details.

The unified software building and deploying mechanism provides easy building and deployment for the software components running on the Cortex-A core and Cortex-M core. These components and features of this mechanism are depicted in Figure 1.

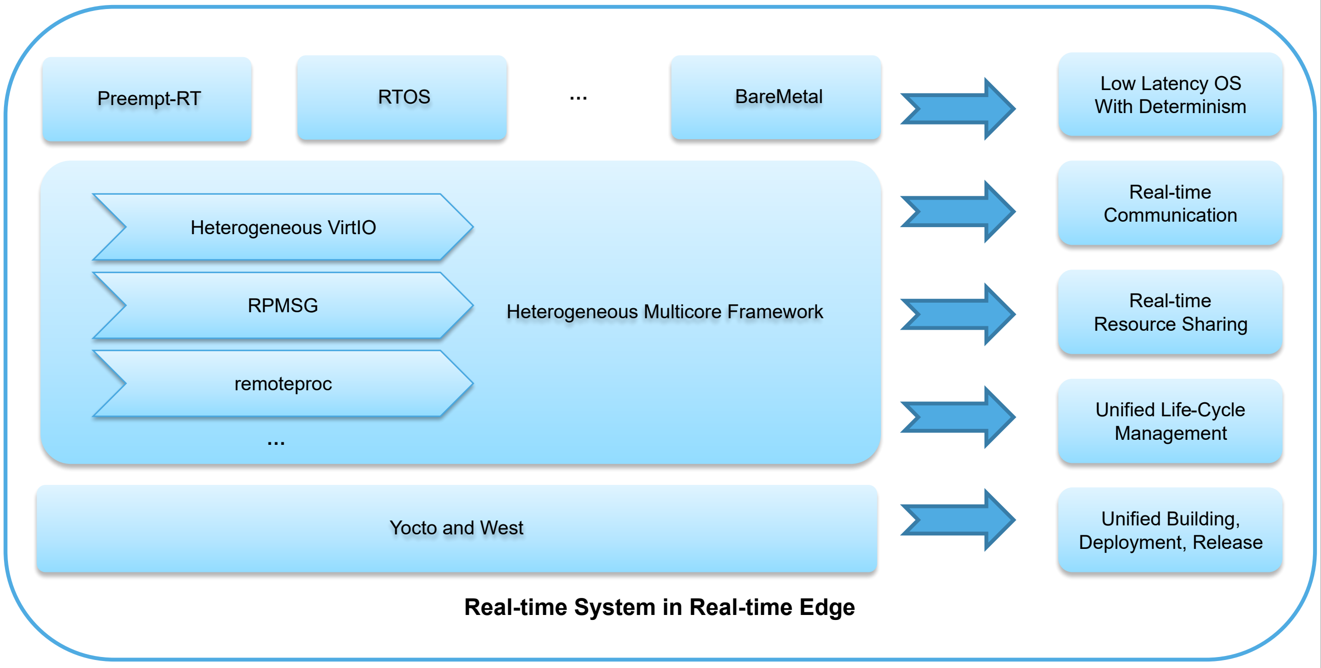

Preempt-RT
RTOS
...
BareMetal
Low Latency OS With Determinism
Heterogeneous VirtIO
RPMSG
remoteproc
...
Heterogeneous Multicore Framework
Real-time Communication
Real-time Resource Sharing
Unified Life-Cycle Management
Yocto and West
Unified Building, Deployment, Release
Real-time System in Real-time Edge

Figure 1. Real-time system

2.1 Real-time system features

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

- **Different frameworks and flexible combinations for different schedule latency requirements**
  The Figure 2 shows all the OSes/BareMetal supported in Real-time Edge on MPU platforms with different-scale schedule latency:
  – **Preempt-RT Linux on Cortex-A Core**
    Real-time Linux kernel provides deterministic low latency compared to Linux.
  – **BareMetal on Cortex-A Core**
    Single or multiple BareMetal instances run Cortex-A Cores with zero schedule latency.
  – **Native RTOS SMP/AMP on Cortex-A Core**
    Native RTOS (FreeRTOS or Zephyr) is kicked to one or more Cortex-A Cores from U-Boot. Jailhouse is not used and this combination targets lower latency and higher performance as compared to RTOS with Jailhouse.
  – **RTOS SMP/AMP on Corex-A Core with Jailhouse**
    RTOS (FreeRTOS or Zephyr) runs in Jailhouse inmate with hardware resource isolation on Cortex-A Core.
  – **RTOS and BareMetal on Cortex-M Core**
    This combination is used for Real-time Control systems, but has less CPU computing ability than Cortex-A Core.

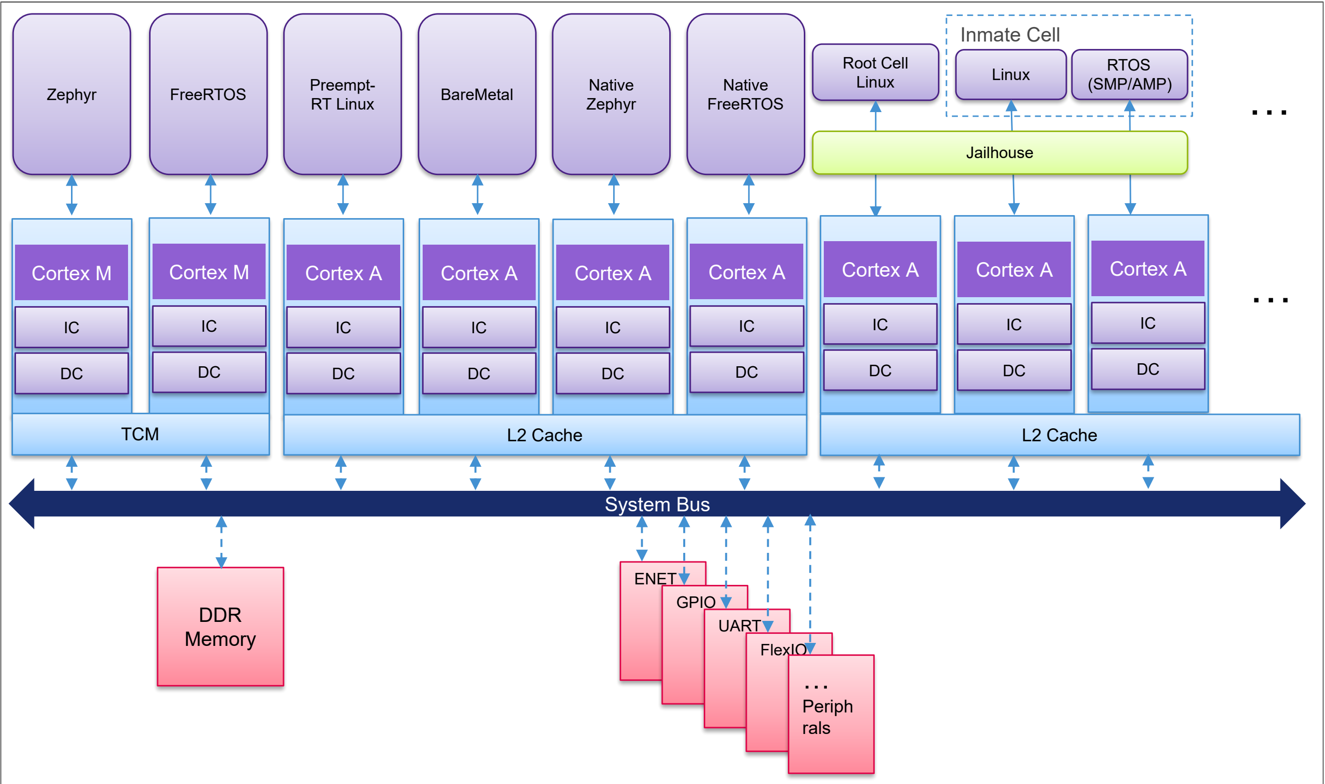


Figure 2. Flexible AMP in Real-time system

All these Real-time operating systems or BareMetal can be combined to be a flexible AMP system on multicore system. For example, the i.MX 8M Plus platform has four Cortex-A53 cores and one Cortex-M7 core. The Real-time Edge software supports the flexible AMP system to run these OS/BareMetal combinations:

– Four Cortex-A53 cores run SMP Preempt-RT Linux while the Cortex-M7 core runs RTOS.
– Four Cortex-A53 cores run four Baremetal/RTOS instances while the Cortex-M7 core runs RTOS.

  - One or more Cortex-A53 cores run Preempt-RT Linux while the other Cortex-A53 cores run one or more Baremetal/RTOS instances and theCortex-M7 core runs RTOS.
  - One or more Cortex-A53 cores run Preempt-RT Linux as the Jailhouse Root Cell. The other Cortex-A53 cores run as one or more inmate cells with RTOS.
- **Unified Software Building, Deploy, and Release**
  - All different OSes or applications running on different CPU cores are built via Yocto.
  - A bitbake command is used to create all images on different cores.
  - A single Flash Image includes all OSes or applications running on all CPU cores.
- **Heterogeneous Multicore Framework**
  A common framework with the following key features and functions:
  - **Inter-Core Data Communication and Resource Sharing**
    - The Common Heterogeneous Multicore Framework provides data communication and resource sharing between the M-Core and A-Cores or between different A-Cores simultaneously.
    - RPMSG provides standard message communication for low-bandwidth use cases.
    - Heterogeneous Multicore Virtio provides high-performance data path and resource sharing to meet high bandwidth requirement.
  - **Unified life cycle management for flexible AMP**

The Real-time Edge software supports Preempt-RT Linux, FreeRTOS, Zephyr, and BareMetal running on different processors with Heterogeneous Multicore Framework.

2.2 Real-time-system support matrix

The Table 4 shows the support matrix on NXP platforms:

Table 4. Real-time-system support matrix

Real-time System		i.MX 8M Mini LPDDR4 EVK		i.MX 8M Plus LPDDR4 EVK		i.MX 8DXL		i.MX 93 EVK		i.MX 93 EVK (14 x 14)		i.MX 943 19x19 LPDDR4/5 EVK		i.MX RT1180 EVK	i.MX 91 9x9 QSB	i.MX 91 11x11 EVK	i.MX 95 15x15 EVK		i.MX 95 19x19 EVK		LS1028ARDB	LS1043ARDB	LS1046ARDB	LX2160ARDB
Cores		**4 xA53**	**1 x M4**	**4 x A53**	**1 x M7**	**2 x A35**	**1 x M4**	**2 x A55**	**1 x M33**	**2 x A55**	**1 x M33**	**4 x A55**	**2 x M7**	**1 x M7 and 1 x M33**	**1 x A55**	**1 x A55**	**6 x A55**	**1 x M7**	**6 x A55**	**1 x M7**	**2 x A72**	**4 x A53**	**4 x A72**	**16 x A72**
Preempt-RT Linux		Y		Y		Y		Y		Y		Y			Y	Y	Y		Y		Y	Y	Y	Y
BareMetal		Y	Y	Y	Y			Y	Y												Y	Y	Y	Y
Native RTOS	FreeRTOS	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y				
	Zephyr	Y	Y	Y	Y			Y	Y	Y		Y			Y	Y	Y	Y	Y	Y				
Jailhouse	Baremetal	Y		Y				Y													Y	Y	Y	Y
	FreeRTOS	Y		Y				Y																
	Zephyr	Y		Y				Y																
	Harpoon	Y		Y				Y				Y												

2.3 Building, deploying, and releasing unified software

The Yocto project is an open source collaboration project that helps developers create custom Linux-based systems regardless of the hardware architecture. The project provides a flexible set of tools and a space where embedded developers worldwide can share technologies, software stacks, configurations, and best practices. These features can further be used to create tailored Linux images for embedded and IoT devices, or for a customized Linux OS. Moreover, the Linux factory selects Yocto as the building tool. Real-time Edge also selects Yocto as the unified SW release tool. Figure 3 shows the unified Yocto structure for Heterogeneous AMP.

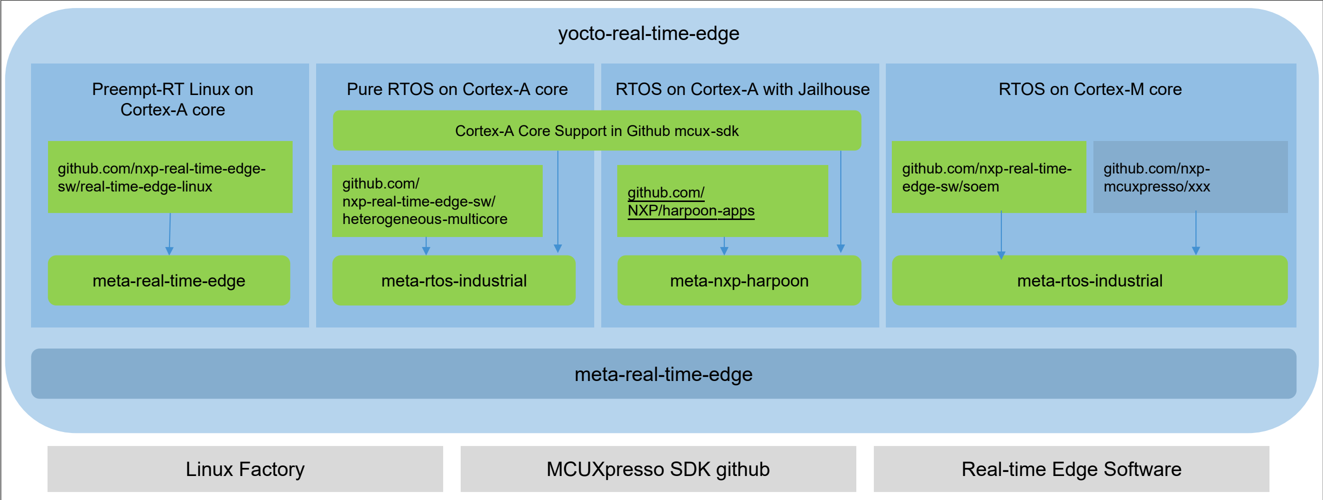

yocto-real-time-edge
Preempt-RT Linux on Cortex-A core
Pure RTOS on Cortex-A core
RTOS on Cortex-A with Jailhouse
RTOS on Cortex-M core
Cortex-A Core Support in Github mcux-sdk
github.com/nxp-real-time-edge-sw/real-time-edge-linux
github.com/nxp-real-time-edge-sw/heterogeneous-multicore
github.com/NXP/harpoon-apps
github.com/nxp-real-time-edge-sw/soem
github.com/nxp-mcuxpresso/xxx
meta-real-time-edge
meta-rtos-industrial
meta-nxp-harpoon
meta-rtos-industrial
meta-real-time-edge
Linux Factory
MCUXpresso SDK github
Real-time Edge Software

Figure 3. Unified Yocto structure of Heterogeneous AMP

The different Yocto Layers support different functions:

- The `meta-real-time-edge` layer focuses on building Linux and BareMetal application on Cortex-A core.
- The `meta-nxp-harpoon` layer focuses on building RTOS on Cortex-A core.
- The `meta-rtos-industrial` layer focuses on building RTOS running on both Cortex-A core and Cortex-M core.

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

For example:

```
# 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
```

2.3.1 Yocto layer for Cortex-A core

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

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

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

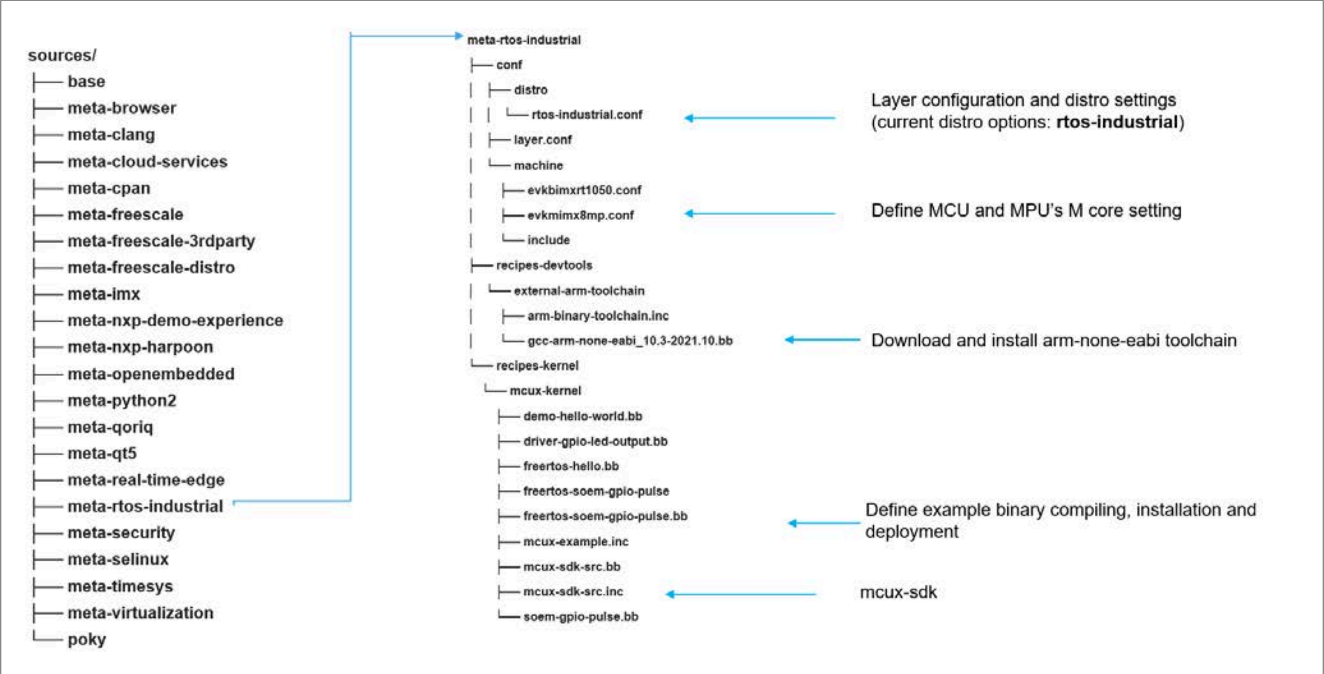
3. Baremetal application
   The scripts under `meta-real-time-edge/recipes-extended/real-time-edge-baremetal` describe how to build baremetal application on Cortex-A core. Refer to Section 2.5 for details.
4. Harpoon (RTOS on A core)
   Harpoon provides an environment to develop real-time demanding applications on an RTOS that runs on one or several Cortex-A cores. The application runs in parallel to a Linux distribution and uses the 64-bit Arm (R) architecture for higher performance. The system starts on Linux and the Jailhouse partitions the hardware to run both Linux and the guest RTOS in parallel. The hardware partitioning is configurable and depends on the use case. The Yocto layer `meta-nxp-harpoon` describes how to build these applications on Cortex-A core. For more information, refer to the Harpoon User's Guide. See Section 1.4.

2.3.2 Yocto layer for Cortex-M core

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

2.3.2.1 Introduction to meta-rtos-industrial

The Figure 4 shows the `meta-rtos-industrial` file structure.



```
sources/
├── base
├── meta-browser
├── meta-clang
├── meta-cloud-services
├── meta-cpan
├── meta-freescale
├── meta-freescale-3rdparty
├── meta-freescale-distro
├── meta-imx
├── meta-nxp-demo-experience
├── meta-nxp-harpoon
├── meta-openembedded
├── meta-python2
├── meta-qoriq
├── meta-qt5
├── meta-real-time-edge
├── meta-rtos-industrial
├── meta-security
├── meta-selinux
├── meta-timesys
├── meta-virtualization
└── poky
```

```
meta-rtos-industrial
├── conf
│  ├── distro
│  │  └── rtos-industrial.conf
│  ├── layer.conf
│  └── machine
│     ├── evkbimxrt1050.conf
│     ├── evkmimx8mp.conf
│     └── include
├── recipes-devtools
│  └── external-arm-toolchain
│     ├── arm-binary-toolchain.inc
│     └── gcc-arm-none-eabi_10.3-2021.10.bb
└── recipes-kernel
   └── mcux-kernel
      ├── demo-hello-world.bb
      ├── driver-gpio-led-output.bb
      ├── freertos-hello.bb
      ├── freertos-soem-gpio-pulse
      ├── freertos-soem-gpio-pulse.bb
      ├── mcux-example.inc
      ├── mcux-sdk-src.bb
      ├── mcux-sdk-src.inc
      └── soem-gpio-pulse.bb
```

Layer configuration and distro settings (current distro options: **rtos-industrial**)
Define MCU and MPU's M core setting
Download and install arm-none-eabi toolchain
Define example binary compiling, installation and deployment
mcux-sdk

Figure 4. RTOS Industrial Layer structure

2.3.2.1.1 Source code definition

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

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

If a new repo must be downloaded, append a new line to “SRC_URI” with the URL and location of the required repo.

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

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

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

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

For example:

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

2.3.2.1.2 Example definition

The file `mcux-examples.inc` describes the common method to compile, install, and deploy examples. Each example `bb` file must include this file and then specify the folder of the example.

Use the command below to add a new example:

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

2.3.2.1.3 Toolchain definition

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

This layer also supports an external toolchain. Parameter “`ARMGCC_DIR`” can be overwritten to point the external toolchain.

For example:

```
ARMGCC_DIR = “/MYPATH/arm-none-eabi”
```

2.3.2.2 Integration of meta-rtos-industrial

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

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

2.3.2.3 Building meta-rtos-industrial

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

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

```
$ 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-walnascar -m real-time-edge-3.3.0.xml
$ repo sync
$ DISTRO=nxp-real-time-edge MACHINE=imx8mm-lpddr4-evk \
  source real-time-edge-setup-env.sh -b build-real-time-edge
$ bitbake nxp-image-real-time-edge
```

The sample binary files are located under the `tmp/deploy/images/imx8mm-lpddr4-evk/examples` and `/examples` directories of rootfs.

```
examples/
├── heterogeneous-multicore
│   ├── hello-world-freertos
│   │   ├── hello_world_ca53.bin
│   │   ├── hello_world_ca53.elf
│   │   ├── hello_world_ca53_RTOS0_RAM_CONSOLE-0x94bff000.bin
│   │   ├── hello_world_ca53_RTOS0_RAM_CONSOLE-0x94bff000.elf
│   │   ├── hello_world_ca53_RTOS0_UART4.bin
│   │   ├── hello_world_ca53_RTOS0_UART4.elf
│   │   ├── hello_world_ca53_RTOS1_RAM_CONSOLE-0x95bff000.bin
│   │   ├── hello_world_ca53_RTOS1_RAM_CONSOLE-0x95bff000.elf
│   │   ├── hello_world_ca53_RTOS2_RAM_CONSOLE-0x96bff000.bin
│   │   ├── hello_world_ca53_RTOS2_RAM_CONSOLE-0x96bff000.elf
│   │   ├── hello_world_ca53_RTOS3_RAM_CONSOLE-0x97bff000.bin
│   │   ├── hello_world_ca53_RTOS3_RAM_CONSOLE-0x97bff000.elf
│   │   ├── hello_world_ca53_RTOS3_UART2.bin
│   │   ├── hello_world_ca53_RTOS3_UART2.elf
│   │   ├── hello_world_cm4.bin
│   │   └── hello_world_cm4.elf
│   ├── hello-world-zephyr
│   │   ├── hello_world_ca53_RTOS0_RAM_CONSOLE-0x93d00000.bin
│   │   ├── hello_world_ca53_RTOS0_RAM_CONSOLE-0x93d00000.elf
│   │   ├── hello_world_ca53_RTOS0_UART4.bin
│   │   ├── hello_world_ca53_RTOS0_UART4.elf
│   │   ├── hello_world_ca53_RTOS1_RAM_CONSOLE-0x94d00000.bin
│   │   ├── hello_world_ca53_RTOS1_RAM_CONSOLE-0x94d00000.elf
│   │   ├── hello_world_ca53_RTOS2_RAM_CONSOLE-0x95d00000.bin
│   │   ├── hello_world_ca53_RTOS2_RAM_CONSOLE-0x95d00000.elf
│   │   ├── hello_world_ca53_RTOS3_RAM_CONSOLE-0x96d00000.bin
│   │   ├── hello_world_ca53_RTOS3_RAM_CONSOLE-0x96d00000.elf
│   │   ├── hello_world_ca53_RTOS3_UART2.bin
│   │   └── hello_world_ca53_RTOS3_UART2.elf
│   ├── lwip-ping-freertos
│   │   ├── lwip_ping_ca53.bin
│   │   └── lwip_ping_ca53.elf
│   ├── rpmsg-str-echo-freertos
│   │   ├── rpmsg_str_echo_ca53.bin
│   │   ├── rpmsg_str_echo_ca53.elf
│   │   ├── rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0x94bff000.bin
│   │   ├── rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0x94bff000.elf
│   │   ├── rpmsg_str_echo_ca53_RTOS0_UART4.bin
│   │   ├── rpmsg_str_echo_ca53_RTOS0_UART4.elf
│   │   ├── rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0x95bff000.bin
│   │   ├── rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0x95bff000.elf
│   │   ├── rpmsg_str_echo_cm4.bin
│   │   └── rpmsg_str_echo_cm4.elf
│   ├── rpmsg-uart-sharing-freertos
│   │   ├── rpmsg_uart_sharing_cm4.bin
│   │   └── rpmsg_uart_sharing_cm4.elf
│   ├── rt-latency-freertos
│   │   ├── rt_latency_ca53.bin
│   │   └── rt_latency_ca53.elf
│   ├── rt-latency-zephyr
│   │   ├── rt_latency_ca53.bin
│   │   └── rt_latency_ca53.elf
│   ├── soem-digital-io-freertos
│   │   ├── soem_digital_io_ca53.bin
│   │   └── soem_digital_io_ca53.elf
│   ├── soem-digital-io-zephyr
│   │   ├── soem_digital_io_ca53.bin
│   │   └── soem_digital_io_ca53.elf
│   ├── soem-servo-freertos
│   │   ├── soem_servo_ca53.bin
│   │   └── soem_servo_ca53.elf
│   ├── soem-servo-rt1180-freertos
│   │   ├── soem_servo_rt1180_ca53.bin
│   │   └── soem_servo_rt1180_ca53.elf
│   ├── soem-servo-rt1180-zephyr
│   │   ├── soem_servo_rt1180_ca53.bin
│   │   └── soem_servo_rt1180_ca53.elf
│   ├── soem-servo-zephyr
│   │   ├── soem_servo_ca53.bin
│   │   └── soem_servo_ca53.elf
│   ├── virtio-net-backend-freertos
│   │   ├── virtio_net_backend_ca53.bin
│   │   ├── virtio_net_backend_ca53.elf
│   │   ├── virtio_net_backend_cm4.bin
│   │   └── virtio_net_backend_cm4.elf
│   └── virtio-perf-freertos
│       ├── virtio_perf_ca53.bin
│       ├── virtio_perf_ca53.elf
│       ├── virtio_perf_cm4.bin
│       └── virtio_perf_cm4.elf
├── igh-ethercat-userspace
│   └── ec_motor_example
└── mcuxsdk
    ├── demo-hello-world
    │   ├── hello_world_cm4.bin
    │   └── hello_world_cm4.elf
    ├── freertos-hello
    │   ├── freertos_hello_cm4.bin
    │   └── freertos_hello_cm4.elf
    ├── igpio-led-output
    │   ├── igpio_led_output_cm4.bin
    │   └── igpio_led_output_cm4.elf
    ├── iuart-9bit-interrupt-transfer
    │   ├── iuart_9bit_interrupt_transfer_cm4.bin
    │   └── iuart_9bit_interrupt_transfer_cm4.elf
    ├── iuart-9bit-polling
    │   ├── iuart_9bit_polling_cm4.bin
    │   └── iuart_9bit_polling_cm4.elf
    ├── soem-gpio-pulse-bm
    │   ├── soem_gpio_pulse_bm_cm4.bin
    │   └── soem_gpio_pulse_bm_cm4.elf
    ├── soem-gpio-pulse-freertos
    │   ├── soem_gpio_pulse_freertos_cm4.bin
    │   └── soem_gpio_pulse_freertos_cm4.elf
    ├── soem-servo-motor-bm
    │   ├── soem_servo_motor_bm_cm4.bin
    │   └── soem_servo_motor_bm_cm4.elf
    ├── soem-servo-motor-freertos
    │   ├── soem_servo_motor_freertos_cm4.bin
    │   └── soem_servo_motor_freertos_cm4.elf
    ├── soem-servo-motor-rt1180-bm
    │   ├── soem_servo_motor_rt1180_bm_cm4.bin
    │   └── soem_servo_motor_rt1180_bm_cm4.elf
    └── soem-servo-motor-rt1180-freertos
        ├── soem_servo_motor_rt1180_freertos_cm4.bin
        └── soem_servo_motor_rt1180_freertos_cm4.elf
```

To compile a special example, use the following command:

For example:

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

2.4 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 feature 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.

2.4.1 System Real-time Latency tests

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

2.4.1.1 Optimizations in Preempt-RT Linux

1. **Linux kernel optimizations**
   a. **Linux kernel configuration optimizations**: `CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE=y`: The CPU frequency scaling feature, also known as Dynamic Frequency Scaling (DFS), can change the CPU core frequency at runtime. However, this feature introduces additional latency.
   Note: *Users should note that the Real-time Edge software sets the CPU frequency governor to performance to use the highest frequency supported by the CPU.*
   b. **Linux kernel driver optimizations**: Linux kernel drivers for some Ethernet IPs are used on NXP SoCs. For example, i.MX 8M Plus EVK uses ENET_QOS, LX2160A uses DPAA2 Ethernet, and LS1028A uses ENETC. These drivers are patched to be compatible with PREEMPT-RT. As a result, the scheduling latency under high network traffic is reduced significantly when using PREEMPT-RT Linux.

   ```
   [PATCH net] net: stmmac: use __napi_schedule() for PREEMPT_RT
   ```

```
[PATCH net-next 1/2] dpaa2-eth: use napi_schedule to be compatible with
 PREEMPT_RT
[PATCH net-next 2/2] enetc: use napi_schedule to be compatible with
 PREEMPT_RT
```

c. **Ethernet interface optimizations**: To avoid introduing extra latency by the software IRQ used by NAPI, change to use `threaded NAPI` of the Ethernet interfaces, which is an operating mode that uses dedicated kernel threads rather than software IRQ context for NAPI processing.

```
echo 1 > /sys/class/net/eth<n>/threaded
```

2. **Linux real-time application optimizations**
   Depending on the actual real-time application, some of the following optimization techniques can be applied to get better real-time performance.
   - Real-time scheduling policy and priority SCHED_FIFO and SCHED_RR are the two real-time scheduling policies. Processes scheduled under one of the real-time policies (SCHED_FIFO, SCHED_RR) have a scheduling priority in the range 1 (low) to 99 (high). The `chrt` command can be used to set the real-time scheduling policy and priority of a process. The `sched_setscheduler()` system call can be used for the same purpose. More information on how to build a real-time application can be found at the below link: https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/application_base
   - CPU isolation
     The `isolcpus=` kernel boot parameter can isolate CPU core from the kernel scheduler. In this way, the isolated CPU core can be used to run real-time application. for examaple:

     ```
     isolcpus=2,3 nohz_full=2,3 rcu_nocbs=2,3
     ```
   - CPU affinity
     The `taskset` command can be used to set the CPU affinity of a real-time application. Alternatively, the `sched_setaffinity()` system call can be used for the same purpose. for example:

     ```
     taskset -c 2 cyclictest ...
     ```
   - SMP IRQ affinity
     Some interrupts introduce jitter and havea negative impact on real-time applications. The affinity of these interrupts can be controlled using the `proc` file system `/proc/irq/<irq>/smp_affinity`. In this way, the interrupts can be moved away from the core running real-time application. For example:

     ```
     echo 1 > /proc/irq/<irq>/smp_affinity
     ```

2.4.1.2 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 the Cyclictest Web Page.*

2.4.2 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): This scheduling policy 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 a `nice value` of `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 but is slightly different. A task running under this policy 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.

2.5 Baremetal on Cortex-A core

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

- Features supported
- Getting started with the BareMetal framework using the supported platforms:
  – NXP Layerscape platforms
  – i.MX 8M / i.MX 93 platforms

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

2.5.1 BareMetal framework

The BareMetal framework supports the scenarios that need low latency, real-time response, and high performance. There is no OS running on the cores and customer-specific application runs on the core directly. The Figure 5 depicts the BareMetal framework architecture.

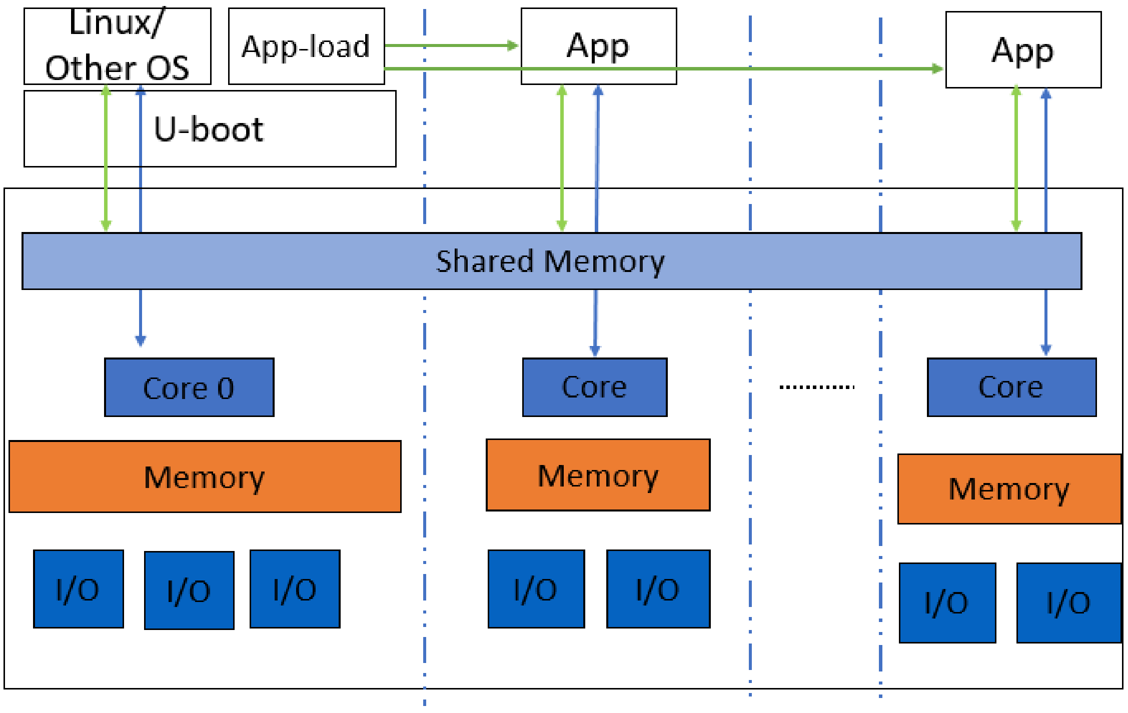


Figure 5. BareMetal framework architecture

The main features of the BareMetal framework are as follows:

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

The primary core0 runs the BareMetal under primary mode. It then loads the BareMetal application to the secondary cores and starts the BareMetal application. The Figure 6 depicts the boot flow diagram:

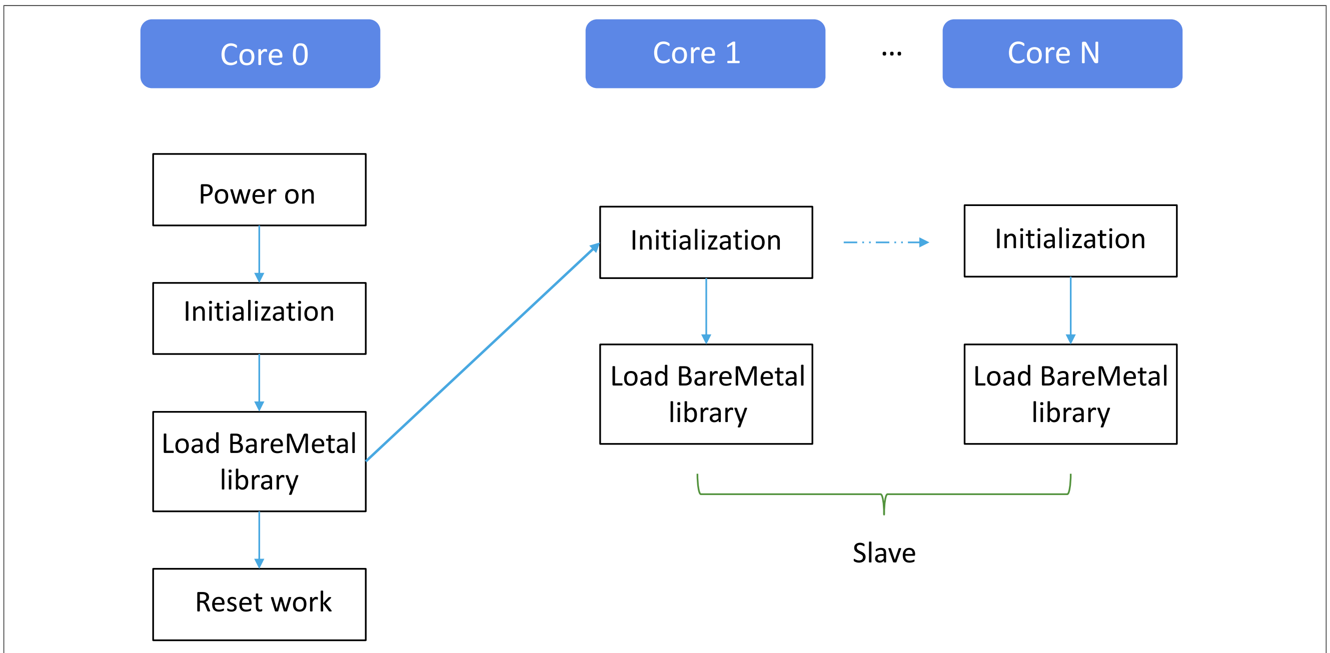


Figure 6. BareMetal framework boot flow diagram

The Table 5 lists the industrial IoT features supported by various NXP processors and boards.

Table 5. BareMetal 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
i.MX 93	i.MX 93 EVK	UART, IPI, data transfer, Ethernet
	i.MX 93 9x9 LPDDR4 QSB	UART, IPI, data transfer
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, GPIO
LX2160A/Rev2	LX2160ARDB	UART, IPI, data transfer

Typical use cases are as follows:

1. Core0 as a primary core runs Linux to manage secondary cores and communicate with the server. Secondary cores run Baremetal application for real-time processing. Refer Figure 7.

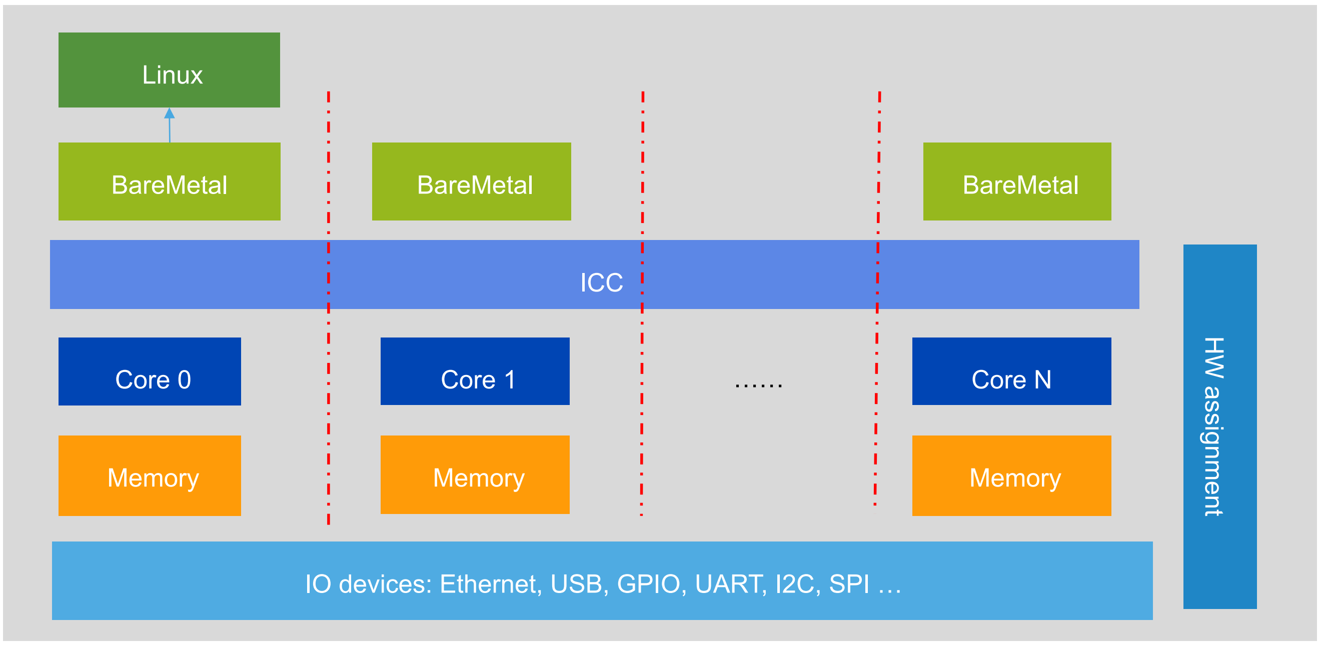


Figure 7. BareMetal Use Case 1

2. All cores run the BareMetal application for real-time processing. See Figure 8.

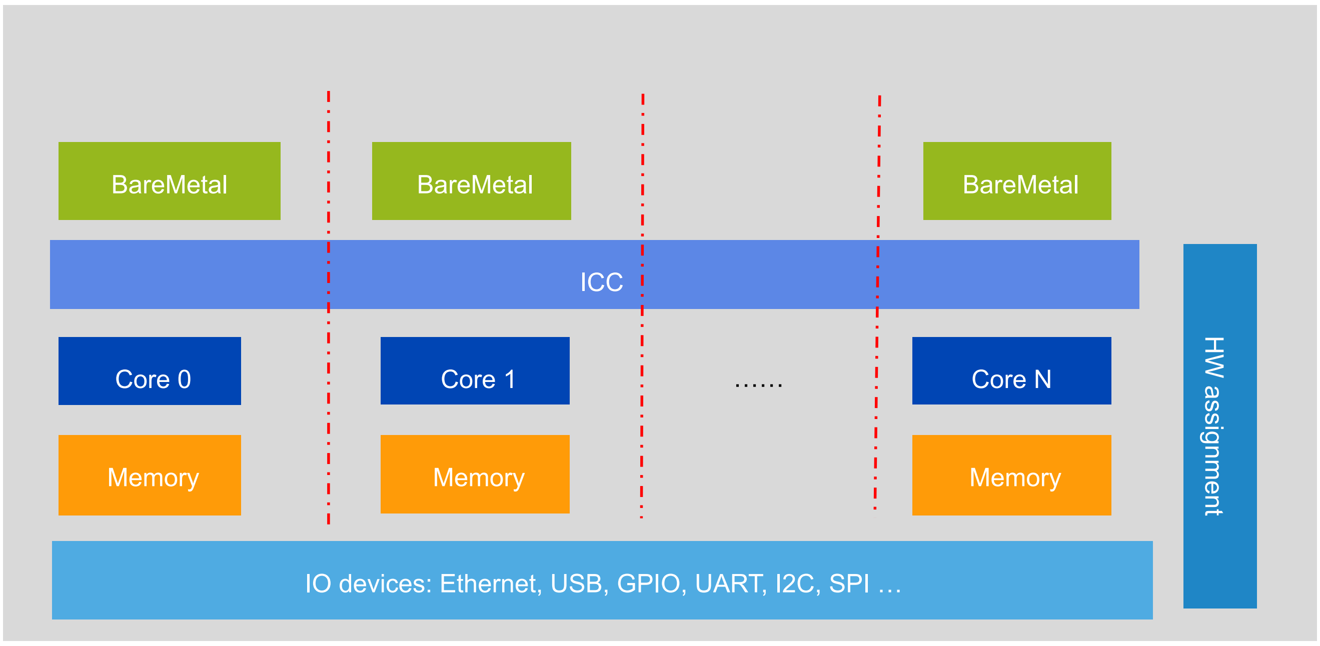


Figure 8. BareMetal Use Case 2

2.5.2 Getting started

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

2.5.2.1 Hardware and software requirements

Running the BareMetal framework scenarios requires the following:

- **Hardware**: i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK, i.MX 93 9x9 LPDDR4 QSB, LS1028ARDB, LS1043ARDB, LS1046ARDB, LX2160ARDB, and serial cables.
- **Software**: Real-time Edge Software v2.7 release or later.

2.5.2.2 Hardware setup

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

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

Follow the steps below.

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

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

   Use `/dev/ttyUSB2` for core0 (primary core) and `/dev/ttyUSB3` for core1, core2, and core3 (secondary cores).
2. **i.MX 8M Mini LPDDR4 EVK**: There is one USB MicroB Debug port on board. Two UART ports can be found when the MicroB cable connects to PC.

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

   Use `/dev/ttyUSB1` for core0 (primary core) and `/dev/ttyUSB0` for core1, core2, and core3 (secondary cores).
3. **GPIO setup**
   For GPIO test on i.MX 8M Plus LPDDR4 EVK, connect pin 7 and pin 8 of J21 by a jumper, as shown in Figure 9.

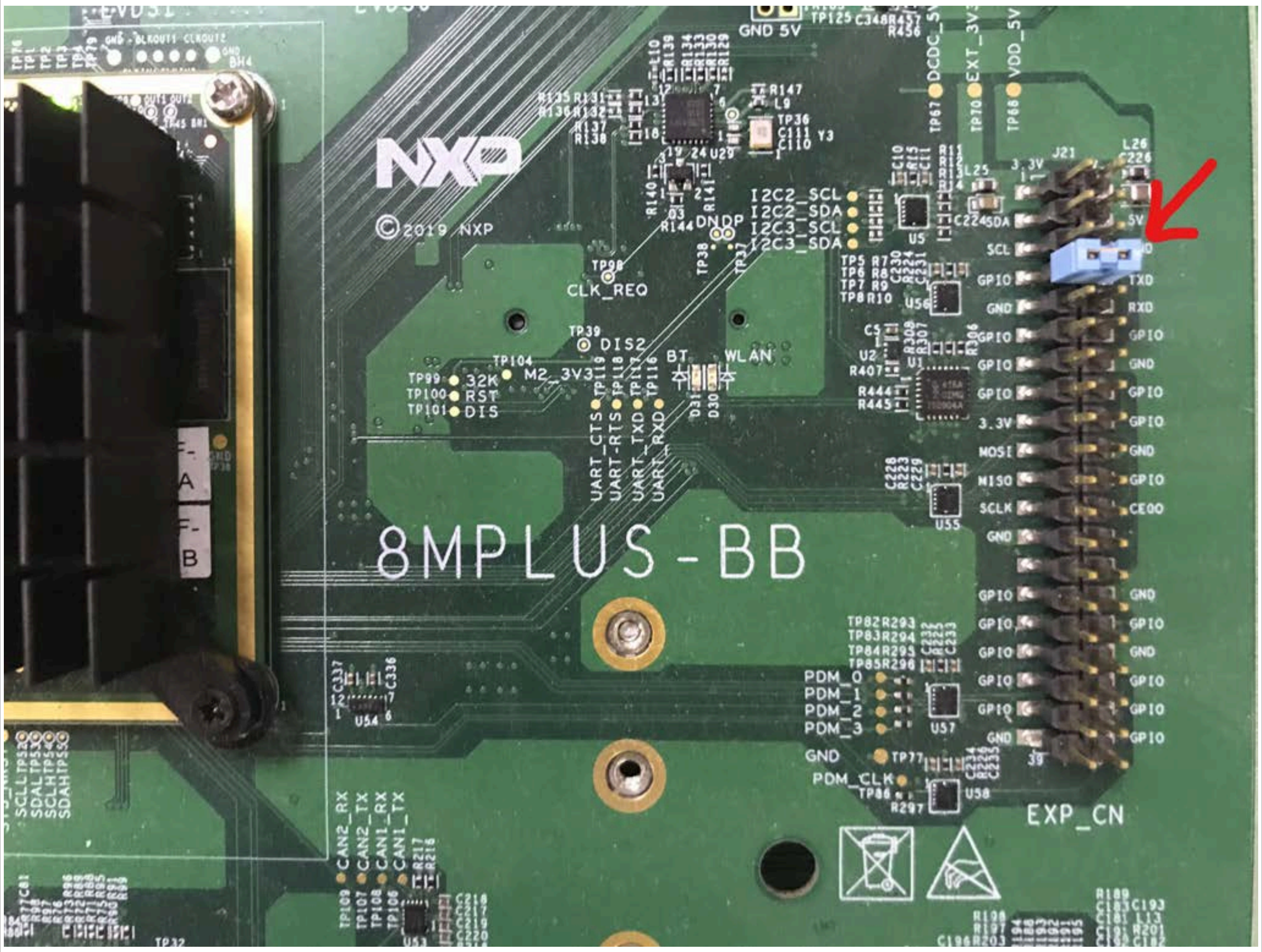


Figure 9. Connections for GPIO test on i.MX 8M Plus LPDDR4 EVK board

4. For GPIO test on i.MX 8M Mini LPDDR4 EVK, connect pin7 and pin8 of J1003 by a jumper as shown in Figure 10.

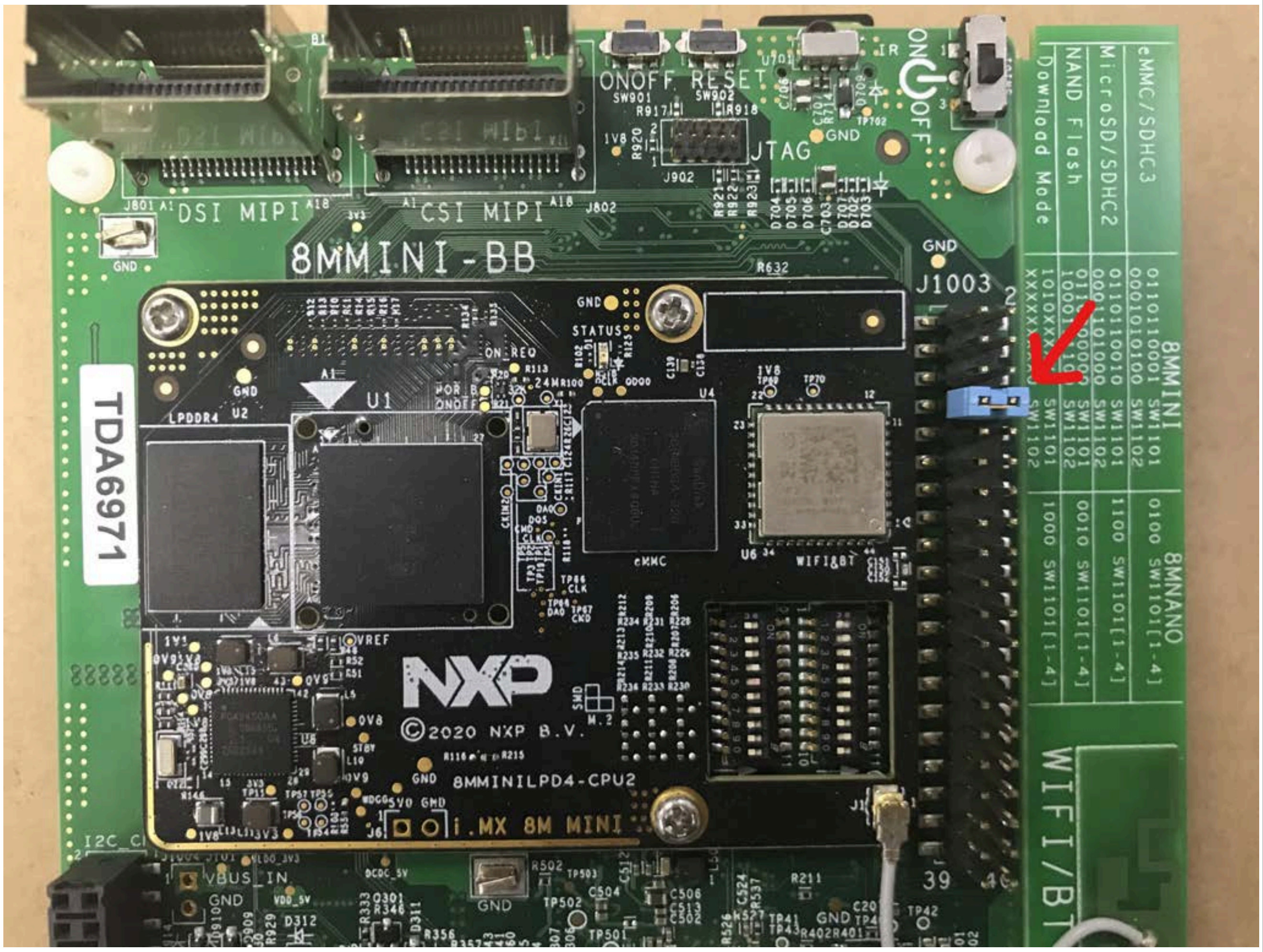


Figure 10. Connections for GPIO test on i.MX 8M Mini LPDDR4 EVK board

2.5.2.2.2 LS1028ARDB, LX2160ARDB, LS1043ARDB, or LS1046ARDB

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

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

2.5.2.2.3 i.MX 93 EVK

On the i.MX 93 EVK board, the USB Type-C connector (J1401) provides four UART ports when connected to a PC using USB cable. The third port (LPUART1) is used for core0 (primary core) and the fourth port (LPUART2) is used for core1 (secondary core).

2.5.2.2.4 i.MX93 9x9 LPDDR4 QSB

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

2.5.2.3 Building the software

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 the Real-time Edge framework. This method is described in the document, Real-time Edge Yocto Project User Guide in the Section "*Building the image through Yocto*".

2.5.2.3.1 Building the BareMetal binary for secondary 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-v3.3-baremetal-202512`
3. Configure the cross-toolchain in 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_baremetal_slave_defconfig
 $ make
/* build Baremetal image for i.MX 8M Plus LPDDR4 EVK board */
 $ make imx8mp_evk_baremetal_slave_defconfig
 $ make
/* build Baremetal image for i.MX 93 EVK board */
 $ make imx93_11x11_evk_baremetal_slave_defconfig
 $ make
/* build Baremetal image for i.MX 93 9x9 LPDDR4 QSB board */
 $ make imx93_9x9_qsb_baremetal_slave_defconfig
 $ make
/* build Baremetal image for LS1028ARDB board */
 $ make ls1028ardb_baremetal_slave_defconfig
 $ make
/* build Baremetal image with SAI for LS1028ARDB board */
 $ make ls1028ardb_baremetal_slave_sai_defconfig
 $ make
/* build Baremetal image for LS1043ARDB board */
 $ make ls1043ardb_baremetal_slave_defconfig
 $ make
/* build Baremetal image for LS1046ARDB board */
 $ make ls1046ardb_baremetal_slave_defconfig
 $ make
/* build Baremetal image for LX2160ARDB board */
 $ make lx2160ardb_baremetal_slave_defconfig
 $ make
```

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

To get the code and build images for these platforms, follow the Real-time Edge Software Yocto Project. See Section 1.4

2.5.2.3.2 Building the image through Yocto

There are two methods to build the Baremetal images. The first method, which is used to compile the images in a standalone way, is described in Section 2.5.2.3. The other method is to build the Baremetal images using Real-time Edge software framework, and is described in this section.

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

2.5.2.3.2.1 Getting Real-time Edge software

The latest release is available at the following URL:

https://github.com/nxp-real-time-edge-sw/yocto-real-time-edge.git

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

2.5.2.3.2.2 Building the Baremetal images

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

Building the Baremetal images for various boards

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

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

For i.MX 93 EVK Baremetal image:

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

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

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

For LS1028ARDB Baremetal image:

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

For LS1043ARDB Baremetal image:

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

For LS1046ARDB Baremetal image:

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

For LX2160ARDB Baremetal image:

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

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

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

2.5.2.4 Booting up Linux with BareMetal

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

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

1. Insert an SD card (at least 4 GB size) into any Linux host machine.
2. Find the image file in the build directory (for example: `ls1028ardb`):
   ```
   tmp/deploy/images/ls1028ardb/nxp-image-real-time-edge-ls1028ardb.wic.zst
   ```
3. Then, run the following commands:
   ```
   $ zstd -d nxp-image-real-time-edge-ls1028ardb.wic.zst
   $ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wic of=/dev/sdx bs=1M
    conv=fsync
   # or in some other host machine:
   $ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wic of=/dev/mmcblkx bs=1M
    conv=fsync
   # find the right SD Card device name in your host machine and replace the
    “sdx” or “mmcblkx”.
   ```
4. Then, insert the SD card into the target board (for example, LS1028ARDB) and power on.

After completion of the above-mentioned steps, the Linux system boots up on the primary core (core 0). The BareMetal system boots up on the secondary core (core 1) automatically.

2.5.3 Running examples

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

2.5.3.1 Preparing the console

In current Baremetal framework design, two UART ports are used as console. One UART is used for primary core and the other UART is used for secondary cores. Refer to Section 3.2.2.2 for preparing the console.

2.5.3.2 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 2.5.2.3 and Section 2.5.2.3.2 respectively.

- If the Real-time Edge software is used to compile the Baremetal image, the Baremetal image is included in the `nxp-image-real-time-edge-xxxx.wic.zst`. In this case, the primary core starts the Baremetal image on secondary cores automatically.
- If a standalone compilation method is used, perform the steps below to run the Baremetal binary from th U-Boot prompt of the primary core. See the below example run on Layerscape platform:
  1. After starting U-Boot on the master, download the bare metal image: `u-boot-dtb.bin` on 0x84000000 using the command below:
     ```
     => tftp 0x84000000 xxxx/u-boot-dtb.bin
     ```
     Where
     – `xxxx` is your tftp server directory.
     – 0x84000000 is the address of `CONFIG_TEXT_BASE` on bare metal for Layerscape platforms.
     Note:
     a. *The address is `0x50200000` for i.MX 8M Plus LPDDR4 EVK and i.MX 8M Mini LPDDR4 EVK boards.*
     b. *The address is `0x90200000` for i.MX 93 EVK board.*
  2. Then, start the Baremetal cores using the command below:
     ```
     =>  dcache flush; cpu 1 release 0x84000000
     ```
     Note: *In the command `cpu <num> release 0x84000000`, the 'num' can be 1, 2, 3, ... to the maximum CPU number.*
     For i.MX 8M Plus LPDDR4 EVK and i.MX 8M Mini LPDDR4 EVK boards, use the below command:
     ```
     => dcache flush;cpu 1 release 50200000;sleep 6;cpu 2 release 50200000;sleep
      2;cpu 3 release 50200000;
     ```
  3. Last, the UART2 port displays the logs, and the bare metal application runs on secondary cores successfully.

2.5.4 Development based on Baremetal framework

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

2.5.4.1 Developing the Baremetal application

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

2.5.4.2 Main file app.c

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

- When the standalone method is used to build the Baremetal image as described in Section 2.5.2.3, change the directory to `U-boot path` to read or edit the `app.c` file.
- When the Real-time Edge software is used to compile the Baremetal binary, change to the building directory to view or edit the `app.c` file.

The following is a sample code of the file `app.c` that shows how to add the example test cases of I2C, IRQ, and GPIO.

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

2.5.4.3 Common header files

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

Table 6. 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`	This file includes APIs for manipulating strings. For example, `strlen`, `strcpy`, and `strcmp`.
`linux/delay.h`	This file includes APIs used for small pauses. For example, `udelay` and `mdelay`.
`linux/types.h`	This file includes APIs specifying common types. For example, `_u32` and `_u64`.
`common.h`	This file includes the common APIs. For example, `printf` and `puts`.

2.5.4.4 GPIO example

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

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

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

The Table 7 shows the APIs used in the file `test_gpio.c` application example.

Table 7. GPIO APIs and their description

Function declaration	Description
`int dm_gpio_lookup_name(const char *name, struct gpio_desc *desc)`	Looks up a named GPIO and returns its description. `name`: Name to look up, such as GPIO5_7 `desc`: GPIO description Returns: `0` if OK, `-ve` on error

Table 7. GPIO APIs and their description*...continued*

Function declaration	Description
`int dm_gpio_request(struct gpio_desc *desc, const char *label)`	Manually requests a GPIO. `desc`: GPIO description of GPIO to request. `label`: Label to attach to the GPIO while claimed, such as "output1". Returns: `0` if OK, `-ve` on error
`int dm_gpio_set_value(const struct gpio_desc *desc, int value)`	Configures the direction of GPIO to OUT and writes the value to it. `desc`: GPIO description. `value`: the value written to this GPIO. Returns: `0` if OK, `-ve` on error.
`int dm_gpio_set_dir_flags(struct gpio_desc *desc, ulong flags)`	Sets direction using description and added flags `desc`: GPIO description `flags`: New flags to use Returns: `0` if OK, `-ve` on error.
`int dm_gpio_free(struct udevice *dev, struct gpio_desc *desc)`	Frees a single GPIO. `dev`: Device that requested the GPIO. `desc`: GPIO description. Returns: `0` if OK, `-ve` on error.

2.5.4.5 I2C example

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

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

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

Table 8. 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)`	Reads data from an 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 an 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.

2.5.4.6 IRQ example

The file `uboot/app/test_irq_init.c` provides 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 BareMetal cores. To communicate between the BareMetal core and the Linux core, refer to Section 2.5.4.16. SGI IRQ id is 0-15. The SGI IRQ id '8' is reserved for ICC.

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

The Table 9 shows the APIs used in the sample file, `test_irq_init.c`.

Table 9. IRQ APIs and their description

Return type API name (parameter list)	Description
`int irq_desc_register(struct irq *irq_data, void (*irq_handle)(int, int, void *), void *data)`	Registers an IRQ function. • `irq_data`- Include IRQ id(0-15 for SGI, 16-31 for PPI, 32-1019 for SPI), and IRQ dev • `irq_handle` – IRQ function • `data` – IRQ data
`void gic_send_sgi(u32 id, int core_mask)`	Sets a SGI IRQ signal. • `core_mask` – target core mask • `id` – IRQ id
`int irq_set_affinity(struct irq *irq, int core_mask)`	Sends the target core for hw IRQ. • `core_mask` – target core mask • `irq` – IRQ data, include IRQ id
`int irq_set_polarity(struct udevice *dev, uint id, bool active_low)`	Sets the type for hardware IRQ to identify whether the corresponding interrupt is edge-triggered or level-sensitive. • `dev` – IRQ dev • `id` – IRQ id • `active_low` – true if active low, false for active high

2.5.4.7 QSPI example

The file `uboot/app/test_qspi.c` provides an example that can be used to test the QSPI feature. The below steps show how to write an 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 10 shows the APIs used in the file `test_qsip.c` example.

Table 10. 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.

Table 10. QSPI APIs...*continued*

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

2.5.4.8 IFC example

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

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

```
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), the below command can be used to verify it:

```
=> md 0x60000000
1:60000000: 55aa55aa 0001ee01 10001008 0000000a    .U.U............
1:60000010: 00000000 00000000 02005514 12400080    .........U....@.
1:60000020: 005002e0 002000c1 00000000 00000000    ..P... .........
1:60000030: 00000000 00880300 00000000 01110000    ................
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
```

2.5.4.9 Ethernet example

The file `uboot/app/test_net.c` provides an example to test the Ethernet feature and shows how to write a net application for using this feature.

Here is an example for the LS1043ARDB (or LS1046ARDB) board.

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

1. Connect one Ethernet port of LS1043ARDB board to one host machine using Ethernet cable.
   - For LS1046ARDB, the default `ethact` is FM1@DTSEC5. Network cable must be connected to the SGMII1 port.
   - For LS1043ARDB, the default `ethact` is FM1@DTSEC3. Network cable must be connected to the 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 11. 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 the 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 the length.

2.5.4.10 USB example

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

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

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

Table 12. 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.

2.5.4.11 PCIe example

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

1. Insert a PCIe network card (such as e1000) into a 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 13. PCIe APIs and their description

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

Table 13. PCIe APIs and their description...*continued*

API name (type)	Description
`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.

2.5.4.12 ENETC example

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

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

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

The table below lists the Net APIs for ENETC and their description, refer to Section 2.5.4 for other Net APIs.

Table 14. 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.

2.5.4.13 SAI example

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

2.5.4.13.1 Synchronous Audio Interface (SAI)

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

• Transmitter with independent bit clock and frame sync supporting 1 data line

• Receiver with independent bit clock and frame sync supporting 1 data line

• Maximum Frame Size of 32 words

• Word size of between 8-bits and 32-bits

• Word size configured separately for first word and remaining words in frame

• Asynchronous 32 × 32-bit FIFO for each transmit and receive channel

• Supports graceful restart after FIFO error

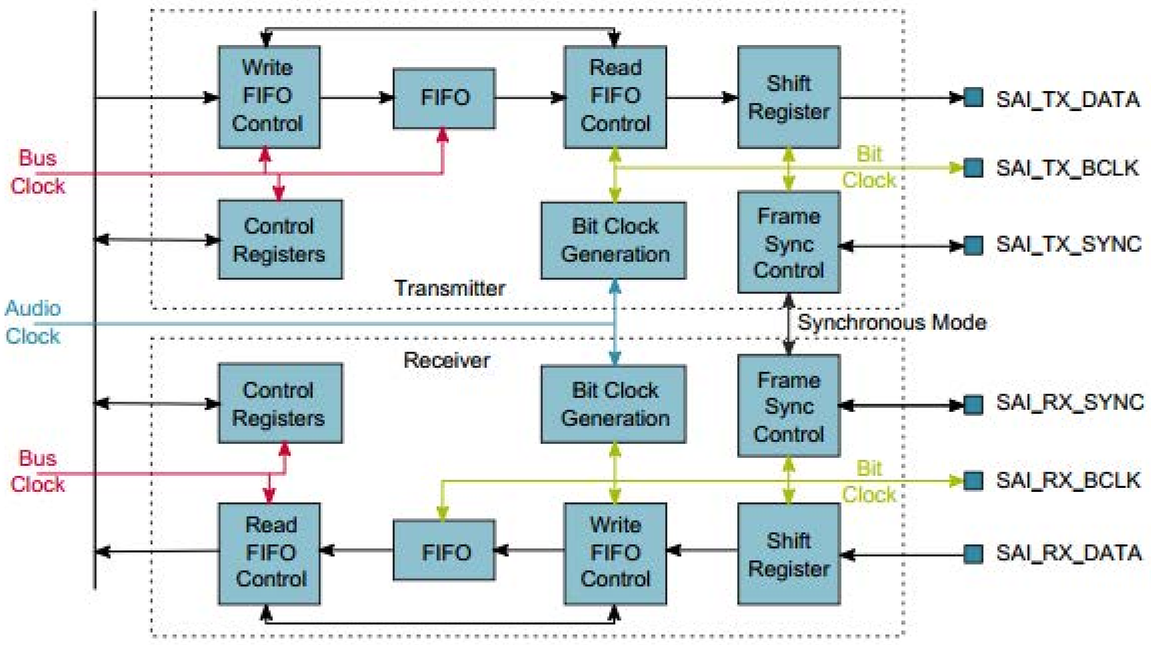


Figure 11. SAI block diagram

2.5.4.13.2 Audio codec (SGTL5000)

The SGTL5000 is a low-power stereo codec with headphone amplifier from NXP. It is designed to provide a complete audio solution for products requiring LINEIN, MIC_IN, LINEOUT, headphone-out, and digital I/Os. It allows an 8.0 MHz to 27 MHz system clock as input. The codec supports the following sampling frequencies:

- 8.0 kHz
- 11.025 kHz
- 12 kHz
- 16 kHz
- 22.05 kHz
- 24 kHz

- 32 kHz
- 44.1 kHz
- 48 kHz
- 96 kHz

The LS1028ARDB board provides a 25 MHz crystal oscillator to the SGTL5000.

The SGTL5000 provides two interfaces (I2C and SPI) to set up registers. The LS1028ARDB board uses the I2C interface.

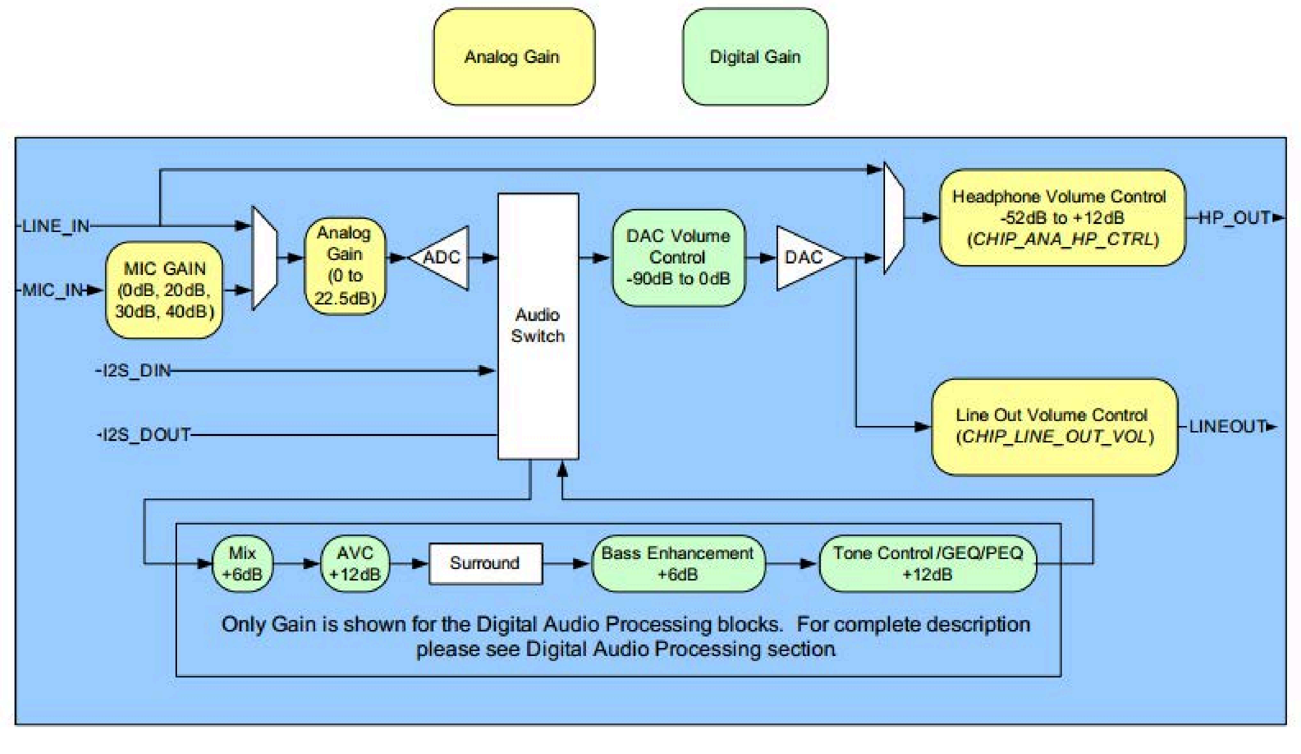

Analog Gain
Digital Gain
LINE_IN
MIC_IN
MIC GAIN (0dB, 20dB, 30dB, 40dB)
Analog Gain (0 to 22.5dB)
ADC
Audio Switch
DAC Volume Control -90dB to 0dB
DAC
Headphone Volume Control -52dB to +12dB (CHIP_ANA_HP_CTRL)
HP_OUT
I2S_DIN
I2S_DOUT
Line Out Volume Control (CHIP_LINE_OUT_VOL)
LINEOUT
Mix +6dB
AVC +12dB
Surround
Bass Enhancement +6dB
Tone Control /GEQ/PEQ +12dB
Only Gain is shown for the Digital Audio Processing blocks. For complete description please see Digital Audio Processing section

Figure 12. System block diagram, signal flow, and gain

2.5.4.13.3 Digital interface formats

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

• **I2S Format (n = bit length)**

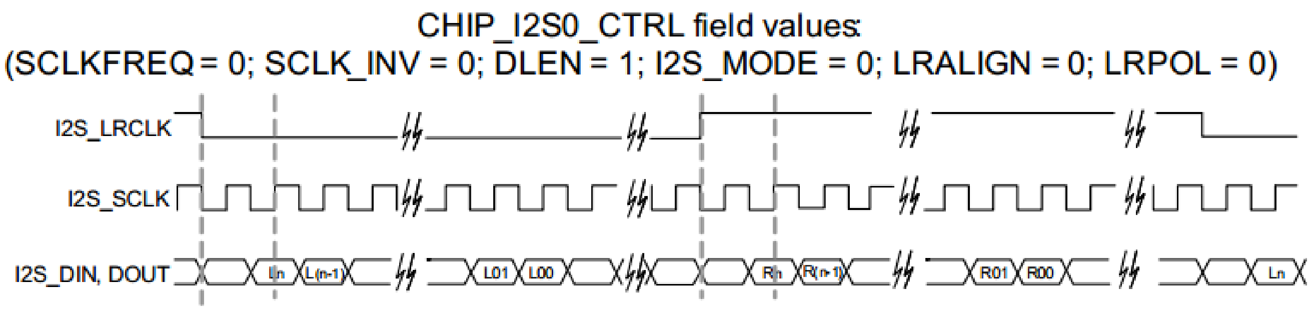

CHIP_I2S0_CTRL field values:
(SCLKFREQ = 0; SCLK_INV = 0; DLEN = 1; I2S_MODE = 0; LRALIGN = 0; LRPOL = 0)
I2S_LRCLK
I2S_SCLK
I2S_DIN, DOUT
Ln
L(n-1)
L01
L00
Rn
R(n-1)
R01
R00
Ln

Figure 13. I2S Format (n = bit length)

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

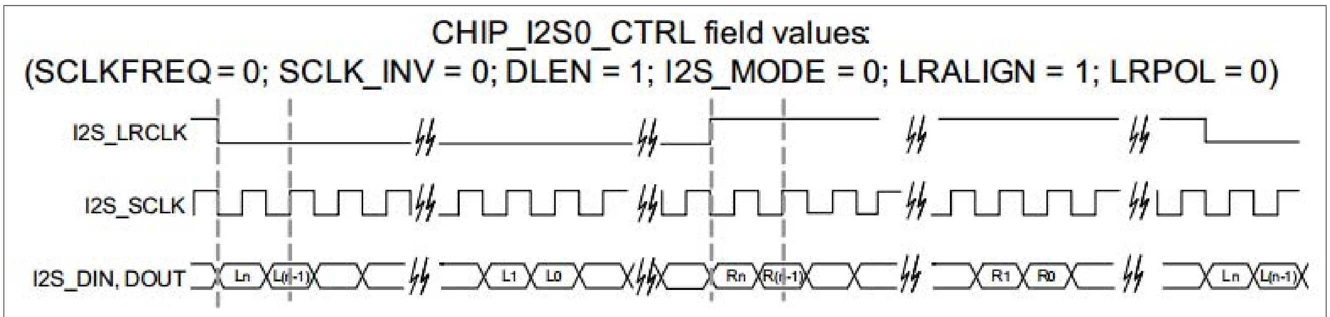

CHIP_I2S0_CTRL field values:
(SCLKFREQ = 0; SCLK_INV = 0; DLEN = 1; I2S_MODE = 0; LRALIGN = 1; LRPOL = 0)
I2S_LRCLK
I2S_SCLK
I2S_DIN, DOUT
Ln
L(n-1)
L1
L0
Rn
R(n-1)
R1
R0
Ln
L(n-1)

Figure 14. Left Justified Format (n = bit length)

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

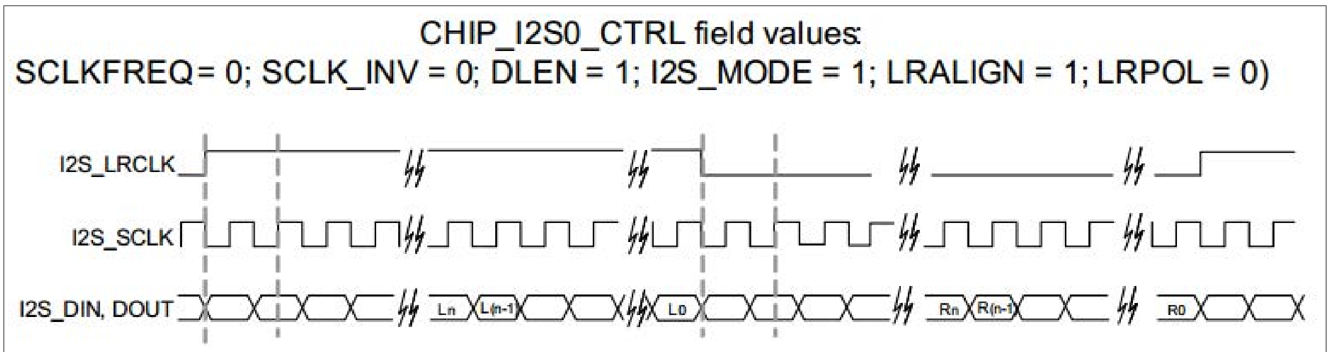

CHIP_I2S0_CTRL field values:
SCLKFREQ = 0; SCLK_INV = 0; DLEN = 1; I2S_MODE = 1; LRALIGN = 1; LRPOL = 0)
I2S_LRCLK
I2S_SCLK
I2S_DIN, DOUT
Ln
L(n-1)
L0
Rn
R(n-1)
R0

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

- **PCM Format A**

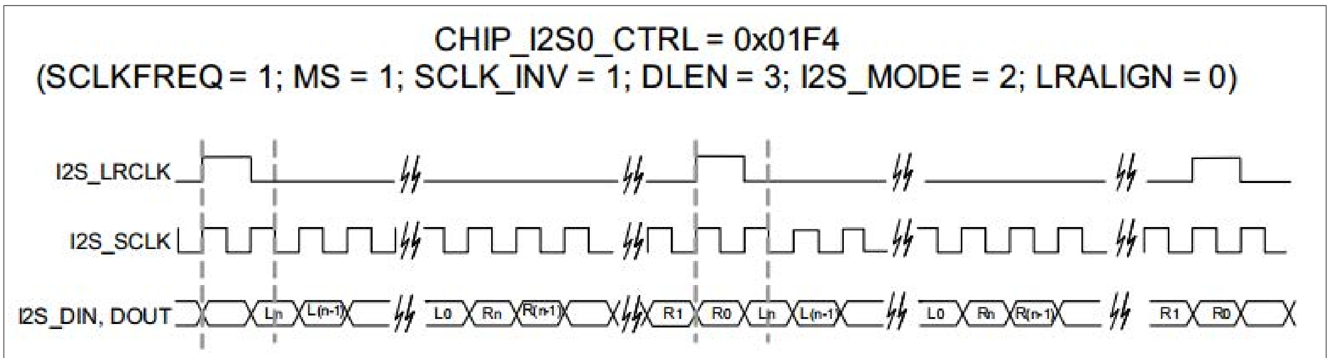


Figure 16. PCM Format A

- **PCM Format B**

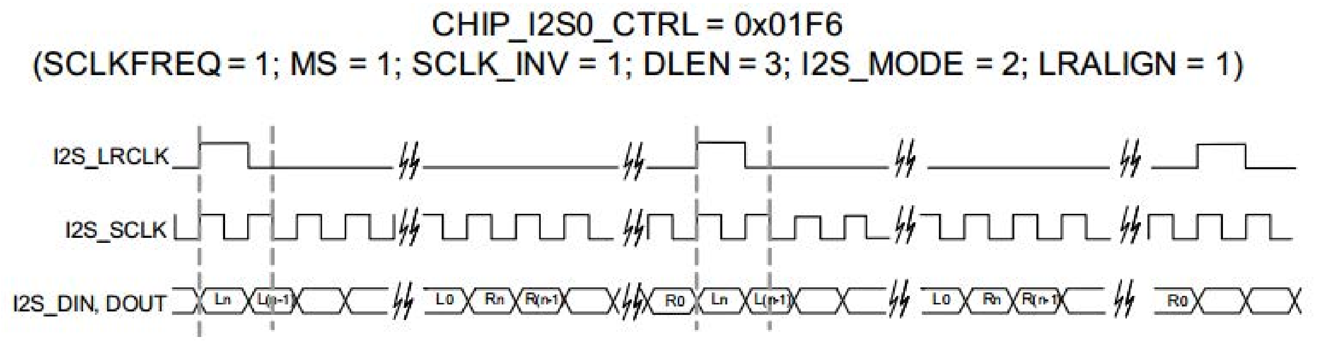

CHIP_I2S0_CTRL = 0x01F6
(SCLKFREQ = 1; MS = 1; SCLK_INV = 1; DLEN = 3; I2S_MODE = 2; LRALIGN = 1)
I2S_LRCLK
I2S_SCLK
I2S_DIN, DOUT
Ln
L(n-1)
L0
Rn
R(n-1)
R0

Figure 17. PCM Format B

2.5.4.13.4 Running the SAI application

To run the SAI application, BareMetal images should be rebuilt with SAI support.

1. Enable SAI support in Real-time Edge software as follows:

```
$ 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 the secondary 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!
 ***********************************************************
```

2.5.4.14 FlexTimer example

The FlexTimer module (FTM) works on BareMetal core as the wake-up source for LS1046ARDB. It can support nanosecond (ns) level alarm setting. There is no example code to test the module. However, a few commands can be used to verify these features.

Use the below commands to verify the FTM feature:

- Use the “`ftm`” command to get help information:

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

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

- Use the “`ftm start [count]`” command to start the ftm test:

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

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

- Use the “`ftm stop`” command to stop the ftm test and show the test result:

```
=> ftm stop
1:FTM test stop.
1:irq count | total (us)  | average (us) | max (us) | min (us)  |
1:3087560   | 309579251   | 100          | 102      | 100       |
```

- Use the “`ftm show`” command to show the test result:

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

The table below lists the attributes for “`ftm show`” and “`ftm stop`” result:

Table 15. FlexTimer module attributes and their description

Attribute Name	Description
irq count	Generated interrupt single count since the “ftm start” command
total (us)	The time since “ftm start” command
average (us)	The average time between two interrupt signals
max (us)	The maximum time between two interrupt signals
min (us)	The minimum time between two interrupts signals

The table below lists the FTM APIs and their description.

Table 16. FlexTimer module APIs

API Name	Description
`int ftm_rtc_set_alarm_by_us (struct udevice *dev, unsigned long us, void (* func)(void *))`	Setting alarm by us count • `struct udevice *dev` – device struct of ftm • `unsigned long us` – the time for ftm alarm • `void (* func)(void *)` – the handle function when time up

Table 16. FlexTimer module APIs*...continued*

API Name	Description
`void ftm_rtc_set_alarm (struct udevice *dev, u16 ticks, void (* func)(void *));`	Setting alarm by ftm timer count • `struct udevice *dev` – device struct of ftm • `u16 ticks` – the timer counter for ftm alarm • `void (* func)(void *)` – the handle function when time up
`void ftm_rtc_alarm_stop(struct udevice *dev)`	Stop and reset ftm alarm • `struct udevice *dev` – device struct of ftm
`unsigned long ftm_rtc_get_max_alarm_us (struct udevice *dev)`	Get the max alarm time value for ftm alarm • `struct udevice *dev` – device struct of ftm

2.5.4.15 Newlib's math library

To control IO devices such as to change the speed or angle, mathematical calculations are required. The math library provided by Newlib is added to support such calculations. Newlib is a C library intended for use on embedded systems.

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

math

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

To use the math library, include the below code in the file header. Then, the user 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 detailed usage, refer to the example file `math.c` under the `cmd` folder. The example shows how to call the math library API including `acos/asin/atan/cos/sin/tan` and `log/pow/sqrt`. 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
```

2.5.4.16 ICC module

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

The 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 must be defined in the 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 the next.

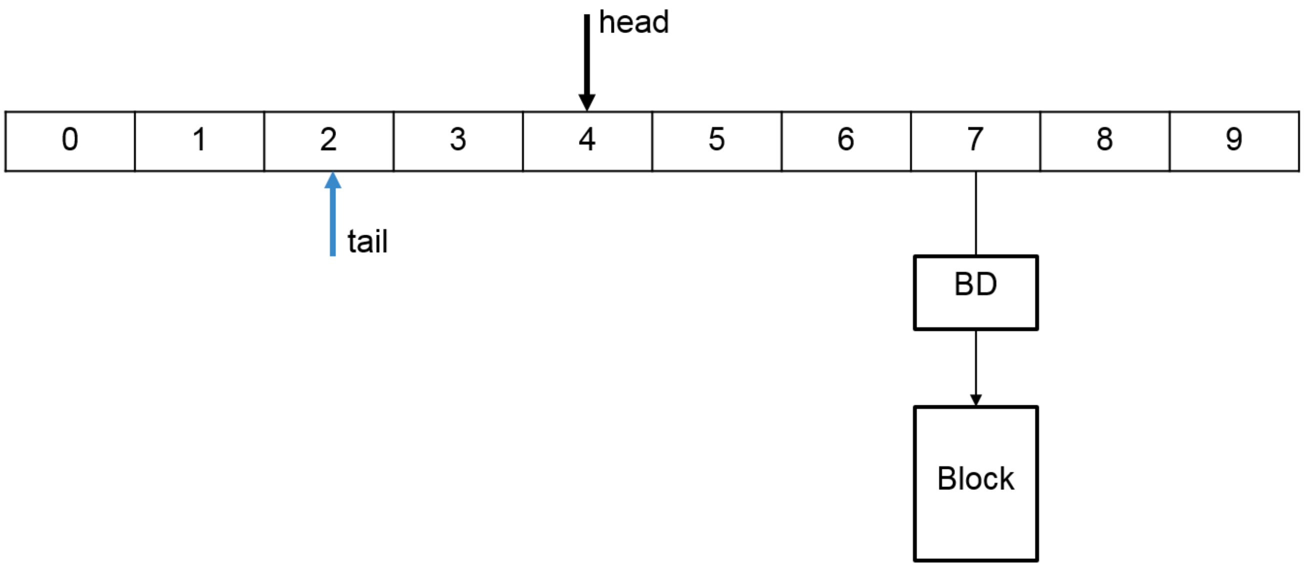


Figure 18. BD rings for inter-core communication

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.)

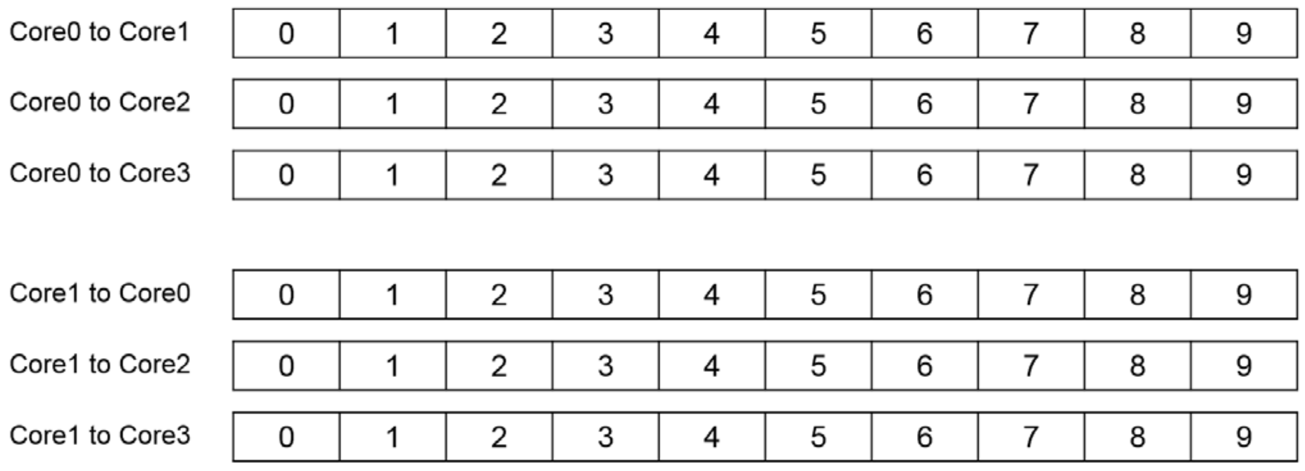


Figure 19. BD rings for multicore platform

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.

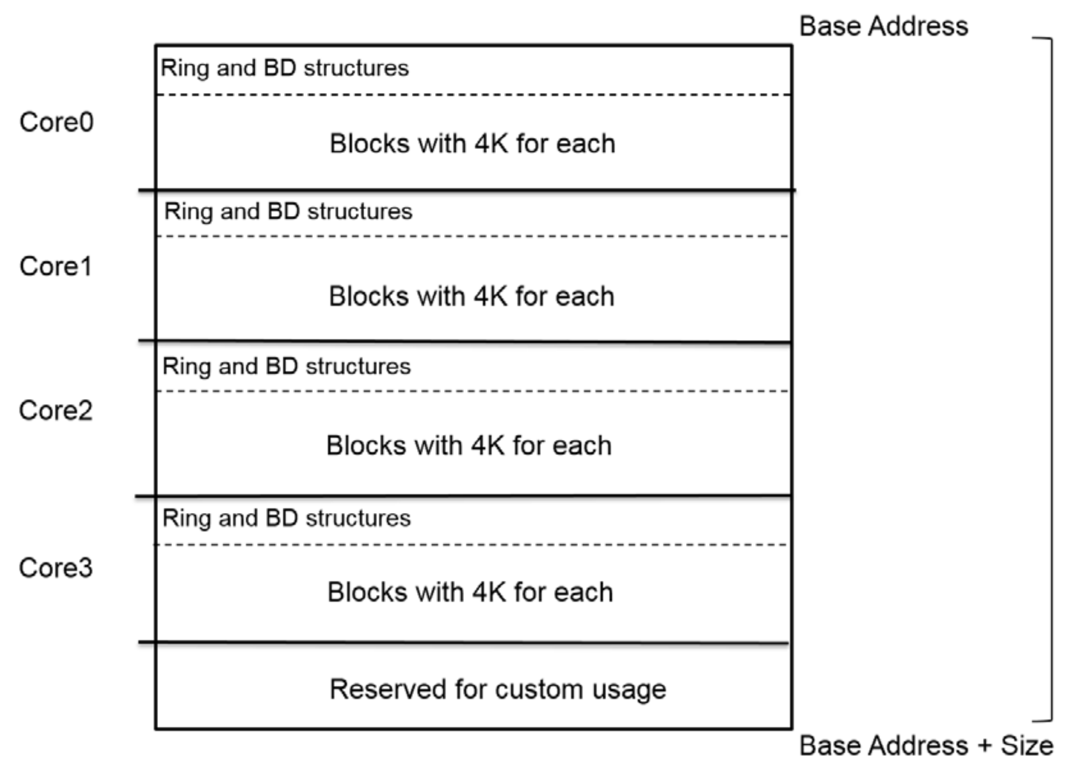


Figure 20. ICC shared memory map for the four-core platform

Generally, Core 0 runs Linux as the primary core while other cores run BareMetal as the secondary core. 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 the LS1028ARDB platform with two cores, the shared memory map is defined as:

- The total shared memory size is 256 MB.
- The reserved space for custom usage is 16 MB at the top of the share memory space.
- Core 0 runs Linux as the primary core. The shared memory size for ICC is 120 MB, in which the ring and BD structure space is 2 M. The block space for data is 118 MB with 4K for each block.
- Core 1 runs BareMetal as the secondary core. The shared memory size for ICC is 120 MB, in which the ring and BD structure space is 2M. 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 primary core and secondary 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 secondary core, works for data transfer between BareMetal cores and primary 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` --- includes 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 --- includes the example case commands.
```

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

Table 17. 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.

2.5.4.16.1 ICC examples

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

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

```
[root@LS1046ARDB ~] # icc
icc show - Shows all icc rings status at this core
icc perf <core_mask> <counts> - ICC performance to cores <core_mask> with
 <counts> bytes
icc send <core_mask> <data> <counts> - Sends <counts> <data> to cores
 <core_mask>
icc irq <core_mask> <irq> - Sends SGI <irq> ID[0 - 15] to <core_mask>
icc read <addr> <counts> - Reads <counts> 32bit register from <addr>
icc write <addr> <data> - Writes <data> to a register <addr>
```

Likewise, in Baremetal system, use the command `icc` to view the supported cases.

```
=> icc
1:icc - Inter-core communication via SGI interrupt
1:Usage:
icc show                                - Show all icc rings status at this
 core
icc perf <core_mask> <counts>           - 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
```

2.5.4.17 Single hardware interrupt routed to multiple cores

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

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

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

In `fsl-ls1046a-rdb.dts`

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

In `imx8mm-evk.dts`

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

On LS1046ARDB, GPIO2_01 and GPIO2_02 are selected. GPIO2_01 is used to simulate an external hardware whereas GPIO2_02 triggers an interrupt. Connect the GPIO2_01 and GPIO2_02 pins on the board. TP14 and TP13 are connected to GPIO2_01 and GPIO2_02 separately. The figure below shows how to connect TP14 and TP13.

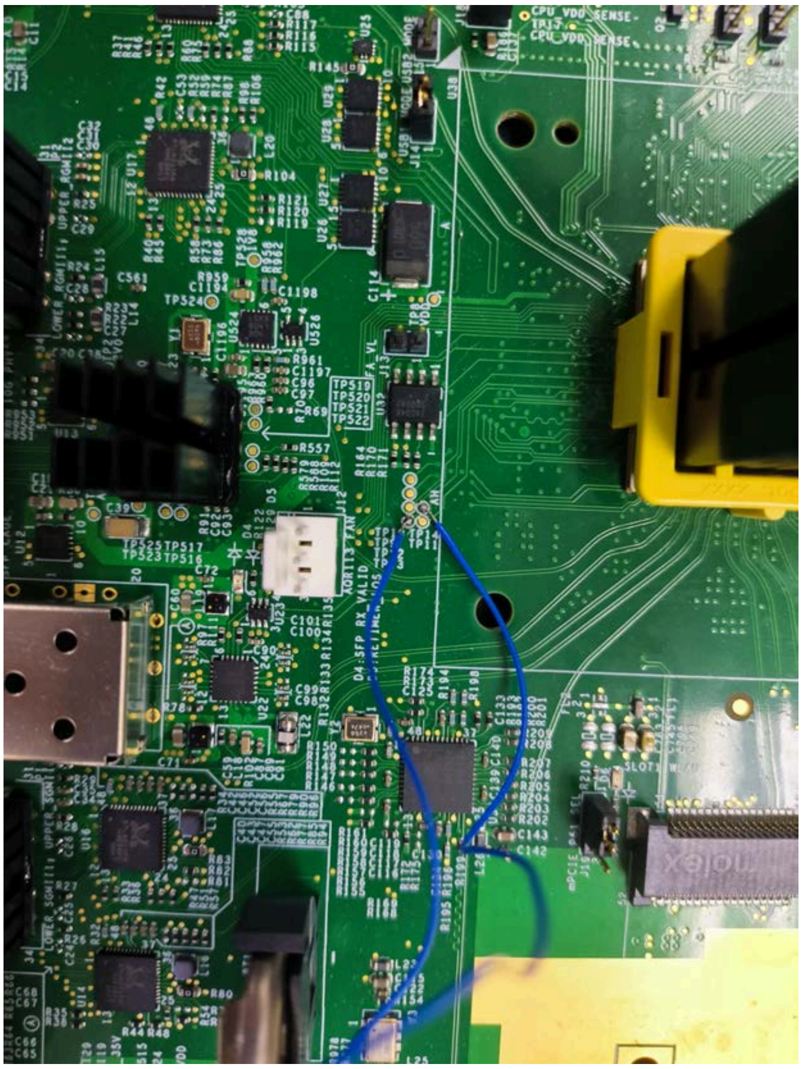


Figure 21. LS1046ARDB hardware interrupt routing to multiple cores

On i.MX 8M Mini, connect the pins GPIO5_07 and GPIO5_08 on the board. GPIO5_07 is used to simulate an external hardware whereas GPIO5_08 triggers an interrupt. The figure below shows how to connect GPIO5_07 and GPIO5_08.

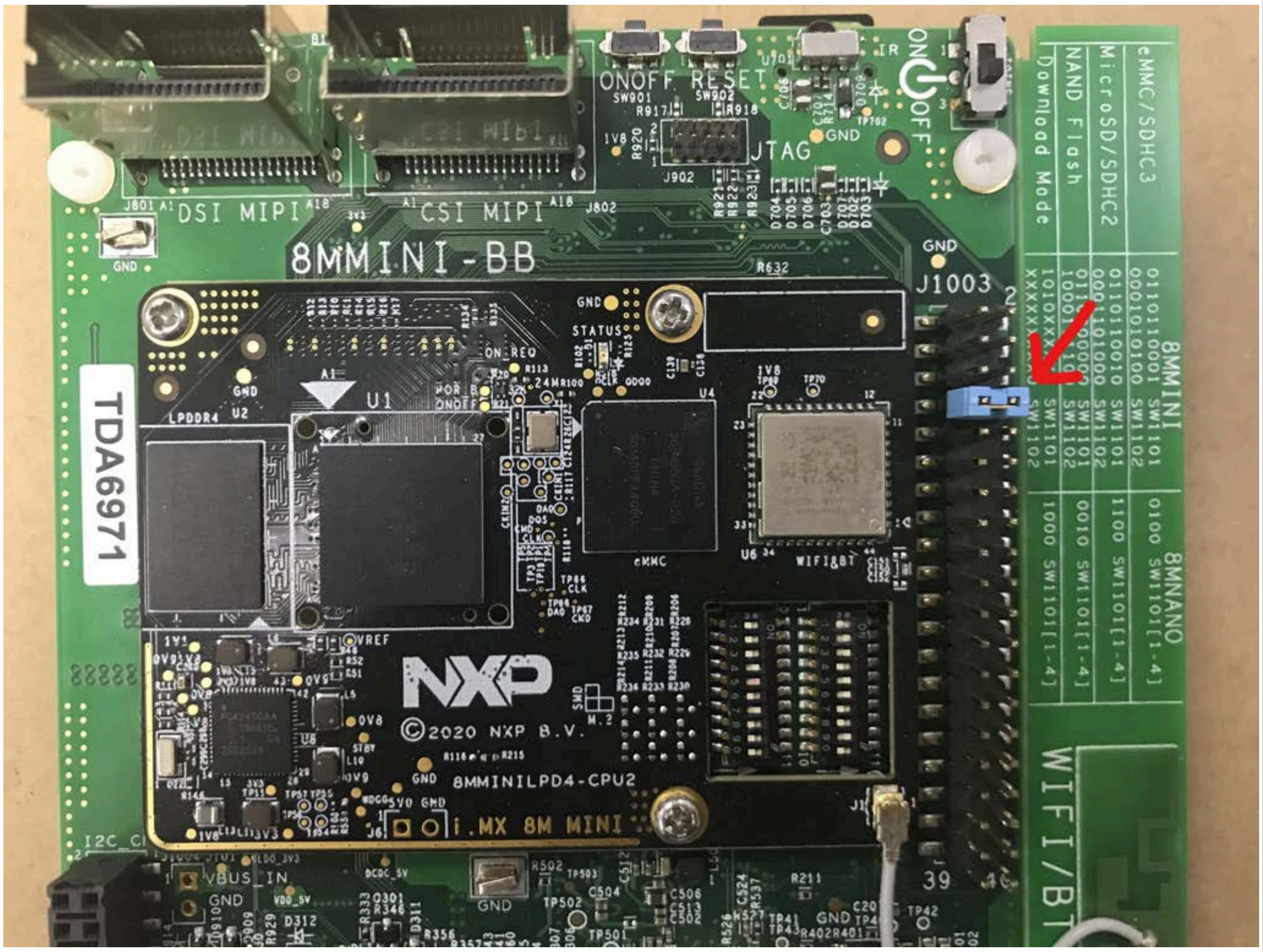


Figure 22. i.MX 8M Mini hardware interrupt routing to multiple cores

On LS1046ARDB, GPIO2_01 and GPIO2_02 are multiplexed with `SPI_CS_B[0]` and `SDHC_DAT[4]` signals. User must configure `RCW[382 ~ 383]` to `0b'10` to enable `GPIO2[0]` signal.

Since GPIO2 is assigned to the BareMetal core, Linux must not use it again. We can disable GPIO2 under Linux via `dts` file. To disable GPIO2, add the below code in the Linux kernel file `fsl-ls1046a-rdb-sdk-bm.dts`.

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

Table 18. GPIO_INT driver APIs and their description

Function declaration	Description
`int gpio_request_by_name(struct udevice *dev, const char *list_name, int index, struct gpio_desc *desc, int flags)`	Locate and request a GPIO by name `dev`- Device requesting the GPIO `index` - Index number of the GPIO in that list use request (0 = first) `desc` - Returns GPIO description information `flags` - Indicates the GPIO input/output settings Returns: `0` if OK, `-ve` on error

Table 18. GPIO_INT driver APIs and their description*...continued*

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

Under BareMetal, `gpio_interrupt` command provides “*`enable`*”, “*`start`*” and “*`stop`*” commands to control these two pins.

- **`gpio_interrupt`** `enable`: This command initializes the `gpio_int` driver and enables the interrupt.
- **`gpio_interrupt`** `start`: This command sets GPIO2_01 to high.
- **`gpio_interrupt`** `stop`: This command sets GPIO2_01 to low.

`gpio_interrupt` can run under the BareMetal console.

The `gpio_interrupt enable` command must be run first to initialize two pins. Then, use the command pair `gpio_interrupt start` and `gpio_interrupt stop` to pull high and pull down GPIO2_01/GPIO5_07. After the command pair, GPIO2_02/GPIO5_08 triggers an interrupt when getting the pull-down signal. The core1 sends the SGI interrupt to other secondary cores. The time of GPIO interrupt and SGI interrupt is dumped by each secondary 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 secondary cores can be triggered within 1 µs.

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

2.5.4.18 Hardware resource allocation

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

2.5.4.18.1 LS1028ARDB board

This section describes the hardware resource allocation for LS1028A reference design board for ENETC, I2C, and SAI configuration settings.

2.5.4.18.1.1 ENETC

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

See the following:

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

2.5.4.18.1.2 I2C

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

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

Note:

Operate the I2C devices in Baremetal side CAREFULLY.

```
#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 secondary 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..... .......
```

2.5.4.18.1.3 SAI

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

See the following:

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

2.5.4.18.2 LS1043ARDB or LS1046ARDB board

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

2.5.4.18.2.1 Linux DTS

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

2.5.4.18.2.2 Memory configuration

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

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

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

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

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

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

The memory configuration for bare metal is shown in the figure below.

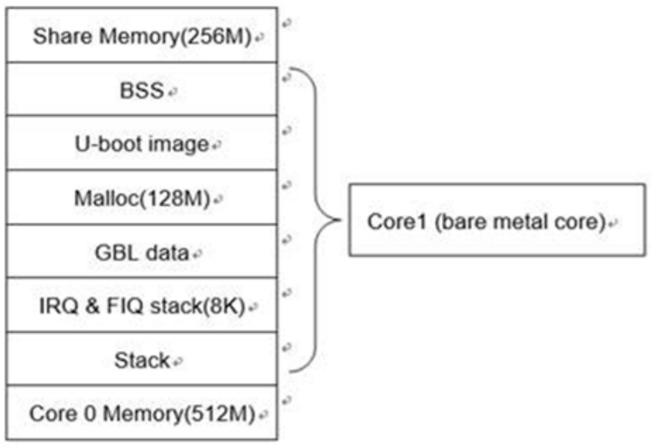


Figure 23. Memory configuration for LS1043ARDB or LS1046ARDB

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 defconfig of the board to change the maximum size of malloc.

Table 19. Memory APIs description

API name (type)	Description
void_t* malloc (size_t n)	Allocates memory

Table 19. Memory APIs description*...continued*

API name (type)	Description
	• “n” – length of allocated chunk • Returns a pointer to the newly allocated chunk
void free (void *ptr)	Releases the chunk of memory pointed to by ptr (where “ptr” is a pointer to the chunk of memory)

The GPIO for LS1043ARDB (or LS1046ARDB) has four GPIO controllers. You must 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`.

2.5.4.18.2.3 GPIO

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

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

2.5.4.18.2.4 I2C

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

The LS1043ARDB (or LS1028ARDB / LS1046ARDB) has four I2C controllers. You can configure the I2C bus using the `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 secondary core that runs the I2C bus.

2.5.4.18.2.5 Hardware interrupts

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

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

2.5.4.18.2.6 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 secondary core that runs this QSPI.

2.5.4.18.2.7 IFC

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

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

   ```
   &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
   ```

2.5.4.18.2.8 Ethernet

This section describes the Ethernet configuration settings for LS1043A or LS1046A reference design boards.

LS1043A or LS1046A has only one FMan. Therefore, you must remove the DPAA driver in Linux.

1. Disable the DPAA driver in the 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`.

2.5.4.18.2.9 USB

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

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

```

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

```
(3)     USB Controller numbers
```

2.5.4.18.2.10 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
```

2.5.4.18.3 LX2160ARDB board

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

2.5.4.18.3.1 Memory configuration

This section describes the memory configuration for LX2160ARDB boards.

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

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

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

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

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

Table 20. Memory API 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
void free (void *ptr)	Releases the chunk of memory pointed to by ptr (where "ptr" is a pointer to the chunk of memory)

2.5.4.18.4 i.MX 8M Mini LPDDR4 EVK and i.MX 8M Plus LPDDR4 EVK boards

The following sections describe the hardware resource allocation for the i.MX 8M Mini LPDDR4 EVK and i.MX 8M Plus LPDDR4 EVK boards for implementing the supported features.

2.5.4.18.4.1 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";
        };
```

2.5.4.18.4.2 Memory configuration

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

1. The boards have a 6 GB DDR memory. To use the BareMetal framework, configure the DDR into five partitions:

• 6016M for core0 (Linux)

• 32M for core1 (BareMetal)

• 32M for core2 (BareMetal)

• 32M for core3 (BareMetal)

• 32M for shared memory.

The configuration can be defined in the file `include/configs/imx8mm_baremetal.h.` or `include/configs/imx8mp_baremetal.h.`

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

2. Memory Reserve

For IPI data transfer, BareMetal shares memory between the primary core and the secondary core. Therefore, users must reserve some memory from the 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>; }; };
```

2.5.4.18.4.3 GPIO

1. Connect pin7 and pin8 of J1003. The `test_gpio` case in BareMetal uses pin7 and pin8 of J1003. Therefore, connect these two pins.

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

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

3. Disable the devices from the 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. Therefore, you must disable GPIO5 and UART3 from the kernel.

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

2.5.4.18.4.4 Ethernet

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

1. Disable the Ethernet card from dts files:

```
&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.

2.6 Native RTOS on Cortex-A core

Native RTOS refer to the RTOS running without using Hypervisor (Jailhouse), it is kicked to specified Cortex-A Core by U-Boot commands or remoteproc under Linux.

2.6.1 Overview

Real-time Edge system supports Native RTOS inlcuding FreeRTOS and Zephyr running on Cortex-A cores.

Currently there are two modes in which RTOS can run on Cortex-A Cores:

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

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

Native RTOS runs on Cortex-A Core directly similar to how the Linux kernel runs, without any Hypervisor. Therefore, native RTOS it has a better real-time performance in comparison with RTOS in Jailhouse inmate. Native RTOS is targeted for high real-time performance use cases with less real-time latency.

To run Native RTOS on Cortex-A Core, hardware resources used by each OS must be defined to ensure that there is no resource conflict between different operating systems. If Linux is used with RTOS simultaneously, the device nodes for peripherals used by RTOS should be disabled or removed from the Linux device tree.

2.6.2 Building Native RTOS on Cortex-A core

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

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

2.6.3 Booting Native RTOS on Cortex-A core

Cortex-A Core Native RTOS can be booted up or stopped on specified Cortex-A Core by using the U-Boot commands or remoteproc under Linux

Refer to Unified Life Cycle Management for details.

2.6.4 Debugging Native RTOS on Cortex-A Core

Some commercial debugging tools can be used to debug Native RTOS running ong Cortex-A Core, such as TRACE32 Development System which is a product from Lauterbach Datentechnik GmbH, and JLink which is product from SEGGER.

For using JLink debugger, refer to application note Debugging Cortex-A U-Boot and Native RTOS on i.MX 8M Plus and i.MX 93 EVKs which focuses on the NXP MPU Cortex-A Core debugging method and flow. It provides two methods, one is to use the command-line debugging tool (GDB), the other one is to use the GUI debugging tool (Eclipse in this AN), and takes U-Boot and RTOS debugging as examples.

2.6.5 Native RTOS real-time latency benchmark

2.6.5.1 rt_latency introduction

`rt_latency` is a real-time latency test application developed by NXP to measure the interrupt and scheduling latency of Jailhouse + RTOS under different workloads. It measures the latency (time delta, in nanoseconds) between hardware IRQ events and software actions.

A sample illustration is provided in the Figure 24.

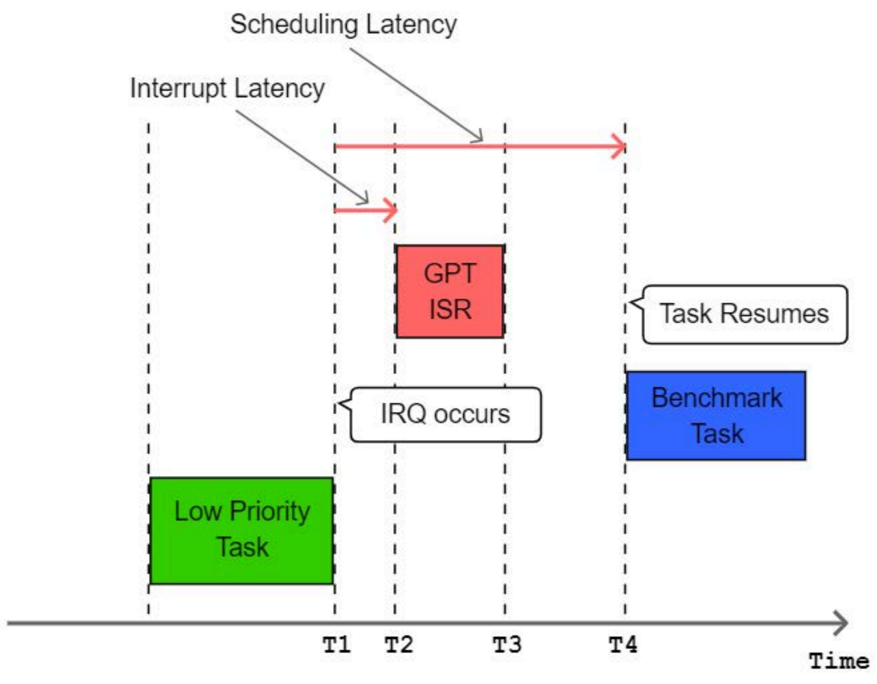


Figure 24. Interrupt latency and scheduling latency illustration

irq delay:

The time to enter in the software IRQ handler after a hardware IRQ occurs (hardware + hypervisor + IRQ vector latency). It is equal to T2 - T1 in the above graph.

irq to sched:

The time to enter in an RTOS task, scheduled by the IRQ handler (irq delay + RTOS scheduler). It is equal to T4 - T1 in the above graph.

All measurements are done using hardware timer and relative to the hardware IRQ event time, with submicrosecond precision. The benchmark task sets the timeout value to 20 ms for hardware timer interrupt. When running, the `rt_latency` application displays latency statistics every 10 seconds, based on the measurements taken, to help characterize the system real-time latency.

`rt_latency` application provides different latency test cases. For example, it is possible to engage some CPU load and/or IRQ load to measure their impact on the latency. The Table 21 shows the six cases that are currently implemented in the `rt_latency` application. Each case has a corresponding case ID and can be selected at running time.

Table 21. rt_latency application use cases

Test Case ID	Case Description
1	No extra load
2	Extra CPU load (low-priority task, executing busy loop and consuming all available CPU time)
3	Extra IRQ load
4	Extra CPU load + semaphore load
5	Extra CPU load + Linux load (not provided by the rt_latency RTOS application)
6	Extra CPU load + cache flush (instruction cache only for this release)

FreeRTOS related information of rt_latency

FreeRTOS is customized using a configuration file named `FreeRTOSConfig.h`. In the **rt_latency FreeRTOS** project, the macro `configMAX_PRIORITIES` is defined to 5 in `FreeRTOSConfig.h`. It is the number of priorities available to the application tasks.

When rt_latency is started, it creates a task called "`main_task`" of priority 1. The task runs an endless loop to process the input of UART console for native RTOS or control application `harpoon_ctrl` for Jailhouse RTOS and then delay 100 ms using the `xTaskDelay()` FreeRTOS API function. This task always runs and never ends.

When a specified benchmark use case is selected to run, the rt_latency application creates a task called "benchmark_task" of priority 3 by the `xTaskCreate()` FreeRTOS API function to measure the latency. It also creates a task called “`log_task`” of priority 1 to display the latency statistics every 10 seconds.

Running benchmark cases that include extra load uses more system resources and creates more tasks. The details are shown in the Table 22.

Table 22. Running benchmark cases that include extra load

Case ID	Additional FreeRTOS tasks (except "`benchmark_task`" and “`log_task`”)
1	None
2	A task called "`cpu_load_task`" of priority 0, executing busy loop and consuming all available CPU time.
3	No additional task is created but additional IRQ load is added in the "`benchmark_task`". Note that the IRQ latency testing in "`benchmark_task`" uses GPT1 while the IRQ load uses GPT2.
4	A task called "`cpu_load_task`" of priority 0, executing busy loop and consuming all available CPU time plus obtaining/releasing a semaphore using the `xSemaphoreTake()`/`xSemaphoreGive()` FreeRTOS API functions in the loop.
5	A task called "`cpu_load_task`" of priority 0, executing busy loop and consuming all available CPU time. Linux load (that is, hackbench) has to be run separately by the user.
6	A task called "`cpu_load_task`" of priority 0, executing busy loop and consuming all available CPU time. A task called "`cache_inval_task`" of priority 1, executing a loop to execute cache invalidation instructions and then delay 100 ms using the `xTaskDelay()` FreeRTOS API function.

2.6.5.2 Building rt_latency benchmark application

RTOS latency benchmark application "rt_latency" runs on FreeRTOS or Zephyr, it is an application in the repo: heterogeneous-multicore.

Refer to "Building Heterogeneous Multicore RTOS Application" for how to build RPMSG performance evaluation application.

2.6.5.3 Running Native RTOS rt_latency application

Native RTOS rt_latency application can run on the following boards:

- i.MX 8M Plus EVK
- i.MX 8M Mini EVK
- i.MX 91 EVK and QSB
- i.MX 93 11x11 and 14x14 EVK
- i.MX 943 15x15 and 19x19 EVK
- i.MX 95 15x15 and 19x19 EVK

Test Environment

Requirements:

- i.MX 8M Plus EVK, i.MX 8M Mini EVK, i.MX 91 EVK and QSB, i.MX 93 11x11 and 14x14 EVK, i.MX 943 15x15 and 19x19 EVK or the i.MX 95 15x15 and 19x19 EVK board
- Real-time Edge Software v2.8 or above
- Host PC
- Console cables

Test Procedure

1. Set up the UART Console for Native RTOS
   Refer to the steps listed in *Set up the UART console* in the Section 3.8.1.
   The first UART console is used for Linux boots up, whereas the other one is used for RTOS running on Cortex-A Core.
2. Booting Native RTOS rt_latency Application
   After powering up the board and entering U-Boot command line, execute the following U-Boot commands to run the `rt_latency` application.
   - Boot the FreeRTOS `rt_latency` application on i.MX 8M Mini EVK board

     ```
     u-boot=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/rt-
     latency-freertos/rt_latency_ca53.bin
     u-boot=> dcache flush; icache flush; cpu 3 release 0x93C00000;
     ```
   - Boot the Zephyr `rt_latency` application on i.MX 8M Mini EVK board

     ```
     u-boot=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/rt-
     latency-zephyr/rt_latency_ca53.bin
     u-boot=> dcache flush; icache flush; cpu 3 release 0x93C00000;
     ```
   - Boot the FreeRTOS `rt_latency` application on i.MX 8M Plus EVK board:

     ```
     u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/rt-
     latency-freertos/rt_latency_ca53.bin
     u-boot=> dcache flush; icache flush; cpu 3 release 0xC0000000;
     ```
   - Boot Zephyr `rt_latency` application on i.MX 8M Plus EVK board:

     ```
     u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/rt-
     latency-zephyr/rt_latency_ca53.bin
     u-boot=> dcache flush; icache flush; cpu 3 release 0xC0000000;
     ```
   - Boot FreeRTOS `rt_latency` application on i.MX 91 EVK and QSB board:

     ```
     u-boot=> ext4load mmc 1:2 0x80000000 /examples/heterogeneous-multicore/rt-
     latency-freertos/rt_latency_ca55.bin
     u-boot=> dcache flush; icache flush; go 0x80000000;
     ```
   - Boot Zephyr `rt_latency` application on i.MX 91 EVK and QSB board:

     ```
     u-boot=> ext4load mmc 1:2 0x80000000 /examples/heterogeneous-multicore/rt-
     latency-zephyr/rt_latency_ca55.bin
     u-boot=> dcache flush; icache flush; go 0x80000000;
     ```
   - Boot FreeRTOS `rt_latency` application on i.MX 93 11x11 and 14x14 EVK board:

     ```
     u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/rt-
     latency-freertos/rt_latency_ca55.bin
     u-boot=> dcache flush; icache flush; cpu 1 release 0xD0000000;
     ```
   - Boot Zephyr `rt_latency` application on i.MX 93 11x11 and 14x14 EVK board:

     ```
     u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/rt-
     latency-zephyr/rt_latency_ca55.bin
     u-boot=> dcache flush; icache flush; cpu 1 release 0xD0000000;
     ```

- Boot FreeRTOS `rt_latency` application on i.MX 943 15x15 and 19x19 EVK board:

  ```
  u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/rt-
  latency-freertos/rt_latency_ca55.bin
  u-boot=> dcache flush; icache flush; cpu 3 release 0xD0000000;
  ```

- Boot Zephyr `rt_latency` application on i.MX 943 15x15 and 19x19 EVK board:

  ```
  u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/rt-
  latency-zephyr/rt_latency_ca55.bin
  u-boot=> dcache flush; icache flush; cpu 3 release 0xD0000000;
  ```

- Boot FreeRTOS `rt_latency` application on i.MX 95 15x15 and 19x19 EVK board:

  ```
  u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/rt-
  latency-freertos/rt_latency_ca55.bin
  u-boot=> dcache flush; icache flush; cpu 5 release 0xD0000000;
  ```

- Boot Zephyr `rt_latency` application on i.MX 95 15x15 and 19x19 EVK board:

  ```
  u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/rt-
  latency-zephyr/rt_latency_ca55.bin
  u-boot=> dcache flush; icache flush; cpu 5 release 0xD0000000;
  ```

3. On the RTOS serial console connected, the following output is displayed.
   For FreeRTOS:

   ```
   ****************************************
    FreeRTOS Latency Benchmark on Cortex-A
   ****************************************

   Test Case ID (1-6):
           1: no extra load
           2: extra CPU load
           3: extra IRQ load
           4: extra CPU load + semaphore load
           5: extra CPU load + Linux load (not provided by the test case)
           6: extra CPU load + cache flush
   Please input the test case ID (1-6), 'q' for quit:
   ```

   For Zephyr:

   ```
   *** Booting Zephyr OS build v4.1.0-39-gb2ab2031bad8 ***

   **************************************
   Zephyr Latency Benchmark on Cortex-A
   **************************************

   Test Case ID (1-6):
           1: no extra load
           2: extra CPU load
           3: extra IRQ load
           4: extra CPU load + semaphore load
           5: extra CPU load + Linux load (not provided by the test case)
           6: extra CPU load + cache flush
   Please input the test case ID (1-6), 'q' for quit:
   ```

4. Add Extra Linux CPU Load (Only feasible for test case 5 and 6)
   For test case 5 and 6, you must add extra load in Linux. Therefore, boot into Linux by running the following command in the U-Boot command line:

Note: *i.MX 91 implements only one Cortex-A55 core, so the extra load is not applicable.*

- For i.MX 8M Mini EVK:

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

- For i.MX 8M Plus EVK:

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

- For i.MX 93 11x11 and 14x14 EVK:

```
# For i.MX 93 11x11 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rtos.dtb
# For i.MX 93 14x14 EVK
u-boot=> setenv fdtfile imx93-14x14-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

- For i.MX 943 15x15 and 19x19 EVK:

```
u-boot=> setenv fdtfile imx943-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

- For i.MX 95 15x15 and 19x19 EVK:

```
# For i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rtos.dtb
# For i.MX 95 19x19 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

Once the Linux system has booted up, optimize the system configuration to achieve maximum performance:

```
$ real_time_setup.sh
```

Then, run the following command to add extra Linux load:

```
$ dd if=/dev/urandom of=/dev/null &>/dev/null & disown;
$ dd if=/dev/urandom of=/dev/null &>/dev/null & disown;
$ dd if=/dev/urandom of=/dev/null &>/dev/null & disown;
```

5. Select test case ID
   When the user inputs a test case ID (1 ~ 6) on the RTOS console, the benchmarking process starts. Then, the latency statistics is displayed on the RTOS console.
6. Input the testing duration for the selected case

```
Test Case ID (1-6):
        1: no extra load
        2: extra CPU load
        3: extra IRQ load
        4: extra CPU load + semaphore load
        5: extra CPU load + Linux load (not provided by the test case)
        6: extra CPU load + cache flush
Please input the test case ID (1-6), 'q' for quit: 1
Enter test duration ('0' for infinite loop):
  - number for seconds (e.g., '10')
  - number followed by 'h' for hours (e.g., '2h')
  - number followed by 'd' for days (e.g., '1d')
>
```

7. Capture the test log
   Save the logs displayed by the rt_latency application in the RTOS console to a file (that is, `rt_latency.log`) on host PC for further analysis. Below is a snippet of an example log:

```
Test duration: 1 hours (3600 seconds)

INFO: start_test_case       : Running test case 1:
INFO: benchmark_task        : running
INFO: stats_print           : stats(D0601B30) irq delay (ns) min 250 mean 306
 max 416 rms^2 94616 stddev^2 747 absmin 250 absmax 416
INFO: hist_print            : n_slot 21 slot_size 1000
INFO: hist_print            : 100000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0
INFO: stats_print           : stats(D0601F90) irq to sched (ns) min 1500 mean
 1596 max 2583 rms^2 2550950 stddev^2 798 absmin 1500 absmax 2583
INFO: hist_print            : n_slot 21 slot_size 1000
INFO: hist_print            : 0 99999 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INFO: print_stats           : late alarm scheduling: 0
INFO: print_stats           :
INFO: stats_print           : stats(D0601B30) irq delay (ns) min 250 mean 310
 max 333 rms^2 97174 stddev^2 681 absmin 250 absmax 416
INFO: hist_print            : n_slot 21 slot_size 1000
INFO: hist_print            : 200000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0
INFO: stats_print           : stats(D0601F90) irq to sched (ns) min 1500 mean
 1601 max 1625 rms^2 2565317 stddev^2 695 absmin 1500 absmax 2583
INFO: hist_print            : n_slot 21 slot_size 1000
INFO: hist_print            : 0 199999 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0
INFO: print_stats           : late alarm scheduling: 0
INFO: print_stats           :
INFO: stats_print           : stats(D0601B30) irq delay (ns) min 250 mean 310
 max 333 rms^2 97001 stddev^2 689 absmin 250 absmax 416
INFO: hist_print            : n_slot 21 slot_size 1000
INFO: hist_print            : 300000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0
INFO: stats_print           : stats(D0601F90) irq to sched (ns) min 1541 mean
 1601 max 1625 rms^2 2564434 stddev^2 704 absmin 1500 absmax 2583
INFO: hist_print            : n_slot 21 slot_size 1000
INFO: hist_print            : 0 299999 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0
INFO: print_stats           : late alarm scheduling: 0
INFO: print_stats           :
…… (omitted)
```

2.6.5.4 Output of rt_latency

The following latency statistics are displayed after every 10 seconds. The first three lines are for "`irq delay`" and the other three lines are for "`irq to sched`".

```
INFO: stats_print           : stats(C0601260) irq delay (ns) min 708 mean 711
 max 833 rms^2 506896 stddev^2 174 absmin 708 absmax 3083
            INFO: hist_print            : n_slot 21 slot_size 200
            INFO: hist_print            : 0 0 0 999 2 0 0 0 0 0 0 0 0 0 1 1 0 0
 0 0 0 0
```

```
           INFO: stats_print           : stats(C06016C0) irq to sched (ns) min
 2416 mean 2499 max 5333 rms^2 6341435 stddev^2 92571 absmin 2416 absmax 5625
           INFO: hist_print            : n_slot 21 slot_size 1000
           INFO: hist_print            : 0 0 986 0 11 7 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0
           INFO: print_stats           :
```

As stated before, the benchmark task sets the timeout value to 20 ms for GPT interrupt. This means that one sample of "`irq delay`" latency data and one sample of "`irq to sched`" latency data are measured about every 20 ms. So a 10 s time would have approximately 500 samples.

1. **min/mean/max**: minimum, average and maximum latency value measured within the last 10 s period of time. These statistics are collected and calculated for approximately 500 samples.
2. **rms^2**: square of the RMS (Root Mean Square) measured within the last 10 s period of time.
3. **stddev^2**: square of the standard deviation measured within the last 10 s period of time. The square of the standard deviation is also called "variance".
4. **absmin/absmax**: minimum and maximum latency value measured since the beginning of the test. Therefore, we take the value of `absmin` and `absmax` in the last set of latency statistics as the benchmarking results.
5. A histogram is also shown to give an idea of repartition of the measured latency values. Note that the histogram data is accumulated so the sum of one line of histogram data increases by about 500 compared to the previous one. Therefore, we only care about the histogram data in the last set of latency statistics. That is the repetition of the measured latency values over the whole test period (that is, 12h).

2.7 RTOS on Cortex-A core with Jailhouse

The following sections describe the steps for running RTOS on Cortex A core with Jailhouse.

2.7.1 Jailhouse

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

2.7.1.1 Overview

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, which means 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 resources and assign them to other cells.

2.7.1.2 Running PREEMPT_RT Linux in Inmate

The below sections describe running PREEMPT_RT Linux in Inmate using the i.MX 8M Plus, LS1028ARDB, and LS1046ARDB platforms.

2.7.1.2.1 i.MX 8M Plus LPDDR4 EVK

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

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

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

4. Check the output on the terminal of UART4:

```
[    0.717545] printk: console [ttymxc3] enabled
[    0.721628] printk: bootconsole [ec_imx6q0] disabled
[    0.732428] loop: module loaded
[    0.732902] of_reserved_mem_lookup() returned NULL
[    0.732952] megasas: 07.714.04.00-rc1
[    0.733632] imx ahci driver is registered.
[    0.735615] tun: Universal TUN/TAP device driver, 1.6
[    0.735835] thunder_xcv, ver 1.0
[    0.735863] thunder_bgx, ver 1.0
[    0.735889] nicpf, ver 1.0
[    0.736340] hclge is initializing
[    0.736351] hns3: Hisilicon Ethernet Network Driver for Hip08 Family -
 version
[    0.736354] hns3: Copyright (c) 2017 Huawei Corporation.
[    0.736382] e1000: Intel(R) PRO/1000 Network Driver
[    0.736384] e1000: Copyright (c) 1999-2006 Intel Corporation.
[    0.736416] e1000e: Intel(R) PRO/1000 Network Driver
[    0.736418] e1000e: Copyright(c) 1999 - 2015 Intel Corporation.
[    0.736447] igb: Intel(R) Gigabit Ethernet Network Driver
[    0.736450] igb: Copyright (c) 2007-2014 Intel Corporation.
[    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.

2.7.1.2.2 LS1028ARDB

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

2.7.1.2.2.1 Linux in non-root cell

Perform the following steps to run PREEMPT_RT Linux in Inmate on LS1028ARDB platform:

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

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

4. Exit Jailhouse.

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

2.7.1.2.2.2 ENETC in non-root cell

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

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

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

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

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

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

4. Exit Jailhouse.

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

Note:

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

2.7.1.2.2.3 GPIO in non-root cell

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

1. Hardware setup
   Connect J11 Pin5 (1588_ALARM_OUT1/GPIO3_DAT11) to Pin 8 (1588_CLK_OUT/GPIO3_DAT10)
2. RCW setting
   In `dash-rcw/ls1028ardb/R_SQPP_0x85bb/rcw_1500_sdboot.rcw`, change as below:

   ```
   EC1_SAI4_5_PMUX=1
   EC1_SAI3_6_PMUX=1
   ```

   `EC1_SAI4_5_PMUX` is set to `0b001`, `EC1_SAI3_6_PMUX` is set to `0b001` to select GPIO.
3. Software configuration required:
   a. Configure CPLD register BRDCFG3 (offset 053h) bit 2 to 0 (IEEE signals connect to the IEEE header) in U-Boot prompt:

   ```
   => i2c mw 66 53 00
   ```

   b. Boot up Linux using Jailhouse DTB and bring up non-root Linux:

   ```
   => run jh_mmcboot
   ```

   c. Wait for Linux OS to boot up and login.
   d. Execute non-root Linux demo.

   ```
   # cd /usr/share/jailhouse/scripts
   # ./linux-demo-ls1028ardb.sh
   ```
4. Test GPIO function in non-root Linux.
   a. Export GPIO pin

   ```
   # ls /sys/class/gpio
   # echo 490 > /sys/class/gpio/export
   # echo 491 > /sys/class/gpio/export
   ```

   b. Configure GPIO output and input.

   ```
   # echo out > /sys/class/gpio/gpio490/direction
   # cat /sys/class/gpio/gpio490/direction
   # cat /sys/class/gpio/gpio491/direction
   ```

   c. Verify write 1 to GPIO 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
```

2.7.1.2.3 LS1046ARDB

Perform the following steps:

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

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

4. Exit Jailhouse:

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

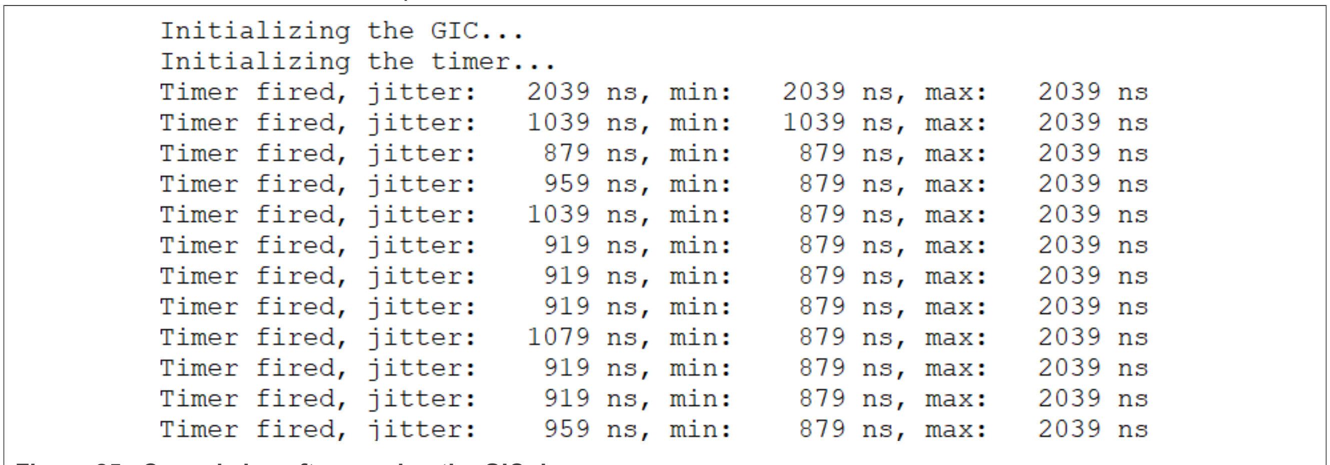
2.7.1.3 Running Jailhouse examples In Inmate

2.7.1.3.1 i.MX 8M Plus LPDDR4 EVK

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

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

4. Check the result on the serial port:



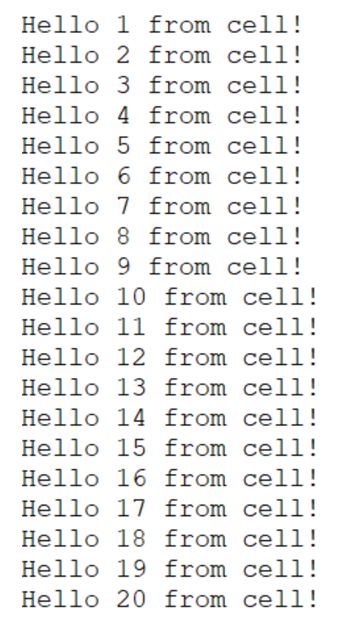
```
        Initializing the GIC...
        Initializing the timer...
        Timer fired, jitter:   2039 ns, min:   2039 ns, max:   2039 ns
        Timer fired, jitter:   1039 ns, min:   1039 ns, max:   2039 ns
        Timer fired, jitter:    879 ns, min:    879 ns, max:   2039 ns
        Timer fired,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

Figure 25. Console log after running the GIC demo

5. Execute the 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!
```

Figure 26.

7. Exit Jailhouse.

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

2.7.1.3.2 LS1028ARDB Jailhouse example in Inmate

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

1. Execute `run jh_mmcboot` in U-Boot stage.
2. Wait for the 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 the ivshmem demo using the command below:

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

Note: *If the 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.

2.7.1.3.3 LS1046ARDB Jailhouse example

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

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

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

4. Execute UART demo:

```
# ./uart-demo-ls1046ardb.sh
```

5. Execute ivshmem demo:

```
# ./ivshmem-demo-ls1046ardb.sh
```

6. Exit Jailhouse.

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

2.7.2 Harpoon (RTOS on Cortex-A)

2.7.2.1 Overview

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

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

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

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

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

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

2.8 RTOS and Baremetal on Cortex-M core

Regarding RTOS and Baremetal building, refer to Building, deploying, and releasing unified software.

Real-time Edge images have some demos for testing.

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

2.8.1 Booting Cortex-M Core RTOS Image

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

2.8.1.1 Using U-Boot Commands to Manage Life Cycle of RTOS on Cortex-M Core

U-boot command "`bootaux`" is used to boot Cortex-M Core RTOS Image from U-Boot, for example, after the board is booted into the U-Boot console.

- Use the following command to boot Arm Cortex-M core on i.MX 8M Mini LPDDR4 EVK board, it uses UART4:

```
=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm4.bin; cp.b 0x48000000 0x7e0000 20000;
=> bootaux 0x7e0000
```

- Use the following command on i.MX 8M Plus LPDDR4 EVK board, it uses UART4:

```
=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm7.bin; cp.b 0x48000000 0x7e0000 20000;
=> bootaux 0x7e0000
```

- Use the following command on i.MX 93 EVK board, it uses UART2:

```
=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm33.bin; cp.b 0xd0000000 0x201e0000 20000;
=> bootaux 0x1ffe0000
```

- Use the following command on i.MX 95 15x15 EVK and i.MX95 19x19 EVK board, it uses UART3:

```
=> ext4load mmc 1:2 0x90000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm7.bin;
=> cp.b 0x90000000 0x203c0000 ${filesize}
=> bootaux 0 1
```

- Then, the below RTOS log is displayed on the UART console (taking i.MX 8M Mini EVK as an example)

```
Cortex-M4: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M4 core0 (MPID: 0x0)
```

2.8.1.2 Using RemoteProc under Linux to Manage Life Cycle of RTOS on Cortex-M Core

Use Linux remoteproc to start and stop RTOS on Cortex-M Core, taking the Heterogeneous Multicore `hello_world` application as an example:

- Booting Arm Cortex-M core on i.MX 8M Mini EVK, i.MX 8M Plus EVK, i.MX 93 EVK, i.MX 95 15x15 EVK and i.MX 95 19x19 EVK boards:
  1. Starting Linux kernel:

```
=> run prepare_mcore
=> setenv fdtfile imx8mm-evk-multicore-rtos.dtb # for i.MX 8M Mini EVK
=> setenv fdtfile imx8mp-evk-multicore-rtos.dtb # for i.MX 8M Plus EVK
=> setenv fdtfile imx93-11x11-evk-multicore-rtos.dtb # for i.MX 93 EVK
=> setenv fdtfile imx95-15x15-evk-multicore-rtos.dtb; # for i.MX 95 15x15
 EVK
=> setenv fdtfile imx95-19x19-evk-multicore-rtos.dtb; # for i.MX 95 19x19
 EVK
=> boot
```

  2. After Linux booting up, boot up Cortex-M core RTOS application:

```
# i.MX 8M Mini EVK uses UART4
root@imx8mm-lpddr4-evk:~# echo -n /examples/heterogeneous-multicore/hello-
world-freertos/hello_world_cm4.elf > /sys/devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/state

# i.MX 8M Plus EVK uses UART4
root@imx8mp-lpddr4-evk:~# echo -n /examples/heterogeneous-multicore/hello-
world-freertos/hello_world_cm7.elf > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/state

# i.MX 93 EVK uses UART2
root@imx93evk:~# echo -n /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm33.elf > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/state

# i.MX 95 15x15 EVK and i.MX 95 19x19 EVK use UART3
```

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_cm7.elf > /sys/devices/platform/imx95-cm7/
remoteproc/remoteproc7/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/imx95-cm7/
remoteproc/remoteproc7/state
```

Then, the below RTOS log is displayed on the UART console (taking i.MX 8M Mini EVK as an example):

```
Cortex-M4: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M4 core0 (MPID: 0x0)
```

3. Then the Cortex-M core RTOS application can be shut down:

```
# i.MX 8M Mini EVK
root@imx8mm-lpddr4-evk:~# echo stop > /sys/devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/state

# i.MX 8M Plus EVK
root@imx8mp-lpddr4-evk:~# echo stop > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/state

# i.MX 93 EVK
root@imx93evk:~# echo stop > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/state

# i.MX 95 15x15 EVK and i.MX 95 19x19 EVK
root@imx95-19x19-lpddr5-evk:~# echo stop > /sys/devices/platform/imx95-cm7/
remoteproc/remoteproc7/state
```

Note: *The remoteproc portal instance ID `/sys/devices/platform/remoteproc-cm7/remoteproc/remoteproc4` might be different, depending on the device tree configuration. Therefore, use the `name` under each remoteproc instance portal to check which instance must be used.*

3 Heterogeneous Multicore Framework

Heterogeneous Multicore Framework provides a general software framework to support Heterogeneous AMP. It enables AMP to be interconnected and provides a unified resource management and life-cycle management as shown in Figure 27.

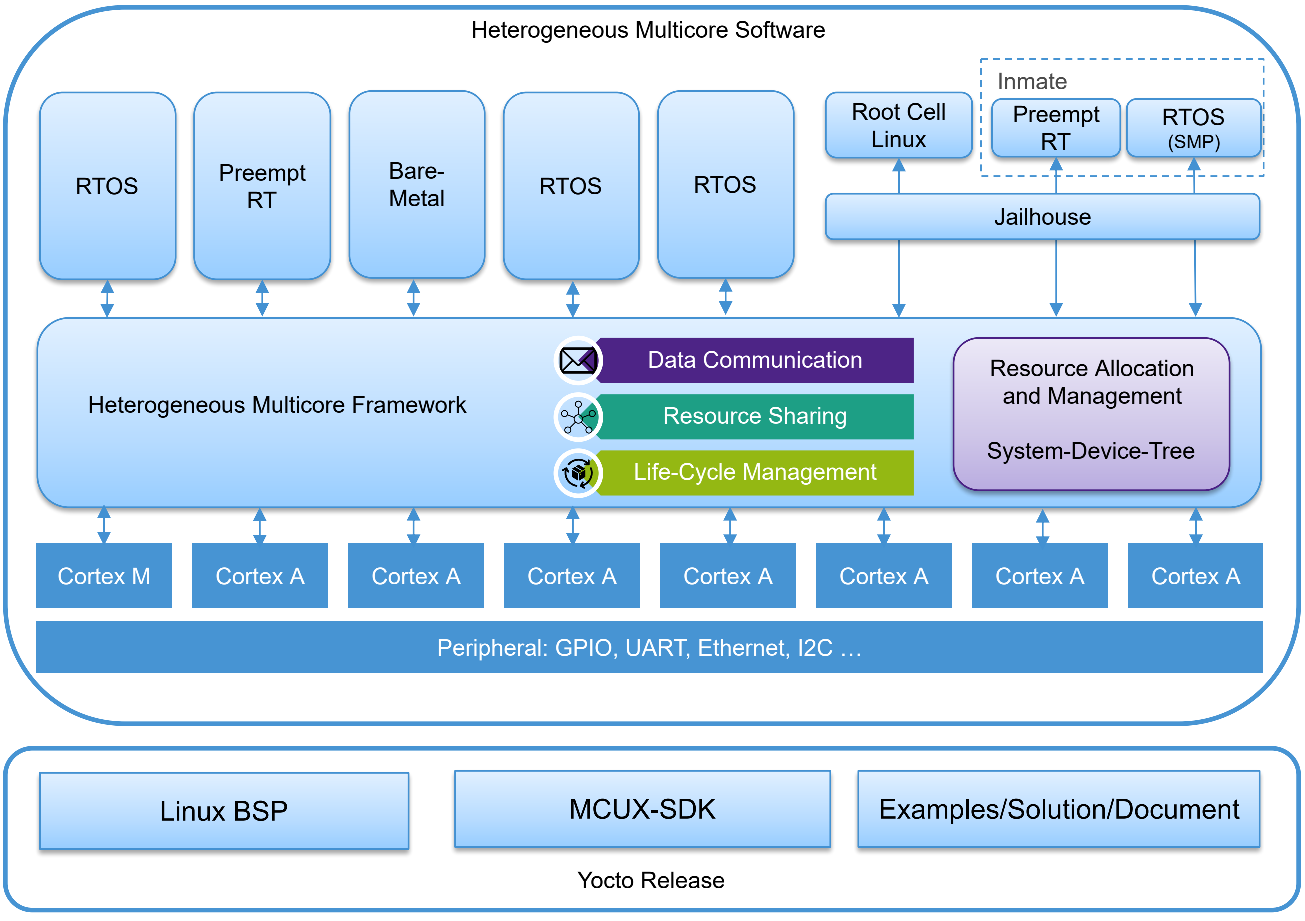


Figure 27. Heterogeneous multicore framework architecture

3.1 Overview

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

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

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

- RPMsg
  Use RPMsg to build control and data path crossing OS, physical resource is shared with another OS in terms of virtual device. The simplified Real-Time Messaging (SRTM) protocol provided in Real-time Edge is an implementation based on RPMsg. It is used to share the physical resources of the Cortex-M core with the Cortex-A core in terms of virtual devices in Linux.
- Heterogeneous Multicore VirtIO
  Heterogeneous Multicore VirtIO have a better performance than RPMSG, and it can also be used for resource sharing. POSIX compatible API can be used to access virtual device, and some existing VirtIO device drivers in Linux can be reused. Networking sharing is provided in Real-time Edge to share the same networking interface between multiple OSes.

3. **Unified Life-Cycle Management**
   Heterogeneous Multicore Framework provides unified Life Cycle Management both for Cortex-A Core and Cortex-M Core. It supports using U-Boot commands or Linux remoteproc to start or stop RTOS.
4. **Industrial Applications**
   The Heterogeneous Multicore framework provides a flexible mechanism to run Preemp-RT Linux or RTOS on different cores with Industrial applications.

There are some sample applications provided in Heterogeneous Multicore Framework. These applications can be used to demo and evaluate the features in Heterogeneous Multicore Framework:

1. `hello_world`
   The `hello_world` application demonstrates a sample flexible Real-time System on MPU platforms. The multiple images provided can be used to run single or multiple RTOS on Cortex-A Core or Cortex-M with or without running Linux simultaneously.
2. RPMSG Applications
   Heterogeneous Multicore Framework supports RPMSG communication between any Real-time Systems on MPU Platforms, such as:
   - RPMSG between RTOS on Cortex-M Core and Linux on Cortex-A core
   - RPMSG between RTOS on Cortex-A Core and Linux on Cortex-A Core
   - RPMSG between RTOS on Cortex-A Core and RTOS on Cortex-A Core
   - RPMSG between RTOS on Cortex-M Core and RTOS on Cortex-A Core

   The following applications provide filed trail for RPMSG-related features.
   - `rpmsg_str_echo`
     This demo demonstrates building up multiple RPMSG endpoints between RTOS and Linux. For example, on the i.MX 8M Plus EVK board, images provided in Real-time Edge can be used to run three RTOS `rpmsg_str_echo` applications on two Cortex-A Core and one Cortex-M Core. The other two Cortex-A Cores run SMP Linux, then each RTOS establishes three RPMSG channels with Linux.
   - `rpmsg_pingpong`
     This demo demonstrates RPMSG communication between RTOS and RTOS, one is RPMSG MainDevice and the other is RPMSG remote.
   - `rpmsg_perf`
     `rpmsg_perf` is a tool to evaluate RPMSG bandwidth performance between RTOS and Linux Kernel.
   - RPMSG enhanced 8MB buffer
     The application demonstrates how to change the default RPMSG buffer size and count.
   - UART sharing based on RPMSG

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

3. Heterogeneous Multicore VirtIO Applications
   - `virtio_perf`
     `virtio_perf` is a tool that evaluates Heterogeneous Multicore VirtIO bandwidth performance between RTOS and Linux.
   - VirtIO Networking Sharing
     This application demonstrates how to use a Heterogeneous Multicore VirtIO to share physical peripherals or other resources between different CPU Cores or the OSes. The applications provide networking sharing, where the physical networking interface is controlled by Cortex-M Core or Cortex-A Core. Then, the information about which virtual NIC device driver is provided in Linux is shared with the Cortex-A Core. Heterogeneous Multicore VirtIO is used to establish a high performance data path between two sides.
4. SOEM Applications
   - `soem_digital_io`
     SOEM digital_io example for IO control using BECKHOFF EK1100 as EtherCAT SubDevice.
   - `soem_servo`
     SOEM servo_motor example for motor control using Inovance SV680 servo as EtherCAT SubDevice.
   - `soem_servo_rt1180`
     SOEM servo_motor_rt1180 example for motor control using i.MX RT1180 EVK board as EtherCAT SubDevice.

Table 23 shows the support matrix on NXP platforms:

Table 23. Heterogeneous Multicore Application Support Matrix

Heterogeneous Multicore Framework				i.MX 8M Mini LPDDR4 EVK		i.MX 8M Plus LPDDR4 EVK		i.MX 91 QSB	i.MX 91 EVK	i.MX 93 9x9 LPDDR4 QSB		i.MX 93 EVK		i.MX 93 14x14 EVK		i.MX 943 15x15/19x19 EVK			i.MX 95 15x15 EVK		i.MX 95 19x19 EVK	
Feature	**Sub-feature**	**RTOS**	**Application**	**4 X A53**	**1 X M4**	**4 X A53**	**1 X M7**	**1 X A55**	**1 X A55**	**2 X A55**	**1 X M33**	**2 X A55**	**1 X M33**	**2 X A55**	**1 X M33**	**4 X A55**	**1 X M33**	**2 X M7**	**6 X A55**	**1 X M7**	**6 X A55**	**1 X M7**
Flexible Real-time System	Flexible Real-time System	Free RTOS	hello_world	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y	Y			Y	Y	Y	Y
		Zephyr	hello_world	Y		Y		Y	Y			Y	Y[1]	Y	Y[1]	Y			Y	Y[1]	Y	Y[1]
	RAM Console	Free RTOS	hello_world	Y		Y						Y		Y		Y			Y		Y	
		Zephyr	hello_world	Y		Y						Y		Y		Y			Y		Y	
	SMP RTOS	Zephyr	hello_world			Y						Y		Y					Y		Y	
	networking stack on A-Core RTOS	Free RTOS	lwip_ping	Y		Y						Y		Y								
Unified Life Cycle Management	U-Boot commands booting/ stopping A-Core Native RTOS	Free RTOS	all	Y		Y		Y	Y			Y		Y		Y			Y		Y	
		Zephyr	all	Y		Y		Y	Y			Y		Y		Y			Y		Y	
	Linux remoteproc booting/ stopping A-Core Native RTOS	Free RTOS	all	Y		Y						Y		Y					Y		Y	
		Zephyr	all	Y		Y						Y		Y					Y		Y	
	U-Boot commands booting M-Core Native RTOS	Free RTOS	all		Y		Y				Y		Y		Y					Y		Y
		Zephyr	all										Y		Y					Y		Y
	Linux remoteproc booting M-Core Native RTOS	Free RTOS	all		Y		Y				Y		Y		Y					Y		Y
		Zephyr	all										Y		Y					Y		Y
RPMSG	RPMSG between Linux and M-Core RTOS	Free RTOS	rpmsg_str_echo		Y		Y				Y[2]		Y		Y							
		Zephyr	rpmsg_str_echo										Y		Y					Y		Y
	RPMSG between Linux and A-Core RTOS	Free RTOS	rpmsg_str_echo	Y		Y						Y		Y					Y		Y	
		Zephyr	rpmsg_str_echo			Y						Y		Y					Y		Y	
	RPMSG between 2 A-Core RTOS	Free RTOS	rpmsg_pingpong			Y						Y		Y					Y		Y	
	RPMSG between Linux and M-Core RTOS with enhanced 8MB buffer	Free RTOS	rpmsg_str_echo		Y																	
	RPMsg Performance Evaluation	Free RTOS	rpmsg_perf			Y	Y															

Table 23. Heterogeneous Multicore Application Support Matrix*...continued*

Heterogeneous Multicore Framework				i.MX 8M Mini LPDDR4 EVK		i.MX 8M Plus LPDDR4 EVK		i.MX 91 QSB	i.MX 91 EVK	i.MX 93 9x9 LPDDR4 QSB		i.MX 93 EVK		i.MX 93 14x14 EVK		i.MX 943 15x15/19x19 EVK			i.MX 95 15x15 EVK		i.MX 95 19x19 EVK	
Feature	**Sub-feature**	**RTOS**	**Application**	**4 X A53**	**1 X M4**	**4 X A53**	**1 X M7**	**1 X A55**	**1 X A55**	**2 X A55**	**1 X M33**	**2 X A55**	**1 X M33**	**2 X A55**	**1 X M33**	**4 X A55**	**1 X M33**	**2 X M7**	**6 X A55**	**1 X M7**	**6 X A55**	**1 X M7**
	UART Sharing based on RPMsg	Free RTOS	rpmsg_uart_ sharing		Y						Y		Y		Y							
Heterogeneous Multicore VirtIO	Heterogeneous Multicore VirtIO Performance Evaluation	Free RTOS	virtio_perf	Y	Y	Y	Y															
	Heterogeneous Multicore VirtIO Network Sharing	Free RTOS	virtio_net_ backend	Y	Y	Y	Y					Y	Y	Y	Y							
NETC networking sharing	NETC PSI/VSI networking sharing	Free RTOS	netc_share																			Y
Industrial Applications	SOEM Digital IO	Free RTOS	soem_digital_ io	Y	Y	Y	Y					Y										
	SOEM Servo Motor	Free RTOS	soem_servo	Y	Y	Y	Y					Y										
	SOEM Servo Motor RT1180	Free RTOS	soem_servo_ rt1180	Y	Y	Y	Y					Y										

[1] Using the `rpmsg_str_echo`
[2] Using the `rpmsg_uart_sharing`

3.2 Building Heterogeneous Multicore RTOS applications

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

There are two methods to build Heterogeneous Multicore RTOS applications. The first method leverages Yocto, whereas the other is a standalone method that uses ARM gcc directly.

3.2.1 Building with Yocto

Real-time Edge supports to build all images by using Yocto. Refer to Building, deploying, and releasing unified software for how to leverage Yocto to build RTOS Applications on Cortex-A or Cortex-M cores.

The followings are some Yocto quick commands:

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

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

Build single Heterogeneous Multicore RTOS application separately:

```
bitbake APP-NAME
```

It will build both the FreeRTOS and Zephyr applications on both Cortex-A and Cortex-M core if supported:

- hello-world

- lwip-ping
- netc-share
- rpmsg-perf
- rpmsg-str-echo
- rpmsg-pingpong
- rpmsg-uart-sharing
- rt-latency
- virtio-net-backend
- virtio-perf
- some-digital-io
- some-servo
- some-servo-rt1180

3.2.2 Building in standalone mode

Some RTOS applications are in the repository: heterogeneous-multicore, which the `west` tool manages. Therefore, you can use `west` to download all the software components and then use the Arm gcc toolchain to build the application directly. To build in standalone mode, follow the steps below:

1. **Download and install the prerequisites**
2. **Update the OS**
   Ubuntu 22.04 LTS and later versions are the recommended development environment.
   ```
   sudo apt update
   sudo apt upgrade
   ```
3. **Install dependencies**
   The current minimum required version for the main dependencies are:
   - CMake: v3.20.5
   - Python: V3.10

   Use `apt` to install the required dependencies:
   ```
   sudo apt install --no-install-recommendsgit cmake ninja-build gperf \
     ccache dfu-util device-tree-compiler wget python3-dev python3-venv python3-
   tk \
     xz-utils file makegcc gcc-multilib g++-multilib libsdl2-dev libmagic1
   ```
4. Create a new virtual environment:
   ```
   # create workspace directory
   mkdir workspace
   # create a virtual environment in the workspace directory
   python3 -m venv workspace/.venv
   # active the virtual environment
   source workspace/.venv/bin/activate
   # install west
   pip3 install west
   ```
   Note: *Execute the command `source workspace/.venv/bin/activate` to return to this virtual environment.*
5. **Get the source code**
   a. Initialize the development workspace with the `heterogeneous_multicore` manifest repo:
   ```
   export revision=main
   west init -m https://github.com/nxp-real-time-edge-sw/heterogeneous-
   multicore.git --mr${revision} workspace
   ```

NOTE: Using `${revision}` with "`main`" uses the latest development branch. Alternatively, you can replace "`main`" with any Real-Time Edge release you wish to use, such as Real_Time_Edge_v3.3_202512.

b. Enable the `west extensions` commands as shown below:

```
cd workspace
west config commands.allow_extensions true
```

c. Update the workspace:

```
west update
```

d. Install the required Python packages:

```
pip install -r mcuxsdk/mcuxsdk/scripts/requirements.txt
pip install -r zsdk/zephyr/scripts/requirements.txt
```

e. **Install the Cross Compiler Toolchain for building FreeRTOS applications.**
A cross-compiler is required to build Cortex-A and Cortex-M applications. This project is compatible with the Arm GCC toolchain. Download the toolchain and install it by following the steps below:

```
mkdir ~/toolchains/; cd ~/toolchains/
wget https://developer.arm.com/-/media/Files/downloads/gnu/14.2.rel1/
binrel/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi.tar.xz
tar xf arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi.tar.xz
wget https://developer.arm.com/-/media/Files/downloads/gnu/14.2.rel1/
binrel/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-elf.tar.xz
tar xf arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-elf.tar.xz
```

f. Install Zephyr SDK for building Zephyr applications (***optional***)
To build the Zephyr application, you can use the Zephyr SDK, which provides a cross-compile toolchain for multiple architectures.
Refer to the Zephyr document to Install the Zephyr SDK.
Or use the ARM GNU cross-compile GCC toolchain for building Zephyr applications, which could use the same toolchain with FreeRTOS building. For this case, installing Zephyr SDK is not required.

3.2.2.1 Building the application

Both FreeRTOS application and Zephyr application use west extension commands for building.

3.2.2.1.1 Building the FreeRTOS Application

Use the ***west sdk_build*** extension command for building FreeRTOS applications. Each application in the heterogeneous_multicore project also provides a building script for convenience.

1. Building the application on Cortex-M Core
   - Using the *west sdk_build* extension command:
     For example, use the command below to build the `hello_world` application running on evkmimx8mm Cortex-M Core:

     ```
     cd workspace/heterogeneous-multicore
     export ARMGCC_DIR=~/toolchains/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-
     eabi
     west sdk_build -p always apps/hello_world/freertos/ -b evkmimx8mm --config
      release -Dcore_id=cm4
     ```

     The application binary image "`hello_world_cm4.bin`" is located in the "`build`" directory.
   - Using the building script:

For example, to build the network sharing backend firmware running on evkmimx8mm Cortex-M Core, use the command below:

```
export ARMGCC_DIR=~/toolchains/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-
eabi
cd workspace/heterogenous-multicore/apps/virtio_net_backend/freertos/boards/
evkmimx8mm/cm4/armgcc
./build_release.sh
```

The backend firmware image "`virtio_net_backend_cm4.bin`" is created in the "`release`" directory after this command is run.

2. Building the application on Cortex-A Core
   - Use the `west sdk_build` extension command.
     For example, to build the `hello_world` application running on evkmimx8mm Cortex-A Core, use the command below:

```
cd workspace/heterogeneous-multicore
export ARMGCC_DIR=~/toolchains/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-
none-elf
west sdk_build -p always apps/hello_world/freertos/ -b evkmimx8mm --config
 ddr_release -Dcore_id=ca53
```

     The application binary image `hello_world_ca53.bin` is located in the "`build`" directory.
   - Use a building script
     For example, to build the network sharing backend firmware running on Cortex-A Core, use the command below:

```
export ARMGCC_DIR=~/toolchains/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-
none-elf
cd workspace/heterogeneous-multicore/apps/virtio_net_backend/freertos/
boards/evkmimx8mm/ca53/armgcc_aarch64
./build_ddr_release.sh
```

     The backend firmware image "`virtio_net_backend_ca53.bin`" is created in the directory "`ddr_release`".

3.2.2.1.2 Building the Zephyr application

Setup environment variables for the cross-compile toolchain (no need if using Zephyr SDK)

If you are using cross compile toolchain to build Zephyr, set the following environment variables:

- For building a Cortex-A Core Zephyr application:

```
export ZEPHYR_TOOLCHAIN_VARIANT=cross-compile
export CROSS_COMPILE=~/toolchains/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-
none-elf/bin/aarch64-none-elf-
export CROSS_COMPILE_TOOLCHAIN_PATH=~/toolchains/arm-gnu-toolchain-14.2.rel1-
x86_64-aarch64-none-elf
```

- For building a Cortex-M Core Zephyr application:

```
export ZEPHYR_TOOLCHAIN_VARIANT=cross-compile
export CROSS_COMPILE=~/toolchains/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-
eabi/bin/arm-none-eabi-
export CROSS_COMPILE_TOOLCHAIN_PATH=~/toolchains/arm-gnu-toolchain-14.2.rel1-
x86_64-arm-none-eabi/
```

Building Zephyr application examples

Use the ***west build*** extension command for building Zephyr applications, and each application in the heterogeneous_multicore project also provides a building script for convenience.

- Building applications provided by the Zephyr kernel

  ```
  cd workspace/zsdk/zephyr/
  west build -p always -b imx93_evk/mimx9352/a55 samples/hello_world/
  ```

- Building the application in `heterogeneous_multicore` project by using the `west build` command:

  ```
  cd workspace/heterogeneous-multicore
  west build -p always -b imx93_evk/mimx9352/a55 apps/hello_world/zephyr/ -DCONSOLE=UART2 -
  DRTOS_ID=0
  ```

  Note: *This completes the `west build` command for the hello_world Zephyr application on theA55 core of imx93_evk board with optional CONSOLE and RTOS_ID parameters.*

- Building the application in a `heterogeneous_multicore` project by using a building script:

  ```
  cd workspace/heterogeneous-multicore/apps/hello_world/zephyr/boards/evkmimx8mm/ca53/armgcc_aarch64
  ./build_release.sh
  ```

  Then, the following binary Zephyr images are built out:

  ```
  ./build_RTOS0_UART4/zephyr/hello_world_ca53_RTOS0_UART4.bin
  ./build_RTOS0_RAM_CONSOLE/zephyr/
  hello_world_ca53_RTOS0_RAM_CONSOLE-0x93d00000.bin
  ./build_RTOS1_RAM_CONSOLE/zephyr/
  hello_world_ca53_RTOS1_RAM_CONSOLE-0x94d00000.bin
  ./build_RTOS2_RAM_CONSOLE/zephyr/
  hello_world_ca53_RTOS2_RAM_CONSOLE-0x95d00000.bin
  ./build_RTOS3_UART2/zephyr/hello_world_ca53_RTOS3_UART2.bin
  ./build_RTOS3_RAM_CONSOLE/zephyr/
  hello_world_ca53_RTOS3_RAM_CONSOLE-0x96d00000.bin
  ```

3.2.2.1.3 Building with a Helper Script

The file "`build_apps.sh`" located in the root directory of "`heterogeneous-multicore`" can be used to build a single or all applications for all boards. The following is the help information for the "`build_apps.sh`" tool:

```
./build_apps.sh [clean]                                        - build or clean
 all applications
./build_apps.sh [clean] [os] [board-list] [app-list] [core]    - build or clean
 specified applications
      - os: specify freertos or zephyr or both if no specified.
      - core: a-core or m-core
      - board-list: specify one or some or all boards if no specified:
 evkmimx8mm_ca53 evkmimx8mp_ca53 mcimx91evk_ca55 mcimx91qsb_ca55 mcimx93evk_ca55
 mcimx95evk_ca55 evkmimx8mm_cm4 evkmimx8mp_cm7 mcimx93evk_cm33
      - app-list: specify one or some or all applications if no
 specified: hello_world lwip_ping rpmsg_perf rpmsg_pingpong rpmsg_str_echo
 rpmsg_uart_sharing rt_latency soem_digital_io soem_servo virtio_net_backend
 virtio_perf
```

For example:

```
     ./build_apps.sh a-core                             -build all a-core FreeRTOS
 and Zephyr applications
     ./build_apps.sh a-core evkmimx8mp_ca53 zephyr  -build all a-core Zephyr
 applications on evkmimx8mp
     ./build_apps.sh m-core                             -build all m-core
 applications
```

```
    ./build_apps.sh a-core hello_world zephyr      -build all a-core Zephyr
hello_world applications
    ./build_apps.sh clean                          -clean all applications
    ./build_apps.sh a-core clean                   -clean all a-core
applications
    ./build_apps.sh m-core hello_world clean       -clean all m-core
hello_world applications
```

Important: *Set the toolchain environment variable "`ARMGCC_DIR`" first before using the tool. If using the cross compilation toolchain, also set the environment variables "`ZEPHYR_TOOLCHAIN_VARIANT`", "`CROSS_COMPILE`", and "`CROSS_COMPILE_TOOLCHAIN_PATH`" for building Zephyr.*

For example, use the tool to build all `hello_world` application on Cortex-M Core for all supported boards:

```
export ARMGCC_DIR=~/toolchains/arm-gnu-toolchain-14.2.rel1-x86_64-arm-none-eabi
cd workspace/heterogeneous-multicore/
./build_apps.sh m-core hello_world
```

Use the cross compile toolchain to build all Zephyr applications on Cortex-A Core for all supported boards:

```
export ZEPHYR_TOOLCHAIN_VARIANT="cross-compile"
export CROSS_COMPILE=~/toolchains/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-
none-elf/bin/aarch64-none-elf-
export CROSS_COMPILE_TOOLCHAIN_PATH=~/toolchains/arm-gnu-toolchain-14.2.rel1-
x86_64-aarch64-none-elf
cd workspace/heterogeneous-multicore/
./build_apps.sh a-core zephyr
```

After executing the tool, all binary images built can be located in the directory: "`deploy/images`".

3.3 Flexible Real-time System

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

The system provides RAM Console to make it easy to debug multiple OSes if the physical UART consoles are not enough. The system also provides some common software stacks such as lwIP networking stack on Cortex-A core or Cortex-M core.

3.3.1 Heterogeneous Multicore RAM Console

RAM Console is a virtual debug console that the RTOS can use. It provides common console APIs to print and save console log to a reserved memory region.

If multiple OSes run on an MPU platform, some OSes can use a physical UART as the debug console. In such a case, the other OSes can use the RAM Console. This process requires to dump the RAM console of the other OS from the OS that uses the physical UART console.

The two methods to dump the RAM Console log are by using U-Boot commands or by using the Linux Userspace tool.

3.3.1.1 Using the RAM Console in FreeRTOS

This section describes how to develop a FreeRTOS application based on the RAM console.

3.3.1.1.1 RAM Console technical details

The RAM Console driver is located at the debug console (`utilities/debug_console`) in the repository mcux-sdk. There is a 64-byte Console Header at the start of the RAM Console log memory. Figure 28 shows the memory layout.

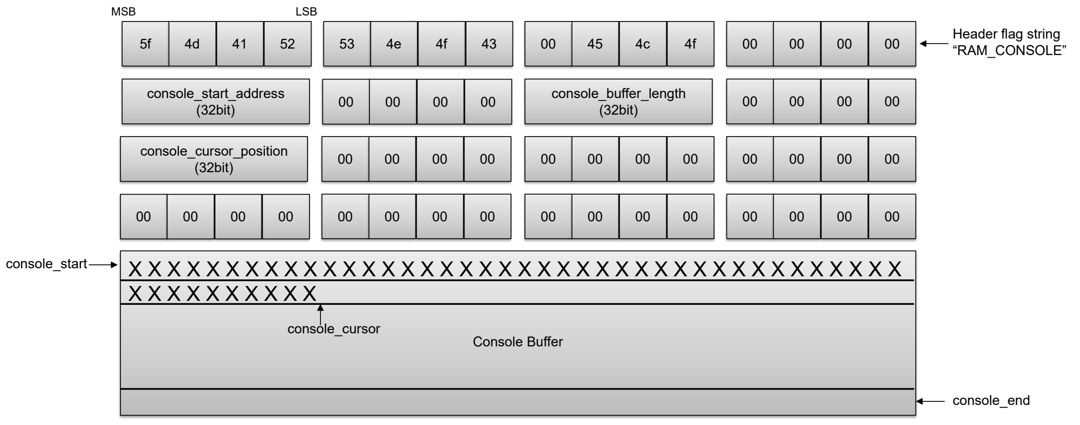

MSB
LSB
5f
4d
41
52
53
4e
4f
43
00
45
4c
4f
00
00
00
00
Header flag string "RAM_CONSOLE"
console_start_address (32bit)
console_buffer_length (32bit)
console_cursor_position (32bit)
console_start
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXX
console_cursor
Console Buffer
console_end

Figure 28. RAM console memory layout

The RAM Console Header includes the following parts:

- 16 bytes fixed string flag: "`RAM_CONSOLE`".
- `console_start_address`: the memory physical address for the console_start of Console buffer.
- `console_buffer_length`: the length in bytes of Console Buffer from console_start to console_end.
- `console_cursor_position`: The current console cursor increased position from console starting.

3.3.1.1.2 Develop with RAM Console

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

1. Reserve a memory block for RAM Console, and add mmu mapping entry in `app_mmu.h`.
2. In the `prj.conf` file of the application, enable the `CONFIG_MCUX_COMPONENT_utility.ram_console` parameter.

```
...
CONFIG_MCUX_COMPONENT_utility.ram_console=y
...
```

3. Call the function `RamConsole_Init()` with the address and size of the RAM console memory block to initialize the RAM console.

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

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

3.3.1.2 Use RAM Console in Zephyr

The following sections describe how to develop a Zephyr application based on the RAM Console.

3.3.1.2.1 RAM Console Technical Details

In Zephyr RAM console, there is the same 64 bytes Console Header as the one in FreeRTOS, refer to Section 3.3.1.1.1.

The RAM Console driver in Zephyr is `drivers/console/ram_console.c`.

3.3.1.2.2 Develop with the RAM Console

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

1. Enable the RAM console driver in the `prj.conf` file, and disable the UART console driver.

```
CONFIG_RAM_CONSOLE=y
CONFIG_UART_CONSOLE=n
```

2. Add a device tree node to add a memory region and section for the RAM console buffer in the board `dts` file.

```

  chosen {
               /delete-property/ zephyr,console;
               /delete-property/ zephyr,shell-uart;
               zephyr,ram-console = &ram_console;
       };
    ram_console: memory@93d00000 {
               compatible = "zephyr,memory-region";
               reg = <0x93d00000 DT_SIZE_K(4)>;
               zephyr,memory-region = "RAM_CONSOLE";
       };
```

3. In general, while running multiple RTOS instances, all the RAM Console memory addresses can be displayed on the physical debug console from the RTOS0 instance using the physical debug console.

3.3.1.3 Dump RAM Console Log

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

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

```
u-boot=> dcache flush; md 0xC0FFF000
c0fff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
c0fff010: c0fff040 00000000 00000fc0 00000000  @...............
c0fff020: 000002e9 00000000 00000000 00000000  ................
c0fff030: 00000000 00000000 00000000 00000000  ................
c0fff040: 6f430a0d 78657472 3335412d 5452203a  ..Cortex-A53: RT
```

```
c0fff050: 3a30534f 6c654820 77206f6c 646c726f  OS0: Hello world
c0fff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
c0fff070: 206e6f20 584d494d 2d4d5038 0d4b5645   on MIMX8MP-EVK.
c0fff080: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
c0fff090: 203a305f 6c6c6568 2030206f 656d6974  _0: hello 0 time
c0fff0a0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
c0fff0b0: 6f632033 20326572 49504d28 30203a44  3 core2 (MPID: 0
c0fff0c0: 0d293278 6572460a 4f545265 68745f53  x2)..FreeRTOS_th
c0fff0d0: 64616572 203a305f 6c6c6568 2031206f  read_0: hello 1
c0fff0e0: 656d6974 72662073 43206d6f 6574726f  times from Corte
c0fff0f0: 35412d78 6f632033 20326572 49504d28  x-A53 core2 (MPI
c0fff100: 30203a44 0d293278 6572460a 4f545265  D: 0x2)..FreeRTO
c0fff110: 68745f53 64616572 203a305f 6c6c6568  S_thread_0: hell
c0fff120: 2032206f 656d6974 72662073 43206d6f  o 2 times from C
c0fff130: 6574726f 35412d78 6f632033 20326572  ortex-A53 core2
c0fff140: 49504d28 30203a44 0d293278 6572460a  (MPID: 0x2)..Fre
c0fff150: 4f545265 68745f53 64616572 203a305f  eRTOS_thread_0:
c0fff160: 6c6c6568 2033206f 656d6974 72662073  hello 3 times fr
c0fff170: 43206d6f 6574726f 35412d78 6f632033  om Cortex-A53 co
c0fff180: 20326572 49504d28 30203a44 0d293278  re2 (MPID: 0x2).
c0fff190: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
c0fff1a0: 203a305f 6c6c6568 2034206f 656d6974  _0: hello 4 time
c0fff1b0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
c0fff1c0: 6f632033 20326572 49504d28 30203a44  3 core2 (MPID: 0
c0fff1d0: 0d293278 6572460a 4f545265 68745f53  x2)..FreeRTOS_th
c0fff1e0: 64616572 203a305f 6c6c6568 2035206f  read_0: hello 5
c0fff1f0: 656d6974 72662073 43206d6f 6574726f  times from Corte
```

- Dump from Linux
  There is a Linux userspace tool provided in the repository heterogeneous-multicore "tool" directory. Use this tool to dump RAM Console log.
  The following shows the help information of the tool:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump

RAM Console tool to dump console log.
Usage:ram_console_dump -a buffer_address [-r refresh_period_in_seconds]
                -a buffer_address:      RAM Console buffer physical address
                -r refresh_period:      Refresh period time in seconds for
 continuous dump, ctrl+c to exit
```

  The following log shows the example to use this tool. It dumps the RAM Console from address `0xC0FFF000` in the loop period of one second:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xC0FFF000 -r 1
RAM Console@0xc0fff000:

Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MP-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 4 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 5 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 6 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 7 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 8 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 9 times from Cortex-A53 core2 (MPID: 0x2)
 >
```

- Dump by using debugger tools:

If there is no U-Boot or Linux running on the platform, users can use hardware debugger tools such as JLink or TRACE32 to dump the console log from a specified buffer memory.

3.3.2 Heterogeneous Multicore hello_world

This section describes details of the Heterogeneous Multicore `hello_world` application.

3.3.2.1 Overview

Heterogeneous Multicore `hello_world` application demonstrates flexible Real-time System on MPU platforms. It can run single or multiple FreeRTOS or Zephyr RTOS instances on Cortex-A Core and single FreeRTOS instance on Cortex-M with or without running Linux simulteneously.

Note: *In this release, it only supports the Zephyr `hello_world` application on Cortex-A cores.*

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

Table 24. Flexible Real-time System on i.MX 8M Plus

ID	M7	A53	A53	A53	A53
0	RTOS	SMP Linux			
1	RTOS	SMP Linux			RTOS
2	RTOS	SMP Linux		RTOS	RTOS
3	RTOS	Linux	RTOS	RTOS	RTOS
4	RTOS	RTOS	RTOS	RTOS	RTOS

3.3.2.2 Technical Points

The following examples are for the i.MX 8M Plus Applications Processor.

- **Debug Console**: In general, UART4 is used for Cortex-M Core RTOS or Cortex-A Core RTOS, UART2 is used for Linux, the other RTOS instance can use RAM Console.
  The `hello_world` application allows building all RTOS instances with any possible UART Console or RAM Console. The following images are for i.MX 8M Plus EVK. Different RTOS instances RTOS0, RTOS1, RTOS2 and RTOS3 use different memory space. Each RTOS instance also provides the images with different Debug Console (UART4, UART2, or RAM Console (the buffer address is appended in the image name). Users can run any of these images according to use case setup in order to avoid memory and debug console conflicts.

```
evkmimx8mp/
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.bin       #RTOS0 with RAM
 Console at 0xc0fff000
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.elf       #RTOS0 with RAM
 Console at 0xc0fff000
├── hello_world_ca53_RTOS0_UART4.bin                        #RTOS0 with UART4
 Debug Console
├── hello_world_ca53_RTOS0_UART4.elf                        #RTOS0 with UART4
 Debug Console
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.bin       #RTOS1 with RAM
 Console at 0xc1fff000
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.elf       #RTOS1 with RAM
 Console at 0xc1fff000
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0xc2fff000.bin       #RTOS2 with RAM
 Console at 0xc2fff000
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0xc2fff000.elf       #RTOS2 with RAM
 Console at 0xc2fff000
```

```
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0xc3fff000.bin       #RTOS3 with RAM
 Console at 0xc3fff000
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0xc3fff000.elf       #RTOS3 with RAM
 Console at 0xc3fff000
├── hello_world_ca53_RTOS3_UART2.bin                        #RTOS3 with UART2
 Debug Console
└── hello_world_ca53_RTOS3_UART2.elf                        #RTOS3 with UART2
 Debug Console
```

- **Memory Usage**
  - For FreeRTOS applications in the repository: heterogeneous-multicore, the memory resource used by FreeRTOS kernel is defined in "`os/freertos/Core_AArch64/boards/<PLAT_NAME>/rtos_memory.h`", for example, in the following rtos_memory.h for i.MX 8M Plus, RTOS0, RTOS1, RTOS2 and RTOS3, each RTOS uses 16M bytes memory from 0xC0000000, 0xC1000000, 0xC2000000 and 0xC3000000 respectively.

```
#ifndef _RTOS_MEMORY_H_
#define _RTOS_MEMORY_H_

/* Memory used by RTOS kernel */
#define M_INTERRUPTS_BASE        (0xC0000000 + 0x1000000 * RTOSID)
#define M_INTERRUPTS_LEN         0x00002000 /*  8 kB */

#define M_TEXT_BASE              (M_INTERRUPTS_BASE + M_INTERRUPTS_LEN)
#define M_TEXT_LEN               0x005FE000 /* ~6 MB */

#define M_DATA_BASE              (M_TEXT_BASE + M_TEXT_LEN)
#define M_DATA_LEN               0x005FE000 /* ~6 MB */

#define M_STACKS_BASE            (M_DATA_BASE + M_DATA_LEN)
#define M_STACKS_LEN             0x00002000 /*  8 kB */

#define M_STACKS_NC_BASE         (M_STACKS_BASE + M_STACKS_LEN)
#define M_STACKS_NC_LEN          0x003FF000 /*  ~4 MB */

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

#define RTOS_MEM_LEN             0x01000000 /* 16 MB */

#endif
```

  - **For Zephyr applications** in the repository: heterogeneous-multicore, the memory resource used by Zephyr kernel is defined in "`apps/hello_world/zephyr/boards/<PLAT_NAME>/rtos_memory.h`", for example, in the following rtos_memory.h for i.MX 8M Plus, RTOS0, RTOS1, RTOS2 and RTOS3, each RTOS uses 1M bytes memory for Zephyr kernel starting from 0xC0000000, 0xC1000000, 0xC2000000 and 0xC3000000 respectively, and 4K bytes memory for RAM console buffer appended.

```
#if (RTOSID == 0)
#define SRAM_BASE                c0000000
#define RAM_CONSOLE_BASE         c0100000
#elif (RTOSID == 1)
#define SRAM_BASE                c1000000
#define RAM_CONSOLE_BASE         c1100000
#elif (RTOSID == 2)
#define SRAM_BASE                c2000000
#define RAM_CONSOLE_BASE         c2100000
#elif (RTOSID == 3)
#define SRAM_BASE                c3000000
```

```
#define RAM_CONSOLE_BASE        c3100000
#else
#error "Unsupported RTOSID!"
#endif
```

– Running Linux kernel with RTOS simultaneously: For this case, you must reserve the memory space used by RTOS in the device tree, for example, in `imx8mp-evk-multicore-rtos.dts`. The following code shows an example:

```
             /*
              * Reserve up to 48MB (16MB x 3) for three FreeRTOS instances
 running on
              * three Cortex-A Cores when booting Linux on at least on Cortex-
A Core.
              */
             ca53_reserved: ca53@c0000000 {
                     no-map;
                     reg = <0 0xc0000000 0x0 0x3000000>;
             };

             /* Reserve 16MB for RTOS running on CM7 */
             m7_reserved: m7@80000000 {
                     no-map;
                     reg = <0 0x80000000 0 0x1000000>;
             };
```

3.3.2.3 Running flexible multicore hello_world application

3.3.2.3.1 Running use cases on i.MX 8M Plus LPDDR4 EVK

3.3.2.3.1.1 Overview

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

Table 25. Flexible Real-time System on i.MX 8M Plus

ID	M7	A53 Core0	A53 Core1	A53 Core2	A53 Core3
0	RTOS	SMP Linux			
1	RTOS	SMP Linux			RTOS
2	RTOS	SMP Linux		SMP RTOS	
3	RTOS	Linux	RTOS	RTOS	RTOS
4	RTOS	RTOS	RTOS	RTOS	RTOS
5	RTOS	SMP RTOS			

Note: *The SMP RTOS is not supported on FreeRTOS.*

By default, the RTOS of the Cortex-M Core uses UART4 as the debug console. Refer to Section 2.8.1 for how to boot Cortex-M Core's RTOS image.

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

```
#FreeRTOS A53 hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
```

```
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.bin
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.elf
├── hello_world_ca53_RTOS0_UART4.bin
├── hello_world_ca53_RTOS0_UART4.elf
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.bin
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.elf
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0xc2fff000.bin
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0xc2fff000.elf
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0xc3fff000.bin
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0xc3fff000.elf
├── hello_world_ca53_RTOS3_UART2.bin
└── hello_world_ca53_RTOS3_UART2.elf
#Zephyr A53 hello_world Images
/examples/heterogeneous-multicore/hello-world-zephyr/
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0100000.bin
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0100000.elf
├── hello_world_ca53_RTOS0_SMP_2CORES_RAM_CONSOLE-0xc0100000.bin
├── hello_world_ca53_RTOS0_SMP_2CORES_RAM_CONSOLE-0xc0100000.elf
├── hello_world_ca53_RTOS0_SMP_2CORES_UART4.bin
├── hello_world_ca53_RTOS0_SMP_2CORES_UART4.elf
├── hello_world_ca53_RTOS0_SMP_4CORES_RAM_CONSOLE-0xc0100000.bin
├── hello_world_ca53_RTOS0_SMP_4CORES_RAM_CONSOLE-0xc0100000.elf
├── hello_world_ca53_RTOS0_SMP_4CORES_UART4.bin
├── hello_world_ca53_RTOS0_SMP_4CORES_UART4.elf
├── hello_world_ca53_RTOS0_UART4.bin
├── hello_world_ca53_RTOS0_UART4.elf
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1100000.bin
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1100000.elf
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0xc2100000.bin
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0xc2100000.elf
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0xc3100000.bin
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0xc3100000.elf
├── hello_world_ca53_RTOS3_UART2.bin
└── hello_world_ca53_RTOS3_UART2.elf
#FreeRTOS M7 hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
├── hello_world_cm7.bin
└── hello_world_cm7.elf
```

RAM Console address is appended at the end of image name, for example `hello_world_ca53_RTOS3_RAM_CONSOLE-0xc3100000.bin` , it uses RAM Console which buffer address is `0xc3100000`.

Refer to Section 2.6.3 for how to boot RTOS on Cortex-A Core, and refer to Section 3.3.1.3 for how to dump RAM Console log.

The following chapters introduce how to use U-Boot commands or Linux remoteproc to boot FreeRTOS or Zephyr use cases.

3.3.2.3.1.2 Running flexible multicore use cases by using U-Boot Commands

- This section describes the steps to run FreeRTOS applications:
  To run FreeRTOS on M4 core, follow the steps below. These steps are for running two FreeRTOS instances on A53 Core2 and Core3, and a SMP Linux kernel on A53 Core0 and Core1.
  1. Boot RTOS of Cortex-M core.

     ```
     u-boot=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/
     hello-world-freertos/hello_world_cm7.bin;
     u-boot=> cp.b 0x48000000 0x7e0000 20000;
     u-boot=> bootaux 0x7e0000
     ```

Then the following log is displayed from UART4:

```
Cortex-M7: RTOS0: Hello world! Real-time Edge on MIMX8MP-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M7 core0 (MPID: 0x0)
```

2. Boot the first RTOS of Cortex-A core on A53 Core2.

```
u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
```

Then, check the RAM Console log as follows:

```
u-boot=> dcache flush; md 0xC0FFF000
c0fff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
c0fff010: c0fff040 00000000 00000fc0 00000000  @...............
c0fff020: 000002e9 00000000 00000000 00000000  ................
c0fff030: 00000000 00000000 00000000 00000000  ................
c0fff040: 6f430a0d 78657472 3335412d 5452203a  ..Cortex-A53: RT
c0fff050: 3a30534f 6c654820 77206f6c 646c726f  OS0: Hello world
c0fff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
c0fff070: 206e6f20 584d494d 2d4d5038 0d4b5645   on MIMX8MP-EVK.
c0fff080: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
c0fff090: 203a305f 6c6c6568 2030206f 656d6974  _0: hello 0 time
c0fff0a0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
c0fff0b0: 6f632033 20326572 49504d28 30203a44  3 core2 (MPID: 0
c0fff0c0: 0d293278 6572460a 4f545265 68745f53  x2)..FreeRTOS_th
c0fff0d0: 64616572 203a305f 6c6c6568 2031206f  read_0: hello 1
c0fff0e0: 656d6974 72662073 43206d6f 6574726f  times from Corte
c0fff0f0: 35412d78 6f632033 20326572 49504d28  x-A53 core2 (MPI
c0fff100: 30203a44 0d293278 6572460a 4f545265  D: 0x2)..FreeRTO
c0fff110: 68745f53 64616572 203a305f 6c6c6568  S_thread_0: hell
c0fff120: 2032206f 656d6974 72662073 43206d6f  o 2 times from C
c0fff130: 6574726f 35412d78 6f632033 20326572  ortex-A53 core2
c0fff140: 49504d28 30203a44 0d293278 6572460a  (MPID: 0x2)..Fre
c0fff150: 4f545265 68745f53 64616572 203a305f  eRTOS_thread_0:
c0fff160: 6c6c6568 2033206f 656d6974 72662073  hello 3 times fr
c0fff170: 43206d6f 6574726f 35412d78 6f632033  om Cortex-A53 co
c0fff180: 20326572 49504d28 30203a44 0d293278  re2 (MPID: 0x2).
c0fff190: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
c0fff1a0: 203a305f 6c6c6568 2034206f 656d6974  _0: hello 4 time
c0fff1b0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
c0fff1c0: 6f632033 20326572 49504d28 30203a44  3 core2 (MPID: 0
c0fff1d0: 0d293278 6572460a 4f545265 68745f53  x2)..FreeRTOS_th
c0fff1e0: 64616572 203a305f 6c6c6568 2035206f  read_0: hello 5
c0fff1f0: 656d6974 72662073 43206d6f 6574726f  times from Corte
```

3. Boot the second Cortex-A Core's RTOS on A53 Core3

```
u-boot=> ext4load mmc 1:2 0xC1000000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.bin
u-boot=> dcache flush; icache flush; cpu 3 release 0xC1000000
```

Then check the RAM Console's log as follows:

```
u-boot=> dcache flush; md 0xC1FFF000
c1fff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
c1fff010: c1fff040 00000000 00000fc0 00000000  @...............
c1fff020: 0000032e 00000000 00000000 00000000  ................
c1fff030: 00000000 00000000 00000000 00000000  ................
```

```
c1fff040: 6f430a0d 78657472 3335412d 5452203a  ..Cortex-A53: RT
c1fff050: 3a31534f 6c654820 77206f6c 646c726f  OS1: Hello world
c1fff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
c1fff070: 206e6f20 584d494d 2d4d5038 0d4b5645   on MIMX8MP-EVK.
c1fff080: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
c1fff090: 203a305f 6c6c6568 2030206f 656d6974  _0: hello 0 time
c1fff0a0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
c1fff0b0: 6f632033 20336572 49504d28 30203a44  3 core3 (MPID: 0
c1fff0c0: 0d293378 6572460a 4f545265 68745f53  x3)..FreeRTOS_th
c1fff0d0: 64616572 203a305f 6c6c6568 2031206f  read_0: hello 1
c1fff0e0: 656d6974 72662073 43206d6f 6574726f  times from Corte
c1fff0f0: 35412d78 6f632033 20336572 49504d28  x-A53 core3 (MPI
c1fff100: 30203a44 0d293378 6572460a 4f545265  D: 0x3)..FreeRTO
c1fff110: 68745f53 64616572 203a305f 6c6c6568  S_thread_0: hell
c1fff120: 2032206f 656d6974 72662073 43206d6f  o 2 times from C
c1fff130: 6574726f 35412d78 6f632033 20336572  ortex-A53 core3
c1fff140: 49504d28 30203a44 0d293378 6572460a  (MPID: 0x3)..Fre
c1fff150: 4f545265 68745f53 64616572 203a305f  eRTOS_thread_0:
c1fff160: 6c6c6568 2033206f 656d6974 72662073  hello 3 times fr
c1fff170: 43206d6f 6574726f 35412d78 6f632033  om Cortex-A53 co
c1fff180: 20336572 49504d28 30203a44 0d293378  re3 (MPID: 0x3).
c1fff190: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
c1fff1a0: 203a305f 6c6c6568 2034206f 656d6974  _0: hello 4 time
c1fff1b0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
c1fff1c0: 6f632033 20336572 49504d28 30203a44  3 core3 (MPID: 0
c1fff1d0: 0d293378 6572460a 4f545265 68745f53  x3)..FreeRTOS_th
c1fff1e0: 64616572 203a305f 6c6c6568 2035206f  read_0: hello 5
c1fff1f0: 656d6974 72662073 43206d6f 6574726f  times from Corte
```

4. Boot SMP Linux from A53 Core0 and Core1:
   If M-Core is booting up, use `imx8mp-evk-multicore-rpmsg.dtb`

   ```
   u-boot=> setenv fdtfile imx8mp-evk-multicore-rpmsg.dtb
   ```

   Otherwise, can also use `imx8mp-evk-multicore-rtos.dtb`:

   ```
   u-boot=> setenv fdtfile imx8mp-evk-multicore-rtos.dtb
   ```

   Then, boot up the kernel using the command below:

   ```
   u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
   u-boot=> boot
   ```

   After Linux kernel boots up, check Cortex-A Core's RTOS log from the RAM console:

   ```
   root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xC0FFF000 -r 1
   RAM Console@0xc0fff000:

   Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MP-EVK
   FreeRTOS_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 4 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 5 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 6 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 7 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 8 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 9 times from Cortex-A53 core2 (MPID: 0x2)
    >
   ```

   ```
   root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xC1FFF000 -r 1
   RAM Console@0xc1fff000:

   Cortex-A53: RTOS1: Hello world! Real-time Edge on MIMX8MP-EVK
   FreeRTOS_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
   ```

```
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 4 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 5 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 6 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 7 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 8 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 9 times from Cortex-A53 core3 (MPID: 0x3)
 >
```

- Run the Zephyr applications:
  - Follow the following steps to run two Zephyr instances on A53 Core2 and Core3, and run a SMP Linux kernel on A53 Core0 and Core1.
    1. Boot the first Cortex-A Core's Zephyr instance on A53 Core2 with UART4:

```
u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca53_RTOS0_UART4.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
```

Then the UART4 shows the logs as follows:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mp_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
```

    2. Boot the second Cortex-A Core's Zephyr instance on A53 Core3

```
u-boot=> ext4load mmc 1:2 0xC1000000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1100000.bin
u-boot=> dcache flush; icache flush; cpu 3 release 0xC1000000
```

Then check the RAM Console's log as follows:

```
u-boot=>  dcache flush; md 0xC1100000
c1100000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
c1100010: c1100040 00000000 00000fbf 00000000  @...............
c1100020: 0000030c 00000000 00000000 00000000  ................
c1100030: 00000000 00000000 00000000 00000000  ................
c1100040: 202a2a2a 746f6f42 20676e69 6870655a  *** Booting Zeph
c1100050: 4f207279 75622053 20646c69 322e3476  yr OS build v4.2
c1100060: 312d302e 38333438 3766672d 34633931  .0-18438-gf719c4
c1100070: 33643262 2a203764 430a2a2a 6574726f  b2d3d7 ***.Corte
c1100080: 35412d78 52203a33 31534f54 6548203a  x-A53: RTOS1: He
c1100090: 206f6c6c 6c726f57 52202164 2d6c6165  llo World! Real-
c11000a0: 656d6974 67644520 6e6f2065 786d6920  time Edge on imx
c11000b0: 5f6d7038 0a6b7665 6870655a 745f7279  8mp_evk.Zephyr_t
c11000c0: 61657268 3a305f64 6c656820 30206f6c  hread_0: hello 0
c11000d0: 6d697420 66207365 206d6f72 74726f43   times from Cort
c11000e0: 412d7865 63203335 3365726f 504d2820  ex-A53 core3 (MP
c11000f0: 203a4449 29337830 655a0a0d 72796870  ID: 0x3)..Zephyr
c1100100: 7268745f 5f646165 68203a30 6f6c6c65  _thread_0: hello
c1100110: 74203120 73656d69 6f726620 6f43206d   1 times from Co
c1100120: 78657472 3335412d 726f6320 28203365  rtex-A53 core3 (
c1100130: 4449504d 7830203a 0a0d2933 6870655a  MPID: 0x3)..Zeph
c1100140: 745f7279 61657268 3a305f64 6c656820  yr_thread_0: hel
c1100150: 32206f6c 6d697420 66207365 206d6f72  lo 2 times from 
c1100160: 74726f43 412d7865 63203335 3365726f  Cortex-A53 core3
c1100170: 504d2820 203a4449 29337830 655a0a0d   (MPID: 0x3)..Ze
c1100180: 72796870 7268745f 5f646165 68203a30  phyr_thread_0: h
```

```
c1100190: 6f6c6c65 74203320 73656d69 6f726620  ello 3 times fro
c11001a0: 6f43206d 78657472 3335412d 726f6320  m Cortex-A53 cor
c11001b0: 28203365 4449504d 7830203a 0a0d2933  e3 (MPID: 0x3)..
c11001c0: 6870655a 745f7279 61657268 3a305f64  Zephyr_thread_0:
c11001d0: 6c656820 34206f6c 6d697420 66207365   hello 4 times f
c11001e0: 206d6f72 74726f43 412d7865 63203335  rom Cortex-A53 c
c11001f0: 3365726f 504d2820 203a4449 29337830  ore3 (MPID: 0x3)
```

3. Boot SMP Linux from A53 Core0 and Core1:

```
u-boot=> setenv fdtfile imx8mp-evk-multicore-rtos.dtb
```

Then, boot up the kernel using the command below:

```
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

– Follow the following steps to run the 2 cores SMP Zephyr instances on A53 Core2 and Core3, and run a SMP Linux kernel on A53 Core0 and Core1.

1. Boot the SMP Zephyr instance on A53 Core2 and Core3 with UART4:

```
u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca53_RTOS0_SMP_2CORES_UART4.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
```

Then the UART4 shows the logs as follows:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Secondary CPU core 1 (MPID:0x3) is up
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mp_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_1: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_1: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_1: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_1: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
```

2. Boot SMP Linux from A53 Core0 and Core1:

```
u-boot=> setenv fdtfile imx8mp-evk-multicore-rtos.dtb
```

Then, boot up the kernel using the command below:

```
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

– Follow the following steps to run the 4 cores SMP Zephyr instances on all the 4 A53 cores.

1. Boot the SMP Zephyr instance on A53 with UART4:

```
u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca53_RTOS0_SMP_4CORES_UART4.bin
u-boot=> dcache flush; icache flush; go 0xC0000000
```

Then the UART4 shows the logs as follows:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Secondary CPU core 1 (MPID:0x1) is up
Secondary CPU core 2 (MPID:0x2) is up
Secondary CPU core 3 (MPID:0x3) is up
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mp_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core0 (MPID: 0x0)
Zephyr_thread_1: hello 0 times from Cortex-A53 core1 (MPID: 0x1)
```

```
Zephyr_thread_2: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_3: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 1 times from Cortex-A53 core0 (MPID: 0x0)
Zephyr_thread_1: hello 1 times from Cortex-A53 core2 (MPID: 0x1)
Zephyr_thread_2: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_3: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
```

3.3.2.3.1.3 Running flexible multicore use cases by using remoteproc (i.MX 8M Plus)

In addition to using U-Boot commands, users can also use remoteproc under Linux to run RTOS. Take the following use case as example, using the flexible `hello_world` application images to set up the following use case on the i.MX 8M Plus board.

The remoteproc feature can be used to start and stop RTOS dynamically without rebooting the board. Demo switching from use case #1 to use case #2 dynamically works similarly by stopping the FreeRTOS instances and starting a 2 cores SMP Zephyr.

Table 26. Dynamic Flexible Real-time System on i.MX 8M Plus

ID	M7	A53-Core0	A53-Core1	A53-Core2	A53-Core3
1	FreeRTOS	SMP Linux		FreeRTOS	FreeRTOS
2	FreeRTOS	SMP Linux		SMP Zephyr	

1. Boot SMP Linux from A53 Core0, Core1, Core2 and Core3:

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

   After Linux boots up, use the command `cat /proc/cpuinfo` to check that the four A53 cores (Core0, Core1, Core2 and Core3) are all used by SMP Linux.
2. Boot FreeRTOS on M7 Core
   Use remoteproc to boot FreeRTOS image `hello_world_cm7.elf` on M7 Core, it uses UART4 as the debug console.

   ```
   root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-
   freertos/hello_world_cm7.elf >  /sys/devices/platform/imx8mp-cm7/remoteproc/
   remoteproc4/firmware
   root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mp-cm7/
   remoteproc/remoteproc4/state
   ```

   Then the following log is displayed on UART4:

   ```
   Cortex-M7: RTOS0: Hello world! Real-time Edge on MIMX8MP-EVK
   FreeRTOS_thread_0: hello 0 times from Cortex-M7 core0 (MPID: 0x0)
   FreeRTOS_thread_0: hello 1 times from Cortex-M7 core0 (MPID: 0x0)
   FreeRTOS_thread_0: hello 2 times from Cortex-M7 core0 (MPID: 0x0)
   FreeRTOS_thread_0: hello 3 times from Cortex-M7 core0 (MPID: 0x0)
   ```
3. Boot FreeRTOS on A53 Core2
   Use the following commands to boot FreeRTOS image `hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.elf` on the A53 Core2. Note that it uses the RAM Console, for which the buffer address is at `0xc0fff000`:

   ```
   root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-
   freertos/hello_world_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.elf > /sys/devices/
   platform/remoteproc-ca53-2/remoteproc/remoteproc1/firmware
   ```

```
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

Then, use the tool to dump RAM Console and log, which is as follows:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xC0FFF000 -r 1
RAM Console@0xc0fff000:

Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MP-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 4 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 5 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 6 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 7 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 8 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 9 times from Cortex-A53 core2 (MPID: 0x2)
 >
```

Using commands `cat /proc/cpuinfo` can check that A53 Core0, Core1 and Core3 are used by SMP Linux as Core2 is running FreeRTOS.

4. Boot FreeRTOS on A53 Core3
   Using the following commands to boot FreeRTOS image `hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.elf` on A53 Core3, it use RAM Console which buffer address is at `0xc1fff000`:

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.elf >  /sys/devices/
platform/remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

Then, use the tool to dump RAM Console and log is as follows:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xC1FFF000 -r 1
RAM Console@0xc1fff000:

Cortex-A53: RTOS1: Hello world! Real-time Edge on MIMX8MP-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 4 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 5 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 6 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 7 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 8 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 9 times from Cortex-A53 core3 (MPID: 0x3)
 >
```

Using commands `cat /proc/cpuinfo` can check that A53 Core0 and Core1 are used by SMP Linux as Core2 and Core3 are running FreeRTOS.

5. Stop FreeRTOS instances and then start the 2 cores SMP Zephyr on A53 Core2
   Stop FreeRTOS instances on A53 Core2 and Core3:

```
root@imx8mp-lpddr4-evk:~# echo stop > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
root@imx8mp-lpddr4-evk:~# echo stop > /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

Using commands `cat /proc/cpuinfo` can check that A53 Core0, Core1, Core2 and Core3 are all used by SMP Linux as Core2 and Core3 RTOS instances have been stopped and they were hot plugged back to Linux kernel.
Start Zephyr on A53 Core2:

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca53_RTOS0_SMP_2CORES_RAM_CONSOLE-0xc0100000.elf  > /sys/
devices/platform/remoteproc-ca53-2-3/remoteproc/remoteproc3/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-2-3/remoteproc/remoteproc3/state
```

Then, use the tool to dump RAM Console and log is as follows:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xc0100000 -r 1
RAM Console@0xc0100000:

*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Secondary CPU core 1 (MPID:0x3) is up
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mp_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_1: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_1: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_1: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_1: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
 >
```

Using commands `cat /proc/cpuinfo` can check that A53 Core0 and Core1 are used by SMP Linux as Core2 and Core3 are running RTOS.

3.3.2.3.2 Running use cases on i.MX 8M Mini LPDDR4 EVK

3.3.2.3.2.1 Overview

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

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

ID	M4	A53 Core0	A53 Core1	A53 Core2	A53 Core3
0	RTOS	SMP Linux			
1	RTOS	SMP Linux			RTOS
2	RTOS	SMP Linux		RTOS	RTOS
3	RTOS	Linux	RTOS	RTOS	RTOS
4	RTOS	RTOS	RTOS	RTOS	RTOS

By default, Cortex-M Core's RTOS uses UART4 as debug Console. Refer to Section 2.8.1 for how to boot Cortex-M Core's RTOS image.

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

```
#FreeRTOS A53 hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0x94bff000.bin
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0x94bff000.elf
├── hello_world_ca53_RTOS0_UART4.bin
├── hello_world_ca53_RTOS0_UART4.elf
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0x95bff000.bin
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0x95bff000.elf
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0x96bff000.bin
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0x96bff000.elf
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0x97bff000.bin
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0x97bff000.elf
├── hello_world_ca53_RTOS3_UART2.bin
└── hello_world_ca53_RTOS3_UART2.elf
#Zephyr A53 hello_world Images
/examples/heterogeneous-multicore/hello-world-zephyr/
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0x93d00000.bin
├── hello_world_ca53_RTOS0_RAM_CONSOLE-0x93d00000.elf
├── hello_world_ca53_RTOS0_UART4.bin
├── hello_world_ca53_RTOS0_UART4.elf
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0x94d00000.bin
├── hello_world_ca53_RTOS1_RAM_CONSOLE-0x94d00000.elf
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0x95d00000.bin
├── hello_world_ca53_RTOS2_RAM_CONSOLE-0x95d00000.elf
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0x96d00000.bin
├── hello_world_ca53_RTOS3_RAM_CONSOLE-0x96d00000.elf
├── hello_world_ca53_RTOS3_UART2.bin
└── hello_world_ca53_RTOS3_UART2.elf
#FreeRTOS M4 hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
├── hello_world_cm4.bin
└── hello_world_cm4.elf
```

RAM Console address is appended at the end of image name, for example `hello_world_ca53_RTOS3_RAM_CONSOLE-0x97bff000.bin` , it uses RAM Console which buffer address is `0x97bff000`.

Refer to Section 2.6.3 for how to boot RTOS on Cortex-A Core, and refer to Section 3.3.1.3 for how to dump RAM Console log.

The following chapters introduce how to use U-Boot commands or Linux remoteproc to boot FreeRTOS or Zephyr use cases.

3.3.2.3.2.2 Running use cases by using U-Boot Commands

- Run the FreeRTOS applications:
  Follow the following steps to run FreeRTOS on M4 Core, two FreeRTOS instances on A53 Core2 and Core3, and a SMP Linux kernel on A53 Core0 and Core1.
  1. Boot Cortex-M Core's RTOS

     ```
     u-boot=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/
     hello-world-freertos/hello_world_cm4.bin;
     u-boot=> cp.b 0x48000000 0x7e0000 20000;
     u-boot=> bootaux 0x7e0000
     ```

     Then, the following log is displayed from UART4:

     ```
     Cortex-M4: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
     FreeRTOS_thread_0: hello 0 times from Cortex-M4 core0 (MPID: 0x0)
     FreeRTOS_thread_0: hello 1 times from Cortex-M4 core0 (MPID: 0x0)
     FreeRTOS_thread_0: hello 2 times from Cortex-M4 core0 (MPID: 0x0)
     FreeRTOS_thread_0: hello 3 times from Cortex-M4 core0 (MPID: 0x0)
     ```

2. Boot the first Cortex-A Core's RTOS on A53 Core2

```
u-boot=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_ca53_RTOS0_RAM_CONSOLE-0x94bff000.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0x93C00000
```

Then check the RAM Console's log as follows:

```
u-boot=> dcache flush; md 0x94bff000
94bff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
94bff010: 94bff040 00000000 00000fc0 00000000  @...............
94bff020: 000002e9 00000000 00000000 00000000  ................
94bff030: 00000000 00000000 00000000 00000000  ................
94bff040: 6f430a0d 78657472 3335412d 5452203a  ..Cortex-A53: RT
94bff050: 3a30534f 6c654820 77206f6c 646c726f  OS0: Hello world
94bff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
94bff070: 206e6f20 584d494d 2d4d4d38 0d4b5645   on MIMX8MM-EVK.
94bff080: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
94bff090: 203a305f 6c6c6568 2030206f 656d6974  _0: hello 0 time
94bff0a0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
94bff0b0: 6f632033 20326572 49504d28 30203a44  3 core2 (MPID: 0
94bff0c0: 0d293278 6572460a 4f545265 68745f53  x2)..FreeRTOS_th
94bff0d0: 64616572 203a305f 6c6c6568 2031206f  read_0: hello 1
94bff0e0: 656d6974 72662073 43206d6f 6574726f  times from Corte
94bff0f0: 35412d78 6f632033 20326572 49504d28  x-A53 core2 (MPI
94bff100: 30203a44 0d293278 6572460a 4f545265  D: 0x2)..FreeRTO
94bff110: 68745f53 64616572 203a305f 6c6c6568  S_thread_0: hell
94bff120: 2032206f 656d6974 72662073 43206d6f  o 2 times from C
94bff130: 6574726f 35412d78 6f632033 20326572  ortex-A53 core2
94bff140: 49504d28 30203a44 0d293278 6572460a  (MPID: 0x2)..Fre
94bff150: 4f545265 68745f53 64616572 203a305f  eRTOS_thread_0:
94bff160: 6c6c6568 2033206f 656d6974 72662073  hello 3 times fr
94bff170: 43206d6f 6574726f 35412d78 6f632033  om Cortex-A53 co
94bff180: 20326572 49504d28 30203a44 0d293278  re2 (MPID: 0x2).
94bff190: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
94bff1a0: 203a305f 6c6c6568 2034206f 656d6974  _0: hello 4 time
94bff1b0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
94bff1c0: 6f632033 20326572 49504d28 30203a44  3 core2 (MPID: 0
94bff1d0: 0d293278 6572460a 4f545265 68745f53  x2)..FreeRTOS_th
94bff1e0: 64616572 203a305f 6c6c6568 2035206f  read_0: hello 5
94bff1f0: 656d6974 72662073 43206d6f 6574726f  times from Corte
```

3. Boot the second Cortex-A Core's RTOS on A53 Core3

```
u-boot=> ext4load mmc 1:2 0x94C00000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_ca53_RTOS1_RAM_CONSOLE-0x95bff000.bin
u-boot=> dcache flush; icache flush; cpu 3 release 0x94C00000
```

Then check the RAM Console's log as follows:

```
u-boot=> dcache flush; md 0x95bff000
95bff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
95bff010: 95bff040 00000000 00000fc0 00000000  @...............
95bff020: 0000032e 00000000 00000000 00000000  ................
95bff030: 00000000 00000000 00000000 00000000  ................
95bff040: 6f430a0d 78657472 3335412d 5452203a  ..Cortex-A53: RT
95bff050: 3a31534f 6c654820 77206f6c 646c726f  OS1: Hello world
95bff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
95bff070: 206e6f20 584d494d 2d4d4d38 0d4b5645   on MIMX8MM-EVK.
95bff080: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
95bff090: 203a305f 6c6c6568 2030206f 656d6974  _0: hello 0 time
95bff0a0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
```

```
95bff0b0: 6f632033 20336572 49504d28 30203a44  3 core3 (MPID: 0
95bff0c0: 0d293378 6572460a 4f545265 68745f53  x3)..FreeRTOS_th
95bff0d0: 64616572 203a305f 6c6c6568 2031206f  read_0: hello 1
95bff0e0: 656d6974 72662073 43206d6f 6574726f  times from Corte
95bff0f0: 35412d78 6f632033 20336572 49504d28  x-A53 core3 (MPI
95bff100: 30203a44 0d293378 6572460a 4f545265  D: 0x3)..FreeRTO
95bff110: 68745f53 64616572 203a305f 6c6c6568  S_thread_0: hell
95bff120: 2032206f 656d6974 72662073 43206d6f  o 2 times from C
95bff130: 6574726f 35412d78 6f632033 20336572  ortex-A53 core3
95bff140: 49504d28 30203a44 0d293378 6572460a  (MPID: 0x3)..Fre
95bff150: 4f545265 68745f53 64616572 203a305f  eRTOS_thread_0:
95bff160: 6c6c6568 2033206f 656d6974 72662073  hello 3 times fr
95bff170: 43206d6f 6574726f 35412d78 6f632033  om Cortex-A53 co
95bff180: 20336572 49504d28 30203a44 0d293378  re3 (MPID: 0x3).
95bff190: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
95bff1a0: 203a305f 6c6c6568 2034206f 656d6974  _0: hello 4 time
95bff1b0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
95bff1c0: 6f632033 20336572 49504d28 30203a44  3 core3 (MPID: 0
95bff1d0: 0d293378 6572460a 4f545265 68745f53  x3)..FreeRTOS_th
95bff1e0: 64616572 203a305f 6c6c6568 2035206f  read_0: hello 5
95bff1f0: 656d6974 72662073 43206d6f 6574726f  times from Corte
```

4. Boot SMP Linux from A53 Core0 and Core1:
   If M-Core is booting up, use `imx8mm-evk-multicore-rpmsg.dtb`

   ```
   u-boot=> setenv fdtfile imx8mm-evk-multicore-rpmsg.dtb
   ```

   Otherwise, can also use `imx8mm-evk-multicore-rtos.dtb`:

   ```
   u-boot=> setenv fdtfile imx8mm-evk-multicore-rtos.dtb
   ```

   Then, boot up the kernel using the command below:

   ```
   u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
   u-boot=> boot
   ```

   After Linux kernel boots up, check Cortex-A Core's RTOS log from the RAM console:

   ```
   root@imx8mm-lpddr4-evk:~# ram_console_dump -a 0x94bff000 -r 1
   RAM Console@0x94bff000:

   Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
   FreeRTOS_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
   FreeRTOS_thread_0: hello 4 times from Cortex-A53 core2 (MPID: 0x2)
    >
   ```

   ```
   root@imx8mm-lpddr4-evk:~# ram_console_dump -a 0x95bff000 -r 1
   RAM Console@0x95bff000:

   Cortex-A53: RTOS1: Hello world! Real-time Edge on MIMX8MM-EVK
   FreeRTOS_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
   FreeRTOS_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
   FreeRTOS_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
   FreeRTOS_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
   FreeRTOS_thread_0: hello 4 times from Cortex-A53 core3 (MPID: 0x3)
    >
   ```

- Run the Zephyr applications:

Follow the following steps to run two Zephyr instances on A53 Core2 and Core3, and run a SMP Linux kernel on A53 Core0 and Core1.

1. Boot the first Cortex-A Core's RTOS on A53 Core2 with UART4.

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

   Then, the UART4 displays the logs as follows:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mm_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 4 times from Cortex-A53 core2 (MPID: 0x2)
```

2. Boot the second Cortex-A core RTOS on A53 Core3.

```
u-boot=> ext4load mmc 1:2 0x94C00000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca53_RTOS1_RAM_CONSOLE-0x94d00000.bin
u-boot=> dcache flush; icache flush; cpu 3 release 0x94C00000
```

   Then check the RAM Console log as follows:

```
u-boot=> dcache flush; md 94d00000
94d00000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
94d00010: 94d00040 00000000 00000fbf 00000000  @...............
94d00020: 0000030c 00000000 00000000 00000000  ................
94d00030: 00000000 00000000 00000000 00000000  ................
94d00040: 202a2a2a 746f6f42 20676e69 6870655a  *** Booting Zeph
94d00050: 4f207279 75622053 20646c69 322e3476  yr OS build v4.2
94d00060: 312d302e 38333438 3766672d 34633931  .0-18438-gf719c4
94d00070: 33643262 2a203764 430a2a2a 6574726f  b2d3d7 ***.Corte
94d00080: 35412d78 52203a33 31534f54 6548203a  x-A53: RTOS1: He
94d00090: 206f6c6c 6c726f57 52202164 2d6c6165  llo World! Real-
94d000a0: 656d6974 67644520 6e6f2065 786d6920  time Edge on imx
94d000b0: 5f6d6d38 0a6b7665 6870655a 745f7279  8mm_evk.Zephyr_t
94d000c0: 61657268 3a305f64 6c656820 30206f6c  hread_0: hello 0
94d000d0: 6d697420 66207365 206d6f72 74726f43   times from Cort
94d000e0: 412d7865 63203335 3365726f 504d2820  ex-A53 core3 (MP
94d000f0: 203a4449 29337830 655a0a0d 72796870  ID: 0x3)..Zephyr
94d00100: 7268745f 5f646165 68203a30 6f6c6c65  _thread_0: hello
94d00110: 74203120 73656d69 6f726620 6f43206d   1 times from Co
94d00120: 78657472 3335412d 726f6320 28203365  rtex-A53 core3 (
94d00130: 4449504d 7830203a 0a0d2933 6870655a  MPID: 0x3)..Zeph
94d00140: 745f7279 61657268 3a305f64 6c656820  yr_thread_0: hel
94d00150: 32206f6c 6d697420 66207365 206d6f72  lo 2 times from 
94d00160: 74726f43 412d7865 63203335 3365726f  Cortex-A53 core3
94d00170: 504d2820 203a4449 29337830 655a0a0d   (MPID: 0x3)..Ze
94d00180: 72796870 7268745f 5f646165 68203a30  phyr_thread_0: h
94d00190: 6f6c6c65 74203320 73656d69 6f726620  ello 3 times fro
94d001a0: 6f43206d 78657472 3335412d 726f6320  m Cortex-A53 cor
94d001b0: 28203365 4449504d 7830203a 0a0d2933  e3 (MPID: 0x3)..
94d001c0: 6870655a 745f7279 61657268 3a305f64  Zephyr_thread_0:
94d001d0: 6c656820 34206f6c 6d697420 66207365   hello 4 times f
94d001e0: 206d6f72 74726f43 412d7865 63203335  rom Cortex-A53 c
94d001f0: 3365726f 504d2820 203a4449 29337830  ore3 (MPID: 0x3)
```

3. Boot SMP Linux from A53 Core0 and Core1:

```
u-boot=> setenv fdtfile imx8mm-evk-multicore-rtos.dtb
```

Then, boot up the kernel using the command below:

```
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

3.3.2.3.2.3 Running use cases by using remoteproc (i.MX 8M Mini)

Apart from the U-Boot commands, users can also use remoteproc under Linux to run RTOS. Take the following use case as an example. It uses the flexible `hello_world` application images to set up the following use case on i.MX 8M Mini.

Users can use remoteproc to start and stop RTOS dynamically without rebooting the board. Similarly, demo switching from use case #1 to use case #2 can be done dynamically by stopping FreeRTOS on Cortex-A Core2 and starting a Zephyr on it.

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

ID	M4	A53-Core0	A53-Core1	A53-Core2	A53-Core3
1	FreeRTOS	SMP Linux		FreeRTOS	FreeRTOS
2	FreeRTOS	SMP Linux		Zephyr	FreeRTOS

1. Boot SMP Linux from A53 Core0, Core1, Core2 and Core3:

   ```
   u-boot=> run prepare_mcore
   u-boot=> setenv fdtfile imx8mm-evk-multicore-rtos.dtb
   u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
   u-boot=> run bsp_bootcmd
   ```

   After Linux boot up, using commands `cat /proc/cpuinfo` can check that four A53 Core0, Core1, Core2 and Core3 are all used by SMP Linux.
2. Boot FreeRTOS on M4 Core
   Use remoteproc to boot FreeRTOS image `hello_world_cm4.elf` on M4 Core, it uses UART4 as debug console.

   ```
   root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-
   freertos/hello_world_cm4.elf >  /sys/devices/platform/imx8mm-cm4/remoteproc/
   remoteproc4/firmware
   root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mm-cm4/
   remoteproc/remoteproc4/state
   ```

   Then the following log is displayed from UART4:

   ```
   Cortex-M4: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
   FreeRTOS_thread_0: hello 0 times from Cortex-M4 core0 (MPID: 0x0)
   FreeRTOS_thread_0: hello 1 times from Cortex-M4 core0 (MPID: 0x0)
   FreeRTOS_thread_0: hello 2 times from Cortex-M4 core0 (MPID: 0x0)
   FreeRTOS_thread_0: hello 3 times from Cortex-M4 core0 (MPID: 0x0)
   ```
3. Boot FreeRTOS on A53 Core2
   Using the following commands to boot Zephyr image `hello_world_ca53_RTOS0_RAM_CONSOLE-0x94bff000.elf` on A53 Core2, it use RAM Console which buffer address is at `0x94bff000`:

   ```
   root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-
   freertos/hello_world_ca53_RTOS0_RAM_CONSOLE-0x94bff000.elf > /sys/devices/
   platform/remoteproc-ca53-2/remoteproc/remoteproc1/firmware
   ```

```
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

Then use the tool to dump RAM Console and log is as follows:

```
root@imx8mm-lpddr4-evk:~# ram_console_dump -a 0x94bff000 -r 1
RAM Console@0x94bff000:

Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
FreeRTOS_thread_0: hello 4 times from Cortex-A53 core2 (MPID: 0x2)
 >
```

Using commands `cat /proc/cpuinfo` can check that A53 Core0, Core1 and Core3 are used by SMP Linux as Core2 is running FreeRTOS.

4. Boot FreeRTOS on A53 Core3
   Using the following commands to boot FreeRTOS image `hello_world_ca53_RTOS1_RAM_CONSOLE-0x95bff000.elf` on A53 Core3, it use RAM Console which buffer address is at `0x95bff000`:

```
root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca53_RTOS1_RAM_CONSOLE-0x95bff000.elf >  /sys/devices/
platform/remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

Then use the tool to dump RAM Console and log is as follows:

```
root@imx8mm-lpddr4-evk:~# ram_console_dump -a 0x95bff000 -r 1
RAM Console@0x95bff000:

Cortex-A53: RTOS1: Hello world! Real-time Edge on MIMX8MM-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 4 times from Cortex-A53 core3 (MPID: 0x3)
 >
```

Using commands `cat /proc/cpuinfo` can check that A53 Core0 and Core1 are used by SMP Linux as Core2 and Core3 are running FreeRTOS.

5. Stop FreeRTOS and then start Zephyr on A53 Core2
   Stop FreeRTOS on A53 Core2:

```
root@imx8mm-lpddr4-evk:~# echo stop > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

Using commands `cat /proc/cpuinfo` can check that A53 Core0, Core1 and Core2 are used by SMP Linux as RTOS on Core2 has been stopped and Core2 is hot plugged back to Linux kernel.
Start Zephyr on A53 Core2:

```
root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca53_RTOS0_RAM_CONSOLE-0x93d00000.elf  > /sys/devices/
platform/remoteproc-ca53-2/remoteproc/remoteproc1/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

Then use the tool to dump RAM Console and log is as follows:

```
root@imx8mm-lpddr4-evk:~# ram_console_dump -a 0x93d00000 -r 1
```

```
RAM Console@0x93d00000:
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mm_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 1 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 2 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 3 times from Cortex-A53 core2 (MPID: 0x2)
Zephyr_thread_0: hello 4 times from Cortex-A53 core2 (MPID: 0x2)
 >
```

Using commands `cat /proc/cpuinfo` can check that A53 Core0 and Core1 are used by SMP Linux as Core2 and Core3 are running RTOS.

3.3.2.3.3 Running use cases on i.MX 91 9x9 QSB and i.MX 91 11x11 EVK

3.3.2.3.3.1 Overview

The following use cases can run on i.MX 91 9x9 QSB and i.MX 91 11x11 EVK platforms:

Table 29. Flexible Real-time System on i.MX 91 boards

ID	A55 Core0
0	Linux
1	RTOS

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

```
#FreeRTOS A55 hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
├── hello_world.bin
└── hello_world.elf
#Zephyr A55 hello_world Images
/examples/heterogeneous-multicore/hello-world-zephyr/
├── hello_world.bin
└── hello_world.elf
```

Refer to Section 2.6.3 for how to boot RTOS on Cortex-A core.

3.3.2.3.4 Running use cases on i.MX 93 11x11 EVK and i.MX 93 14x14 EVK

3.3.2.3.4.1 Overview

The following use cases can run on i.MX 93 11x11 EVK and i.MX 93 14x14 EVK:

Table 30. Flexible Real-time System on i.MX 93

ID	M33	A55 Core0	A55 Core1
0	RTOS	SMP Linux	
1	RTOS	Linux	RTOS
2	RTOS	RTOS	RTOS
3	RTOS	SMP RTOS	

Note: *The SMP RTOS is not supported on FreeRTOS.*

By default, RTOS of the Cortex-M core uses UART2 as debug console. Refer to Section 2.8.1 for how to boot the RTOS image of Cortex-M core.

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

```
#FreeRTOS A55 hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.bin
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.elf
├── hello_world_ca55_RTOS0_UART2.bin
├── hello_world_ca55_RTOS0_UART2.elf
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.elf
├── hello_world_ca55_RTOS1_UART1.bin
└── hello_world_ca55_RTOS1_UART1.elf
#Zephyr A55 hello_world Images
/examples/heterogeneous-multicore/hello-world-zephyr/
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0100000.bin
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0100000.elf
├── hello_world_ca55_RTOS0_SMP_RAM_CONSOLE-0xd0100000.bin
├── hello_world_ca55_RTOS0_SMP_RAM_CONSOLE-0xd0100000.elf
├── hello_world_ca55_RTOS0_SMP_UART2.bin
├── hello_world_ca55_RTOS0_SMP_UART2.elf
├── hello_world_ca55_RTOS0_UART2.bin
├── hello_world_ca55_RTOS0_UART2.elf
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1100000.bin
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1100000.elf
├── hello_world_ca55_RTOS1_UART1.bin
└── hello_world_ca55_RTOS1_UART1.elf
#FreeRTOS M33 hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
├── hello_world_cm33.bin
└── hello_world_cm33.elf
```

RAM console address is appended at the end of image name. For example `hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin` , it uses RAM Console whose buffer address is `0xd1fff000`.

Refer to Section 2.6.3 for how to boot RTOS on Cortex-A core, and refer to Section 3.3.1.3 for how to dump RAM console log.

The following chapters introduce how to use U-Boot commands or Linux remoteproc to boot FreeRTOS or Zephyr use cases.

3.3.2.3.4.2 Running use cases by using U-Boot Commands (i.MX 93 EVK)

Taking Use Case #1 as an example, it runs FreeRTOS on M33 Core, run Linux on A55 Core0 and run FreeRTOS or Zephyr on A55 Core1, below are the steps to boot multiple operating systems on the platform:

- Run the FreeRTOS applications:
  1. Boot Cortex-M Core's RTOS

     ```
     u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
     hello-world-freertos/hello_world_cm33.bin;
     u-boot=> cp.b 0xd0000000 0x201e0000 20000;
     u-boot=> bootaux 0x1ffe0000
     ```

     Then, the following log is displayed from UART2:

     ```
     Cortex-M33: RTOS0: Hello world! Real-time Edge on MIMX93-EVK
     ```

```
FreeRTOS_thread_0: hello 0 times from Cortex-M33 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M33 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M33 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M33 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 4 times from Cortex-M33 core0 (MPID: 0x0)
```

2. Boot the Cortex-A Core's RTOS on Core1

```
u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.bin
u-boot=> dcache flush; icache flush; cpu 1 release 0xd0000000
```

Then, check the RAM Console's log as follows:

```
u-boot=> dcache flush; md 0xd0fff000
d0fff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
d0fff010: d0fff040 00000000 00000fc0 00000000  @...............
d0fff020: 000002fc 00000000 00000000 00000000  ................
d0fff030: 00000000 00000000 00000000 00000000  ................
d0fff040: 6f430a0d 78657472 3535412d 5452203a  ..Cortex-A55: RT
d0fff050: 3a30534f 6c654820 77206f6c 646c726f  OS0: Hello world
d0fff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
d0fff070: 206e6f20 584d494d 452d3339 0a0d4b56   on MIMX93-EVK..
d0fff080: 65657246 534f5452 7268745f 5f646165  FreeRTOS_thread_
d0fff090: 68203a30 6f6c6c65 74203020 73656d69  0: hello 0 times
d0fff0a0: 6f726620 6f43206d 78657472 3535412d   from Cortex-A55
d0fff0b0: 726f6320 28203165 4449504d 7830203a   core1 (MPID: 0x
d0fff0c0: 29303031 72460a0d 54526565 745f534f  100)..FreeRTOS_t
d0fff0d0: 61657268 3a305f64 6c656820 31206f6c  hread_0: hello 1
d0fff0e0: 6d697420 66207365 206d6f72 74726f43   times from Cort
d0fff0f0: 412d7865 63203535 3165726f 504d2820  ex-A55 core1 (MP
d0fff100: 203a4449 30317830 0a0d2930 65657246  ID: 0x100)..Free
d0fff110: 534f5452 7268745f 5f646165 68203a30  RTOS_thread_0: h
d0fff120: 6f6c6c65 74203220 73656d69 6f726620  ello 2 times fro
d0fff130: 6f43206d 78657472 3535412d 726f6320  m Cortex-A55 cor
d0fff140: 28203165 4449504d 7830203a 29303031  e1 (MPID: 0x100)
d0fff150: 72460a0d 54526565 745f534f 61657268  ..FreeRTOS_threa
d0fff160: 3a305f64 6c656820 33206f6c 6d697420  d_0: hello 3 tim
d0fff170: 66207365 206d6f72 74726f43 412d7865  es from Cortex-A
d0fff180: 63203535 3165726f 504d2820 203a4449  55 core1 (MPID:
d0fff190: 30317830 0a0d2930 65657246 534f5452  0x100)..FreeRTOS
d0fff1a0: 7268745f 5f646165 68203a30 6f6c6c65  _thread_0: hello
d0fff1b0: 74203420 73656d69 6f726620 6f43206d   4 times from Co
d0fff1c0: 78657472 3535412d 726f6320 28203165  rtex-A55 core1 (
d0fff1d0: 4449504d 7830203a 29303031 72460a0d  MPID: 0x100)..Fr
d0fff1e0: 54526565 745f534f 61657268 3a305f64  eeRTOS_thread_0:
d0fff1f0: 6c656820 35206f6c 6d697420 66207365   hello 5 times f
```

3. Boot Linux from Cortex-A Core0:

```
# Running on i.MX 93 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rtos.dtb
# Running on i.MX 93 14x14 EVK
u-boot=> setenv fdtfile imx93-14x14-evk-multicore-rtos.dtb

u-boot=> run bsp_bootcmd
```

After Linux kernel boots up, check Cortex-A Core's RTOS log from the RAM console:

```
root@imx93evk:~# ram_console_dump -a 0xd0fff000 -r 1
RAM Console@0xd0fff000:
```

```
Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX93-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core1 (MPID: 0x100)
 >
```

- Run the Zephyr applications:
  - Follow the steps below to run the single core Zephyr instance on Cortex-A55 Core1.
    1. Boot the Cortex-A Core's RTOS on Core1 with UART2

       ```
       u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
       hello-world-zephyr/hello_world_ca55_RTOS0_UART2.bin
       u-boot=> dcache flush; icache flush; cpu 1 release 0xd0000000
       ```

       Then the UART2 shows the logs as follows:

       ```
       *** Booting Zephyr OS build v4.1.0-26144-g97b8474ac1a0 ***
       Cortex-A55: RTOS0: Hello World! Real-time Edge on imx93_evk
       Zephyr_thread_0: hello 0 times from Cortex-A55 core1 (MPID: 0x100)
       Zephyr_thread_0: hello 1 times from Cortex-A55 core1 (MPID: 0x100)
       Zephyr_thread_0: hello 2 times from Cortex-A55 core1 (MPID: 0x100)
       Zephyr_thread_0: hello 3 times from Cortex-A55 core1 (MPID: 0x100)
       Zephyr_thread_0: hello 4 times from Cortex-A55 core1 (MPID: 0x100)
       ```

    2. Boot Linux from Cortex-A Core0:

       ```
       # Running on i.MX 93 EVK
       u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rtos.dtb
       # Running on i.MX 93 14x14 EVK
       u-boot=> setenv fdtfile imx93-14x14-evk-multicore-rtos.dtb
       ```

       Then, boot up the kernel using the command below:

       ```
       u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
       u-boot=> boot
       ```

  - Follow the steps below to run the SMP Zephyr instance on all Cortex-A55 Cores.
    1. Boot the SMP Zephyr with UART2

       ```
       u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
       hello-world-zephyr/hello_world_ca55_RTOS0_SMP_UART2.bin
       u-boot=> dcache flush; icache flush; go 0xd0000000
       ```

       Then the UART2 shows the logs as follows:

       ```
       *** Booting Zephyr OS build v4.1.0-26144-g97b8474ac1a0 ***
       Secondary CPU core 1 (MPID:0x100) is up
       Cortex-A55: RTOS0: Hello World! Real-time Edge on imx93_evk
       Zephyr_thread_0: hello 0 times from Cortex-A55 core0 (MPID: 0x0)
       Zephyr_thread_1: hello 0 times from Cortex-A55 core1 (MPID: 0x100)
       Zephyr_thread_0: hello 1 times from Cortex-A55 core0 (MPID: 0x0)
       Zephyr_thread_1: hello 1 times from Cortex-A55 core1 (MPID: 0x100)
       Zephyr_thread_0: hello 2 times from Cortex-A55 core0 (MPID: 0x0)
       Zephyr_thread_1: hello 2 times from Cortex-A55 core1 (MPID: 0x100)
       Zephyr_thread_0: hello 3 times from Cortex-A55 core0 (MPID: 0x0)
       Zephyr_thread_1: hello 3 times from Cortex-A55 core1 (MPID: 0x100)
       Zephyr_thread_0: hello 4 times from Cortex-A55 core0 (MPID: 0x0)
       Zephyr_thread_1: hello 4 times from Cortex-A55 core1 (MPID: 0x100)
       ```

3.3.2.3.4.3 Running use cases by using remoteproc (i.MX 93)

In addition to using U-Boot commands, users can also use remoteproc under Linux to run RTOS. Take the following use case as an example, using the flexible `hello_world` application images to set up the following use case on i.MX 93.

Using the remoteproc allows to start and to stop the RTOS dynamically without rebooting the board. Users can switch the demo from use case #1 to use case #2 dynamically by stopping FreeRTOS on Cortex-A Core2 and start a Zephyr on it.

Table 31. Dynamic Flexible Real-time System on i.MX 93

ID	M4	A55-Core0	A55-Core1
1	FreeRTOS	Linux	FreeRTOS
2	FreeRTOS	Linux	Zephyr

1. Boot SMP Linux from A55 Core0 and Core1

```
# Running on i.MX 93 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rtos.dtb
# Running on i.MX 93 14x14 EVK
u-boot=> setenv fdtfile imx93-14x14-evk-multicore-rtos.dtb

u-boot=> run prepare_mcore
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run bsp_bootcmd
```

   After Linux boots up, using commands `cat /proc/cpuinfo` can check that the A55 Core0 and Core1 are all used by SMP Linux.
2. Boot FreeRTOS on the M33 Core
   Use `remoteproc` to boot FreeRTOS image `hello_world_cm33.elf` on M33 Core, it uses UART2 as debug console.

```
root@imx93evk:~# echo /examples/heterogeneous-multicore/hello-world-freertos/
hello_world_cm33.elf > /sys/devices/platform/remoteproc-cm33/remoteproc/
remoteproc1/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/state
```

   Then, the following log is displayed from UART2:

```
Cortex-M33: RTOS0: Hello world! Real-time Edge on MIMX93-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-M33 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M33 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M33 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M33 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 4 times from Cortex-M33 core0 (MPID: 0x0)
```

3. Boot FreeRTOS on A55 Core1
   Using the following commands to boot FreeRTOS image `hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.elf` on A55 Core1; it uses RAM Console for which the buffer address is `0xd0fff000`:

```
root@imx93evk:~# echo /examples/heterogeneous-multicore/hello-world-freertos/
hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.elf > /sys/devices/platform/
remoteproc-ca55-1/remoteproc/remoteproc0/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

Then use the tool to dump RAM Console and log is as follows:

```
root@imx93evk:~# ram_console_dump -a 0xd0fff000 -r 1
RAM Console@0xd0fff000:

Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX93-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core1 (MPID: 0x100)
 >
```

Using commands `cat /proc/cpuinfo` can check that A55 Core0 is used by Linux as Core1 is running FreeRTOS.

4. Stop FreeRTOS and then start Zephyr on A55 Core1
   Stop FreeRTOS on A55 Core1:

```
root@imx93evk:~# echo stop > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

Using commands `cat /proc/cpuinfo` can check that A55 Core0 and Core1 are used by SMP Linux as RTOS on Core1 has been stopped and Core1 is hot plugged back to Linux kernel.
Start Zephyr on A55 Core1:

```
root@imx93evk:~# echo /examples/heterogeneous-multicore/hello-world-zephyr/
hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0100000.elf  > /sys/devices/platform/
remoteproc-ca55-1/remoteproc/remoteproc0/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

Then, use the tool to dump RAM console and the log is as follows:

```
root@imx93evk:/# ram_console_dump -a 0xd0100000 -r 1
RAM Console@0xd0100000:
*** Booting Zephyr OS build v4.1.0-26144-g97b8474ac1a0 ***
Cortex-A55: RTOS0: Hello World! Real-time Edge on imx93_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core1 (MPID: 0x100)
Zephyr_thread_0: hello 1 times from Cortex-A55 core1 (MPID: 0x100)
Zephyr_thread_0: hello 2 times from Cortex-A55 core1 (MPID: 0x100)
Zephyr_thread_0: hello 3 times from Cortex-A55 core1 (MPID: 0x100)
Zephyr_thread_0: hello 4 times from Cortex-A55 core1 (MPID: 0x100)
 >
```

Using commands `cat /proc/cpuinfo` can check that A55 Core0 is used by SMP Linux as Core1 is running Zephyr.

3.3.2.3.5 Running use cases on i.MX 943 EVK

3.3.2.3.5.1 Overview

The following use cases can run on i.MX 943 15x15 EVK and i.MX 943 19x19 EVK boards:

Table 32. Flexible Real-time System on i.MX 943

ID	A55 Core0~1	A55 Core2	A55 Core3
0	SMP Linux		RTOS
1	SMP Linux	RTOS	RTOS

Table 32. Flexible Real-time System on i.MX 943*...continued*

ID	A55 Core0~1	A55 Core2	A55 Core3
2	RTOS	RTOS	RTOS

Note: *For case ID:2, U-Boot/Linux is not available to dump RAM console outputs. Therefore, you must leverage a JTAG debugger to dump the RAM console buffer memory.*

Due to the limitation of the number of Message Units, there are up to 4 SCMI agents for the A core Logical Machine in the System Manager configuration. One is assigned to the ATF, and only 3 are assigned to the OS of the A core. So there are 3 combinations shown in Table 32.

The RTOS can run with UART8 console, RAM console, or UART1 console. The following RTOS images are provided:

```
#FreeRTOS A55 hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.bin
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.elf
├── hello_world_ca55_RTOS0_UART8.bin
├── hello_world_ca55_RTOS0_UART8.elf
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.elf
├── hello_world_ca55_RTOS2_RAM_CONSOLE-0xd2fff000.bin
├── hello_world_ca55_RTOS2_RAM_CONSOLE-0xd2fff000.elf
├── hello_world_ca55_RTOS2_UART1.bin
└── hello_world_ca55_RTOS2_UART1.elf
#Zephyr A55 hello_world Images
/examples/heterogeneous-multicore/hello-world-zephyr/
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0100000.bin
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0100000.elf
├── hello_world_ca55_RTOS0_UART8.bin
├── hello_world_ca55_RTOS0_UART8.elf
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1100000.bin
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1100000.elf
├── hello_world_ca55_RTOS2_RAM_CONSOLE-0xd2100000.bin
├── hello_world_ca55_RTOS2_RAM_CONSOLE-0xd2100000.elf
├── hello_world_ca55_RTOS2_UART1.bin
└── hello_world_ca55_RTOS2_UART1.elf
```

The RAM console address is appended at the end of image name. For example:

`hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin` uses the RAM console whose buffer address is `0xd1fff000`.

- For details on how to boot RTOS on Cortex-A core refer to Section 2.6.3.
- For details on how to dump RAM Console log, refer to Section 3.3.1.3.

The following chapters introduce how to use U-Boot commands boot FreeRTOS or Zephyr use cases.

3.3.2.3.5.2 Running use cases by using U-Boot Commands

Taking Use Case #1 as an example, it runs SMP Linux on A55 Core0 ~ Core1 and runs FreeRTOS or Zephyr on A55 Core2 and Core3. Follow the below steps to boot multiple operating systems on the platform:

- Run the FreeRTOS applications:
  1. Boot the first Cortex-A Core's RTOS on Core3

     ```
     u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
     hello-world-freertos/hello_world_ca55_RTOS0_UART8.bin;
     ```

```
u-boot=> dcache flush; icache flush; cpu 3 release 0xd0000000
```

Then, the following log is displayed from UART8:

```
Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX943-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 5 times from Cortex-A55 core3 (MPID: 0x300)
```

2. Boot the second Cortex-A Core's RTOS on Core2

```
u-boot=> ext4load mmc 1:2 0xd1000000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xd1000000
```

Then, check the RAM Console's log as follows:

```
u-boot=> dcache flush; md d1fff000
d1fff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
d1fff010: d1fff040 00000000 00000fc0 00000000  @...............
d1fff020: 00000159 00000000 00000000 00000000  Y...............
d1fff030: 00000000 00000000 00000000 00000000  ................
d1fff040: 6f430a0d 78657472 3535412d 5452203a  ..Cortex-A55: RT
d1fff050: 3a31534f 6c654820 77206f6c 646c726f  OS1: Hello world
d1fff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
d1fff070: 206e6f20 584d494d 2d333439 0d4b5645   on MIMX943-EVK.
d1fff080: 6572460a 4f545265 68745f53 64616572  .FreeRTOS_thread
d1fff090: 203a305f 6c6c6568 2030206f 656d6974  _0: hello 0 time
d1fff0a0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
d1fff0b0: 6f632035 20326572 49504d28 30203a44  5 core2 (MPID: 0
d1fff0c0: 30303278 460a0d29 52656572 5f534f54  x200)..FreeRTOS_
d1fff0d0: 65726874 305f6461 6568203a 206f6c6c  thread_0: hello 
d1fff0e0: 69742031 2073656d 6d6f7266 726f4320  1 times from Cor
d1fff0f0: 2d786574 20353541 65726f63 4d282032  tex-A55 core2 (M
```

3. Boot SMP Linux:

```
u-boot=> setenv fdtfile imx943-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

After Linux kernel boots up, check Cortex-A Core's RTOS log from the RAM console:

```
root@imx943-19x19-lpddr5-evk:~# ram_console_dump -a 0xd1fff000 -r 1
RAM Console@0xd1fff000:

Cortex-A55: RTOS1: Hello world! Real-time Edge on MIMX943-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core2 (MPID: 0x200)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core2 (MPID: 0x200)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core2 (MPID: 0x200)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core2 (MPID: 0x200)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core2 (MPID: 0x200)
FreeRTOS_thread_0: hello 5 times from Cortex-A55 core2 (MPID: 0x200)
 >
```

- Run the Zephyr applications:
  1. Boot the first Cortex-A Core's RTOS on Core3

```
u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca55_RTOS0_UART8.bin
```

```
u-boot=> dcache flush; icache flush; cpu 3 release 0xd0000000
```

Then the UART3 shows the logs as follows:

```
*** Booting Zephyr OS build v4.2.0-18435-g97b8474ac1a0 ***
Cortex-A55: RTOS0: Hello World! Real-time Edge on imx943_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 1 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 2 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 3 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 4 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 5 times from Cortex-A55 core3 (MPID: 0x300)
```

2. Boot the second Cortex-A Core's RTOS on Core2

```
u-boot=> ext4load mmc 1:2 0xd1000000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1100000.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xd1000000
```

Then, check the RAM Console's log as follows:

```
u-boot=> dcache flush; md d1100000
d1100000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
d1100010: d1100040 00000000 00000fbf 00000000  @...............
d1100020: 00000144 00000000 00000000 00000000  D...............
d1100030: 00000000 00000000 00000000 00000000  ................
d1100040: 202a2a2a 746f6f42 20676e69 6870655a  *** Booting Zeph
d1100050: 4f207279 75622053 20646c69 322e3476  yr OS build v4.2
d1100060: 312d302e 38333438 3766672d 34633931  .0-18438-gf719c4
d1100070: 33643262 2a203764 430a2a2a 6574726f  b2d3d7 ***.Corte
d1100080: 35412d78 52203a35 31534f54 6548203a  x-A55: RTOS1: He
d1100090: 206f6c6c 6c726f57 52202164 2d6c6165  llo World! Real-
d11000a0: 656d6974 67644520 6e6f2065 786d6920  time Edge on imx
d11000b0: 5f333439 0a6b7665 6870655a 745f7279  943_evk.Zephyr_t
d11000c0: 61657268 3a305f64 6c656820 30206f6c  hread_0: hello 0
d11000d0: 6d697420 66207365 206d6f72 74726f43   times from Cort
d11000e0: 412d7865 63203535 3265726f 504d2820  ex-A55 core2 (MP
d11000f0: 203a4449 30327830 0a0d2930 6870655a  ID: 0x200)..Zeph
```

3. Boot SMP Linux:

```
u-boot=> setenv fdtfile imx943-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

After Linux kernel boots up, check Cortex-A Core's RTOS log from the RAM console:

```
root@imx943-19x19-lpddr5-evk:~# ram_console_dump -a 0xd1100000 -r 1
RAM Console@0xd1100000:
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A55: RTOS1: Hello World! Real-time Edge on imx943_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 1 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 2 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 3 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 4 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 5 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 6 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 7 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 8 times from Cortex-A55 core2 (MPID: 0x200)
Zephyr_thread_0: hello 9 times from Cortex-A55 core2 (MPID: 0x200)
 >
```

3.3.2.3.6 Running use cases on i.MX 95 EVK (15x15 and 19x19)

3.3.2.3.6.1 Overview

The following use cases can run on i.MX 95 15x15 EVK and i.MX 9519x19 EVK boards:

Table 33. Flexible Real-time System on i.MX 95

ID	M7	A55 Core0~3	A55 Core4	A55 Core5
0	RTOS	SMP Linux		RTOS
1	RTOS	SMP Linux	RTOS	RTOS
2	RTOS	SMP Linux	SMP RTOS	
3	RTOS	RTOS	RTOS	RTOS

Note: *For case ID:3, U-Boot/Linux is not available to dump RAM console outputs. Therefore, you must leverage a JTAG debugger to dump the RAM console buffer memory. And the SMP RTOS is not supported on FreeRTOS.*

Due to the limitation of the number of Message Units, there are up to 4 SCMI agents for the A core Logical Machine in the System Manager configuration. One is assigned to the ATF, and only 3 are assigned to the OS of the A core. So there are 3 combinations shown in Table 33.

The RTOS can run with UART3 console, RAM console, or UART1 console. The following RTOS images are provided:

```
#FreeRTOS hello_world Images
/examples/heterogeneous-multicore/hello-world-freertos/
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.bin
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.elf
├── hello_world_ca55_RTOS0_UART3.bin
├── hello_world_ca55_RTOS0_UART3.elf
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.elf
├── hello_world_ca55_RTOS2_RAM_CONSOLE-0xd2fff000.bin
├── hello_world_ca55_RTOS2_RAM_CONSOLE-0xd2fff000.elf
├── hello_world_ca55_RTOS2_UART1.bin
├── hello_world_ca55_RTOS2_UART1.elf
├── hello_world_cm7.bin
└── hello_world_cm7.elf
#Zephyr hello_world Images
/examples/heterogeneous-multicore/hello-world-zephyr/
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0100000.bin
├── hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0100000.elf
├── hello_world_ca55_RTOS0_SMP_2CORES_RAM_CONSOLE-0xd0100000.bin
├── hello_world_ca55_RTOS0_SMP_2CORES_RAM_CONSOLE-0xd0100000.elf
├── hello_world_ca55_RTOS0_SMP_2CORES_UART3.bin
├── hello_world_ca55_RTOS0_SMP_2CORES_UART3.elf
├── hello_world_ca55_RTOS0_UART3.bin
├── hello_world_ca55_RTOS0_UART3.elf
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1100000.bin
├── hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1100000.elf
├── hello_world_ca55_RTOS2_RAM_CONSOLE-0xd2100000.bin
├── hello_world_ca55_RTOS2_RAM_CONSOLE-0xd2100000.elf
├── hello_world_ca55_RTOS2_UART1.bin
└── hello_world_ca55_RTOS2_UART1.elf
```

The RAM console address is appended at the end of image name. For example:

`hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin` uses the RAM console whose buffer address is `0xd1fff000`.

- For details on how to boot RTOS on Cortex-A core refer to Section 2.6.3.
- For details on how to dump RAM Console log, refer to Section 3.3.1.3.

The following chapters introduce how to use U-Boot commands or Linux remoteproc to boot FreeRTOS or Zephyr use cases.

3.3.2.3.6.2 Running use cases by using U-Boot Commands

Taking Use Case #1 as an example, it runs SMP Linux on A55 Core0 ~ Core3 and runs FreeRTOS or Zephyr on A55 Core4 and Core5, runs FreeRTOS on M7 core. Follow the below steps to boot multiple operating systems on the platform:

- Run the FreeRTOS applications:
  1. Boot FreeRTOS on Cortex-M7 core

```
u-boot=> ext4load mmc 1:2 0x90000000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_cm7.bin;
u-boot=> cp.b 0x90000000 0x203c0000 ${filesize}
u-boot=> bootaux 0 1
```

     Then, the following log is displayed from UART3:

```
Cortex-M7: RTOS0: Hello world! Real-time Edge on MIMX95-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 4 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 5 times from Cortex-M7 core0 (MPID: 0x0)
```

  2. Boot the second Cortex-A Core's RTOS on Core5

```
u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.bin
u-boot=> dcache flush; icache flush; cpu 5 release 0xd0000000
```

     Then, check the RAM Console's log as follows:

```
u-boot=> dcache flush; md 0xd0fff000
d0fff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
d0fff010: d0fff040 00000000 00000fc0 00000000  @...............
d0fff020: 000002b6 00000000 00000000 00000000  ................
d0fff030: 00000000 00000000 00000000 00000000  ................
d0fff040: 6f430a0d 78657472 3535412d 5452203a  ..Cortex-A55: RT
d0fff050: 3a30534f 6c654820 77206f6c 646c726f  OS0: Hello world
d0fff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
d0fff070: 206e6f20 584d494d 452d3539 0a0d4b56   on MIMX95-EVK..
d0fff080: 65657246 534f5452 7268745f 5f646165  FreeRTOS_thread_
d0fff090: 68203a30 6f6c6c65 74203020 73656d69  0: hello 0 times
d0fff0a0: 6f726620 6f43206d 78657472 3535412d   from Cortex-A55
d0fff0b0: 726f6320 28203565 4449504d 7830203a   core5 (MPID: 0x
d0fff0c0: 29303035 72460a0d 54526565 745f534f  500)..FreeRTOS_t
d0fff0d0: 61657268 3a305f64 6c656820 31206f6c  hread_0: hello 1
d0fff0e0: 6d697420 66207365 206d6f72 74726f43   times from Cort
d0fff0f0: 412d7865 63203535 3565726f 504d2820  ex-A55 core5 (MP
d0fff100: 203a4449 30357830 0a0d2930 65657246  ID: 0x500)..Free
d0fff110: 534f5452 7268745f 5f646165 68203a30  RTOS_thread_0: h
```

```
d0fff120: 6f6c6c65 74203220 73656d69 6f726620  ello 2 times fro
d0fff130: 6f43206d 78657472 3535412d 726f6320  m Cortex-A55 cor
d0fff140: 28203565 4449504d 7830203a 29303035  e5 (MPID: 0x500)
d0fff150: 72460a0d 54526565 745f534f 61657268  ..FreeRTOS_threa
d0fff160: 3a305f64 6c656820 33206f6c 6d697420  d_0: hello 3 tim
d0fff170: 66207365 206d6f72 74726f43 412d7865  es from Cortex-A
d0fff180: 63203535 3565726f 504d2820 203a4449  55 core5 (MPID:
d0fff190: 30357830 0a0d2930 65657246 534f5452  0x500)..FreeRTOS
d0fff1a0: 7268745f 5f646165 68203a30 6f6c6c65  _thread_0: hello
d0fff1b0: 74203420 73656d69 6f726620 6f43206d   4 times from Co
d0fff1c0: 78657472 3535412d 726f6320 28203565  rtex-A55 core5 (
d0fff1d0: 4449504d 7830203a 29303035 72460a0d  MPID: 0x500)..Fr
d0fff1e0: 54526565 745f534f 61657268 3a305f64  eeRTOS_thread_0:
d0fff1f0: 6c656820 35206f6c 6d697420 66207365   hello 5 times f
```

3. Boot the second Cortex-A Core's RTOS on Core4

```
u-boot=> ext4load mmc 1:2 0xd1000000 /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin
u-boot=> dcache flush; icache flush; cpu 4 release 0xd1000000
```

Then, check the RAM Console's log as follows:

```
u-boot=> dcache flush; md 0xd1fff000
d1fff000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
d1fff010: d1fff040 00000000 00000fc0 00000000  @...............
d1fff020: 00000270 00000000 00000000 00000000  p...............
d1fff030: 00000000 00000000 00000000 00000000  ................
d1fff040: 6f430a0d 78657472 3535412d 5452203a  ..Cortex-A55: RT
d1fff050: 3a31534f 6c654820 77206f6c 646c726f  OS1: Hello world
d1fff060: 65522021 742d6c61 20656d69 65676445  ! Real-time Edge
d1fff070: 206e6f20 584d494d 452d3539 0a0d4b56   on MIMX95-EVK..
d1fff080: 65657246 534f5452 7268745f 5f646165  FreeRTOS_thread_
d1fff090: 68203a30 6f6c6c65 74203020 73656d69  0: hello 0 times
d1fff0a0: 6f726620 6f43206d 78657472 3535412d   from Cortex-A55
d1fff0b0: 726f6320 28203465 4449504d 7830203a   core4 (MPID: 0x
d1fff0c0: 29303034 72460a0d 54526565 745f534f  400)..FreeRTOS_t
d1fff0d0: 61657268 3a305f64 6c656820 31206f6c  hread_0: hello 1
d1fff0e0: 6d697420 66207365 206d6f72 74726f43   times from Cort
d1fff0f0: 412d7865 63203535 3465726f 504d2820  ex-A55 core4 (MP
d1fff100: 203a4449 30347830 0a0d2930 65657246  ID: 0x400)..Free
d1fff110: 534f5452 7268745f 5f646165 68203a30  RTOS_thread_0: h
d1fff120: 6f6c6c65 74203220 73656d69 6f726620  ello 2 times fro
d1fff130: 6f43206d 78657472 3535412d 726f6320  m Cortex-A55 cor
d1fff140: 28203465 4449504d 7830203a 29303034  e4 (MPID: 0x400)
d1fff150: 72460a0d 54526565 745f534f 61657268  ..FreeRTOS_threa
d1fff160: 3a305f64 6c656820 33206f6c 6d697420  d_0: hello 3 tim
d1fff170: 66207365 206d6f72 74726f43 412d7865  es from Cortex-A
d1fff180: 63203535 3465726f 504d2820 203a4449  55 core4 (MPID:
d1fff190: 30347830 0a0d2930 65657246 534f5452  0x400)..FreeRTOS
d1fff1a0: 7268745f 5f646165 68203a30 6f6c6c65  _thread_0: hello
d1fff1b0: 74203420 73656d69 6f726620 6f43206d   4 times from Co
d1fff1c0: 78657472 3535412d 726f6320 28203465  rtex-A55 core4 (
d1fff1d0: 4449504d 7830203a 29303034 72460a0d  MPID: 0x400)..Fr
d1fff1e0: 54526565 745f534f 61657268 3a305f64  eeRTOS_thread_0:
d1fff1f0: 6c656820 35206f6c 6d697420 66207365   hello 5 times f
```

4. Boot SMP Linux:

```
# For i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rtos.dtb
# For i.MX 95 19x19 EVK
```

```
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

After Linux kernel boots up, check Cortex-A Core's RTOS log from the RAM console:

```
root@imx95-19x19-lpddr5-evk:~# ram_console_dump -a 0xd0fff000 -r 1
RAM Console@0xd0fff000:

Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX95-EVK
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 5 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 6 times from Cortex-A55 core5 (MPID: 0x500)
 >
root@imx95-19x19-lpddr5-evk:~# ram_console_dump -a 0xd1fff000 -r 1
RAM Console@0xd1fff000:

Cortex-A55: RTOS1: Hello world! Real-time Edge on MIMX95-EVK
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 5 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 6 times from Cortex-A55 core4 (MPID: 0x400)
 >
```

- Run the Zephyr applications:
  1. Boot the first Cortex-A Core's RTOS on Core5

```
u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca55_RTOS0_UART3.bin
u-boot=> dcache flush; icache flush; cpu 5 release 0xd0000000
```

Then the UART3 shows the logs as follows:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A55: RTOS0: Hello World! Real-time Edge on imx95_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 1 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 2 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 3 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 4 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 5 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 6 times from Cortex-A55 core5 (MPID: 0x500)
```

  2. Boot the second Cortex-A Core's RTOS on Core4

```
u-boot=> ext4load mmc 1:2 0xd1000000 /examples/heterogeneous-multicore/
hello-world-zephyr/hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1100000.bin
u-boot=> dcache flush; icache flush; cpu 4 release 0xd1000000
```

Then, check the RAM Console's log as follows:

```
u-boot=> dcache flush; md 0xd1100000
d1100000: 5f4d4152 534e4f43 00454c4f 00000000  RAM_CONSOLE.....
d1100010: d1100040 00000000 00000fbf 00000000  @...............
d1100020: 00000253 00000000 00000000 00000000  S...............
d1100030: 00000000 00000000 00000000 00000000  ................
d1100040: 202a2a2a 746f6f42 20676e69 6870655a  *** Booting Zeph
d1100050: 4f207279 75622053 20646c69 322e3476  yr OS build v4.2
```

```
d1100060: 312d302e 38333438 3766672d 34633931  .0-18438-gf719c4
d1100070: 33643262 2a203764 430a2a2a 6574726f  b2d3d7 ***.Corte
d1100080: 35412d78 52203a35 31534f54 6548203a  x-A55: RTOS1: He
d1100090: 206f6c6c 6c726f57 52202164 2d6c6165  llo World! Real-
d11000a0: 656d6974 67644520 6e6f2065 786d6920  time Edge on imx
d11000b0: 655f3539 5a0a6b76 79687065 68745f72  95_evk.Zephyr_th
d11000c0: 64616572 203a305f 6c6c6568 2030206f  read_0: hello 0
d11000d0: 656d6974 72662073 43206d6f 6574726f  times from Corte
d11000e0: 35412d78 6f632035 20346572 49504d28  x-A55 core4 (MPI
d11000f0: 30203a44 30303478 5a0a0d29 79687065  D: 0x400)..Zephy
d1100100: 68745f72 64616572 203a305f 6c6c6568  r_thread_0: hell
d1100110: 2031206f 656d6974 72662073 43206d6f  o 1 times from C
d1100120: 6574726f 35412d78 6f632035 20346572  ortex-A55 core4
d1100130: 49504d28 30203a44 30303478 5a0a0d29  (MPID: 0x400)..Z
d1100140: 79687065 68745f72 64616572 203a305f  ephyr_thread_0:
d1100150: 6c6c6568 2032206f 656d6974 72662073  hello 2 times fr
d1100160: 43206d6f 6574726f 35412d78 6f632035  om Cortex-A55 co
d1100170: 20346572 49504d28 30203a44 30303478  re4 (MPID: 0x400
d1100180: 5a0a0d29 79687065 68745f72 64616572  )..Zephyr_thread
d1100190: 203a305f 6c6c6568 2033206f 656d6974  _0: hello 3 time
d11001a0: 72662073 43206d6f 6574726f 35412d78  s from Cortex-A5
d11001b0: 6f632035 20346572 49504d28 30203a44  5 core4 (MPID: 0
d11001c0: 30303478 5a0a0d29 79687065 68745f72  x400)..Zephyr_th
d11001d0: 64616572 203a305f 6c6c6568 2034206f  read_0: hello 4
d11001e0: 656d6974 72662073 43206d6f 6574726f  times from Corte
d11001f0: 35412d78 6f632035 20346572 49504d28  x-A55 core4 (MPI
```

3. Boot SMP Linux:

```
# For i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rtos.dtb
# For i.MX 95 19x19 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> boot
```

After Linux kernel boots up, check Cortex-A Core's RTOS log from the RAM console:

```
root@imx95-19x19-lpddr5-evk:~# ram_console_dump -a 0xd1100000 -r 1
RAM Console@0xd1100000:
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A55: RTOS1: Hello World! Real-time Edge on imx95_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_0: hello 1 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_0: hello 2 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_0: hello 3 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_0: hello 4 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_0: hello 5 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_0: hello 6 times from Cortex-A55 core4 (MPID: 0x400)
 >
```

3.3.2.3.6.3 Running use cases by using remoteproc

In addition to U-Boot commands, users can leverage the `remoteproc framework` command of Linux to dynamically manage RTOS lifecycle on the i.MX 95. For example, the flexible `hello_world` application can be deployed to demonstrate RTOS switching without requiring a board reboot.

A typical workflow involve:

1. Starting **FreeRTOS** on Cortex-M7 and 2 **FreeRTOS** instances on Cortex-A55 Core4 and Core5 respectively.

2. Stopping the 2 **FreeRTOS** instances on Cortex-A55 Core4 and Core5.
3. Launching a **SMP Zephyr RTOS** on these 2 cores.

This enables transitions between use cases (e.g., from Use Case #1 to #2 below) without rebooting by reallocating hardware resources on demand.

Table 34. Dynamically deploy Flexible Real-time System on i.MX 95

ID	M7	A55 Core0~3	A55 Core4	A55 Core5
1	FreeRTOS	SMP Linux	FreeRTOS	FreeRTOS
2	FreeRTOS	SMP Linux	SMP Zephyr RTOS	

1. Booting SMP Linux from all 6 Cortex-A55 Cores

```
# For i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rtos.dtb
# For i.MX 95 19x19 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run prepare_mcore
u-boot=> run bsp_bootcmd
```

   After Linux booting up, using commands `cat /proc/cpuinfo` can check that all the 6 Cortex-A55 Cores are all used by SMP Linux.

2. Booting FreeRTOS on the Cortex-M7 Core
   Use `remoteproc` to boot FreeRTOS `hello_world_cm7.elf`, it uses the UART3 as debug console.

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_cm7.elf > /sys/devices/platform/imx95-cm7/
remoteproc/remoteproc7/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/imx95-cm7/
remoteproc/remoteproc7/state
```

   Then, the following log is displayed from UART3:

```
Cortex-M7: RTOS0: Hello world! Real-time Edge on MIMX95-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 4 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 5 times from Cortex-M7 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 6 times from Cortex-M7 core0 (MPID: 0x0)
```

3. Booting FreeRTOS on the Cortex-A55 Core4
   Use `remoteproc` to boot FreeRTOS `hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.elf` on A55 Core4, it uses the RAM Console as debug console whose buffer address is `0xd0fff000`.

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/hello-
world-freertos/hello_world_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.elf > /sys/
devices/platform/remoteproc-ca55-4/remoteproc/remoteproc3/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/remoteproc-
ca55-4/remoteproc/remoteproc3/state
```

   Then, use the `ram_console_dump` tool to dump RAM Console:

```
root@imx95-19x19-lpddr5-evk:/# ram_console_dump -a 0xd0fff000 -r 1
RAM Console@0xd0fff000:

Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX95-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core4 (MPID: 0x400)
```

```
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 5 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 6 times from Cortex-A55 core4 (MPID: 0x400)
FreeRTOS_thread_0: hello 7 times from Cortex-A55 core4 (MPID: 0x400)
 >
```

Using command `cat /proc/cpuinfo` can check that the A55 Core4 is not online anymore under Linux.

4. Booting FreeRTOS on Cortex-A55 Core5
   Use the following commands to boot FreeRTOS `hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.elf` on A55 Core5; it uses the RAM Console whose buffer address is `0xd1fff000`:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/hello-
world-freertos/hello_world_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.elf > /sys/
devices/platform/remoteproc-ca55-5/remoteproc/remoteproc4/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/remoteproc-
ca55-5/remoteproc/remoteproc4/state
```

   Then use the `ram_console_dump` tool to dump RAM Console:

```
root@imx95-19x19-lpddr5-evk:/# ram_console_dump -a 0xd1fff000 -r 1
RAM Console@0xd1fff000:

Cortex-A55: RTOS1: Hello world! Real-time Edge on MIMX95-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 5 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 6 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 7 times from Cortex-A55 core5 (MPID: 0x500)
 >
```

   Using command `cat /proc/cpuinfo` can check that the A55 Core5 is also not online anymore under Linux.

5. Stopping the FreeRTOS on A55 Core4 and Core5, then start a Zephyr `hello_world_ca55_RTOS0_SMP_2CORES_RAM_CONSOLE-0xd0100000.elf` on these 2 cores.
   Stopping the FreeRTOS on A55 Core4 and Core5:

```
root@imx95-19x19-lpddr5-evk:~# echo stop > /sys/devices/platform/remoteproc-
ca55-4/remoteproc/remoteproc3/state
root@imx95-19x19-lpddr5-evk:~# echo stop > /sys/devices/platform/remoteproc-
ca55-5/remoteproc/remoteproc4/state
```

   Use the commands `cat /proc/cpuinfo` to check that A55 Core4 and Core5 have been released back to Linux and they are online again.
   Starting SMP Zephyr on A55 Core4 and Core5:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/hello-
world-zephyr/hello_world_ca55_RTOS0_SMP_2CORES_RAM_CONSOLE-0xd0100000.elf > /
sys/devices/platform/remoteproc-ca55-4-5/remoteproc/remoteproc5/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/remoteproc-
ca55-4-5/remoteproc/remoteproc5/state
```

   Then, the SMP Zephyr `hello_world` log can be checked using `ram_console_dump` tool:

```
root@imx95-19x19-lpddr5-evk:~# ram_console_dump -a 0xd0100000 -r 1
RAM Console@0xd0100000:
```

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Secondary CPU core 1 (MPID:0x500) is up
Cortex-A55: RTOS0: Hello World! Real-time Edge on imx95_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_1: hello 0 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 1 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_1: hello 1 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 2 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_1: hello 2 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 3 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_1: hello 3 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 4 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_1: hello 4 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 5 times from Cortex-A55 core4 (MPID: 0x400)
Zephyr_thread_1: hello 5 times from Cortex-A55 core5 (MPID: 0x500)
```

Use the command `cat /proc/cpuinfo` to check that the A55 Core4 and Core5 are not online anymore under Linux again.

3.3.3 lwIP Networking Stack

3.3.3.1 Overview

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

3.3.3.2 Running lwIP Application

3.3.3.2.1 Overview

Heterogeneous Multicore Framework provides a **lwIP Ping application** in the repository heterogeneous-multicore, which supports i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, and i.MX 93 EVK, and i.MX 93 14x14 EVK.

The Heterogeneous-multicore LWIP Ping application runs on Cortex-A Core. It initializes the LWIP networking stack and configures ENET port with the default IP address “192.168.0.100” and default gateway address “192.168.0.254”. Then ping the gateway.

Hardware Setup

- Connect the Ethernet port of a test board to another board using an Ethernet cable and configure the other board's Ethernet interface with the IP address "192.168.0.254".
- Run the lwIP Ping application.
  Use U-Boot commands or `remoteproc` under Linux to start the application.

3.3.3.2.2 Running lwIP Networking Stack use cases by using U-Boot Commands

Follow the following steps to start the use cases by using U-Boot commands:

- **Running the application on i.MX 8M Mini LPDDR4 EVK board**
  Boot up i.MX 8M Mini LPDDR4 EVK board, and boot lwIP application from U-Boot command line:

```
u-boot=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/lwip-
ping-freertos/lwip_ping_ca53.bin
u-boot=> dcache flush; icache flush; cpu 3 release 0x93C00000
```

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

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

- **Running the application on i.MX 93 EVK and i.MX 93 14x14 EVK boards:**
  Boot up i.MX 93 EVK board or i.MX 93 14x14 EVK, and boot lwIP application from U-Boot command line:

  ```
  u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/lwip-
  ping-freertos/lwip_ping_ca55.bin
  u-boot=> dcache flush; icache flush; cpu 1 release 0xD0000000
  ```

  Then, the following log is displayed on the FreeRTOS Console:

  ```
  Initializing PHY...



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


  ping: recv
  192.168.0.254
   0 ms

  ping: send
  192.168.0.254
  ```

3.3.3.2.3 Running lwIP Networking Stack use cases using remoteproc

Follow the following steps to start the use cases by using remoteproc under Linux:

1. Boot up Linux
   - For i.MX 8M Mini LPDDR4 EVK

     ```
     u-boot=> setenv fdtfile imx8mm-evk-virtio-net-ca53.dtb
     u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
     u-boot=> run bsp_bootcmd
     ```

   - For i.MX 8M Plus LPDDR4 EVK

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

   - For i.MX 93 EVK

     ```
     u-boot=> setenv fdtfile imx93-11x11-evk-virtio-net-ca55.dtb
     u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
     u-boot=> run bsp_bootcmd
     ```

- For i.MX 93 14x14 EVK

```
u-boot=> setenv fdtfile imx93-14x14-evk-virtio-net-ca55.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run bsp_bootcmd
```

2. Run the application
   - For i.MX 8M Mini LPDDR4 EVK

```
root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/lwip-ping-
freertos/lwip_ping_ca53.elf  >  /sys/devices/platform/remoteproc-ca53-3/
remoteproc/remoteproc2/firmware
root@imx8mm-lpddr4-evk:~# echo start >  /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

   - For i.MX 8M Plus LPDDR4 EVK

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/lwip-ping-
freertos/lwip_ping_ca53.elf  >  /sys/devices/platform/remoteproc-ca53-3/
remoteproc/remoteproc2/firmware
root@imx8mp-lpddr4-evk:~# echo start >  /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

   - For i.MX93 EVK and i.MX 93 14x14 EVK

```
root@imx93evk:~# echo /examples/heterogeneous-multicore/lwip-ping-freertos/
lwip_ping_ca55.elf  > /sys/devices/platform/remoteproc-ca55-1/remoteproc/
remoteproc0/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

Then, the following log is displayed on the FreeRTOS Console:

```
Initializing PHY...



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


ping: recv
192.168.0.254
 0 ms

ping: send
192.168.0.254
```

3.4 RPMSG data communication

3.4.1 Overview

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

Currently Real-time Edge supports the following Heterogeneous AMP:

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

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

3.4.2 RPMSG performance evaluation

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

Note: *The released image does not display the CPU load status of the RTOS side during the evaluation. To display the CPU load status, build the debug type image manually referring to the Section 3.2.2.1.*

3.4.2.1 Building RPMSG performance evaluation application

RPMSG performance evaluation application "rpmsg_perf" runs on FreeRTOS, it is an application in the repo: heterogeneous-multicore.

Refer to "Section 3.2 Building Heterogeneous Multicore RTOS Application" for how to build RPMSG performance evaluation application.

3.4.2.2 Running RPMsg performance application

The FreeRTOS RPMsg remote can run on Cortex-M core or Cortex-A core to test the RPMsg performance between Cortex-A core and Cortex-M core or between 2 Cortex-A cores. Follow the steps below to run the application on i.MX 8M Plus LPDDR4 EVK:

1. Boot FreeRTOS and Linux
   Use U-Boot commands or remoteproc to boot FreeRTOS on Cortex-A Core or Cortex-M Core
   - Steps for using U-Boot commmands to boot FreeRTOS, then start Linux
     a. For FreeRTOS RPMsg remote on Cortex-M Core

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

     Then boot up Linux:

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

     b. Or for FreeRTOS RPMsg remote on Cortex-A core

```
u-boot=> load mmc 1:2 c0000000 /examples/heterogeneous-multicore/rpmsg-
perf-freertos/rpmsg_perf_ca53.bin
u-boot=> dcache flush; icache flush
u-boot=> cpu 2 release c0000000
```

Then boot up Linux:

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

After Linux boot up, FreeRTOS shows RPMSG links up as following log:

```
Cortex-M7/A53: RPMsg performance with linux:
INFO: rpmsg_init              : RPMSG remote init ...
INFO: rpmsg_remote_init       : waiting for link establish ...
INFO: rpmsg_remote_init       : RPMSG link up
```

- Steps for using remoteproc under Linux to boot FreeRTOS
  a. For FreeRTOS RPMsg remote on Cortex-M Core
     Boot up Linux firstly:

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

     Then run FreeRTOS on Cortex-M Core:

```
root@imx8mp-lpddr4-evk:/# echo /examples/heterogeneous-multicore/rpmsg-
perf-freertos/rpmsg_perf_cm7.elf > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc0/firmware
root@imx8mp-lpddr4-evk:/# echo start > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc0/state
```

  b. Or for FreeRTOS RPMsg remote on Cortex-A core
     Boot up Linux firstly:

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

     Then run FreeRTOS on Cortex-A Core:

```
root@imx8mp-lpddr4-evk:/# echo /examples/heterogeneous-multicore/rpmsg-
perf-freertos/rpmsg_perf_ca53.elf > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/firmware
root@imx8mp-lpddr4-evk:/# echo start > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

  FreeRTOS shows RPMSG links up as following log:

```
Cortex-M7/A53: RPMsg performance with linux:
INFO: rpmsg_init              : RPMSG remote init ...
INFO: rpmsg_remote_init       : waiting for link establish ...
INFO: rpmsg_remote_init       : RPMSG link up
```

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

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

3. Use the `rpmsg_perf` tool to test the RPMsg performance in Linux prompt, refer to the usage below:

```
root@imx8mp-lpddr4-evk:~# rpmsg_perf
usage: rpmsg_perf <dev> <as_sender> <no_copy> <packet_size> <test_time>
```

```
        dev: specify rpmsg device, see /dev/rpmsg-perf<x>
        as_sender: true for as_sender, false for as_receiver
        no_copy: specify if use no_copy version API in remote side
        packet_size: specify the packet size, the MAX value is 496
        test_time: specify the test period in unit second
        such as: rpmsg_perf /dev/rpmsg-perf<x> true true 64 60
```

The example runs `rpmsg_perf /dev/rpmsg-perf30 true true 64 60` in the usage:

```
root@imx8mp-lpddr4-evk:~# rpmsg_perf /dev/rpmsg-perf30 true true 64 60
[ 1643.799911] rpmsg_perf: packet size: 64, sent packets: 4075370, time: 60
 s, rate: 67 kpps
```

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

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

Figure 29 shows RPMSG communication between RTOS running on Cortex-M core and Linux running Cortex-A core.

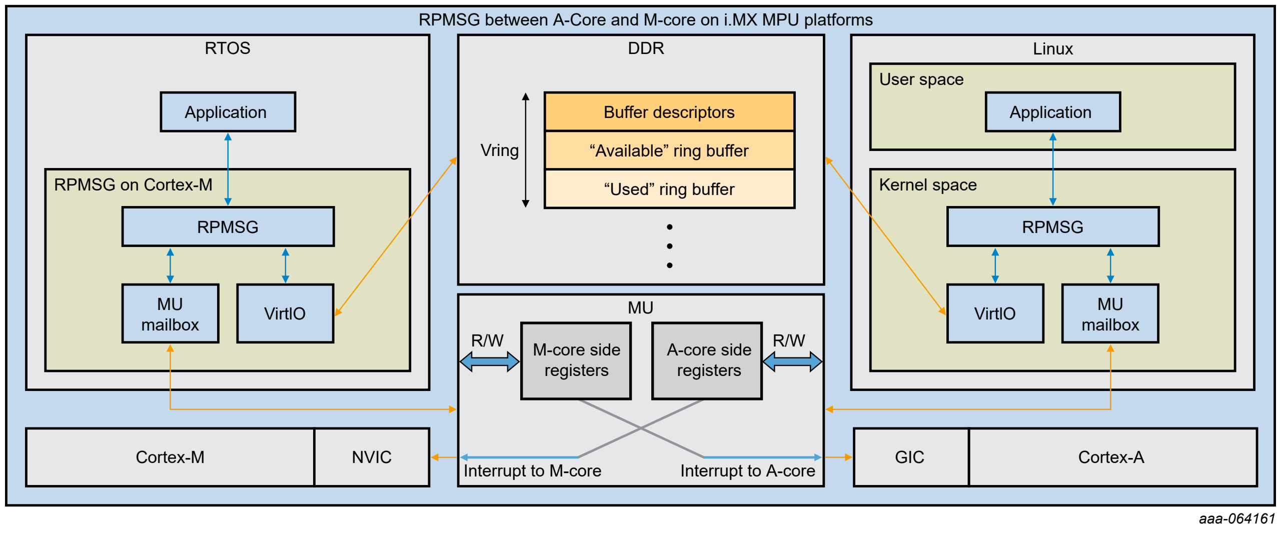


Figure 29. RPMSG between Cortex-A core and Cortex-M core

On i.MX MPU platforms, RPMSG builds virtual queue by using the Vring of VirtIO in the shared memory of DDR. MU (Message Unit) is a hardware component in the MPU platform that provides inter-core interrupts between the Cortex-M and the Cortex-A cores. RPMG uses MU as a mailbox notification mechanism.

In Linux, RPMSG communication is based on VirtIO driver and MU mailbox drivers.

On RTOS side, the RPMSG is supported both on Zephyr and FreeRTOS. On Zephyr, it uses the RPMSG implementation in OpenAMP. And on FreeRTOS, it uses the RPMsg-Lite, which is a lightweight implementation of the RPMSG protocol and an open-source component developed by NXP Semiconductors.

Details about RPMsg-Lite can be found in the RPMsg-Lite User's Guide.

3.4.3.1 RPMSG between Cortex-A Linux and Cortex-M RTOS

3.4.3.1.1 Running FreeRTOS Cortex-M rpmsg_str_echo demo

- Setup UART Console

Refer to the steps described in **Setup UART console** in the Section Using U-Boot Commands to Manage Life Cycle of RTOS on Cortex-A Core.

- Start RTOS RPMSG example and Linux
  The RTOS can be booted up by U-Boot commands or Linux remoteproc:
  - **Using U-Boot commands**
    - For i.MX 8M Mini LPDDR4 EVK
      Start up `FreeRTOS` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_cm4.bin
u-boot=> cp.b 0x48000000 0x7e0000 ${filesize};
u-boot=> bootaux 0x7e0000
```

Boot up Linux with RPMSG DTB:

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

- For i.MX 8M Plus LPDDR4 EVK
  Start up `FreeRTOS` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_cm7.bin
u-boot=> cp.b 0x48000000 0x7e0000 ${filesize};
u-boot=> bootaux 0x7e0000
```

Boot up Linux with RPMSG DTB:

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

- Or **using Linux remoteproc**
  - For i.MX 8M Mini LPDDR4 EVK
    Boot up Linux with RPMSG DTB:

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

Start up `FreeRTOS` RPMSG example:

```
root@imx8mm-lpddr4-evk:/# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_cm4.elf > /sys/devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/state
```

- For i.MX 8M Plus LPDDR4 EVK
  Boot up Linux with RPMSG DTB:

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

Start up `FreeRTOS` RPMSG example:

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_cm7.elf > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/state
```

After the `FreeRTOS` application started, the RTOS console outputs is as follows (taking i.MX 8M Plus EVK as example):

```
Cortex-A53: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
RTOS1: RAM console@0xc1fff000
INFO: rpmsg_init             : RPMSG remote init ...
INFO: rpmsg_remote_init      : waiting for link establish ...
INFO: rpmsg_remote_init      : RPMSG link up
INFO: rpmsg_create_ept       : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
INFO: rpmsg_create_ept       : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
INFO: rpmsg_create_ept       : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
```

- Run the demo

1. After Linux boots up, load `imx_rpmsg_tty.ko`, it creates 3 ttyRPMSG under `/dev`

```
root@imx8mp-lpddr4-evk:~# modprobe imx_rpmsg_tty
root@imx8mp-lpddr4-evk:~# ls /dev/ttyRPMSG*
/dev/ttyRPMSG0  /dev/ttyRPMSG1  /dev/ttyRPMSG2
```

2. Use `minicom` to open a console connecting one of the device `ttyRPMSG` on the Linux prompt as shown below:

```
root@imx8mp-lpddr4-evk:~# minicom -D /dev/ttyRPMSG0
```

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

3.4.3.1.2 Running Zephyr Cortex-M rpmsg_str_echo demo

- Setup UART Console
  Refer to the steps described in **Setup UART console** in the Section Using U-Boot Commands to Manage Life Cycle of RTOS on Cortex-A Core.
- Start RTOS RPMSG example and Linux
  The RTOS can be booted up by U-Boot commands or Linux remoteproc:
  – Using U-Boot commands
  – For i.MX 93 11x11 and 14x14 EVK
  Start up `Zephyr` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0x90000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-zephyr/rpmsg_str_echo_cm33.bin
u-boot=> cp.b 0x90000000 0x201e0000 ${filesize}
u-boot=> bootaux 0x1ffe0000 0
```

Boot up Linux with RPMSG DTB:

```
# Running on i.MX 93 11x11 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rpmsg.dtb
# Running on i.MX 93 14x14 EVK
u-boot=> setenv fdtfile imx93-14x14-evk-multicore-rpmsg.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
```

```
u-boot=> run bsp_bootcmd
```

– For i.MX 95 15x15 and 19x19 EVK
  Start up `Zephyr` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0x90000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-zephyr/rpmsg_str_echo_cm7.bin
u-boot=> cp.b 0x90000000 0x203c0000 ${filesize}
u-boot=> bootaux 0 1
```

  Boot up Linux with RPMSG DTB:

```
# Running on i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rpmsg.dtb
# Running on i.MX 95 19x19 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rpmsg.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run bsp_bootcmd
```

– Or **using Linux remoteproc**

– For i.MX 93 11x11 and 14x14 EVK
  Boot up Linux with RPMSG DTB:

```
# Running on i.MX 93 11x11 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rpmsg.dtb
# Running on i.MX 93 14x14 EVK
u-boot=> setenv fdtfile imx93-14x14-evk-multicore-rpmsg.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run prepare_mcore
u-boot=> run bsp_bootcmd
```

  Start up `Zephyr` RPMSG example:

```
root@imx93evk:~# echo /examples/heterogeneous-multicore/rpmsg-str-echo-
zephyr/rpmsg_str_echo_cm33.elf > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/state
```

– For i.MX 95 15x15 and 19x19 EVK
  Boot up Linux with RPMSG DTB:

```
# Running on i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rpmsg.dtb
# Running on i.MX 95 19x19 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rpmsg.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run prepare_mcore
u-boot=> run bsp_bootcmd
```

  Start up `Zephyr` RPMSG example:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
rpmsg-str-echo-zephyr/rpmsg_str_echo_cm7.elf > /sys/devices/platform/imx95-
cm7/remoteproc/remoteproc7/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/imx95-
cm7/remoteproc/remoteproc7/state
```

After the `Zephyr` application started, the RTOS console shows (taking i.MX 95 EVK as example) :

```
*** Booting Zephyr OS build v4.1.0-26144-g97b8474ac1a0 ***

Cortex-M7: RTOS0: Multiple Endpoints RPMsg String Echo Zephyr Demo
```

- Run the demo

1. After Linux boots up, load `imx_rpmsg_tty.ko`, it creates 3 ttyRPMSG under `/dev`

```
root@imx95-19x19-lpddr5-evk:~# modprobe imx_rpmsg_tty
root@imx95-19x19-lpddr5-evk:~# ls /dev/ttyRPMSG*
/dev/ttyRPMSG0  /dev/ttyRPMSG1  /dev/ttyRPMSG2
```

2. Use `minicom` to open a console connecting one of the device `ttyRPMSG` on the Linux prompt as shown below:

```
root@imx95-19x19-lpddr5-evk:~# minicom -D /dev/ttyRPMSG0
```

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

3.4.3.2 RPMSG with enhanced 8MB Vring buffer

3.4.3.2.1 RPMSG merits

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

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

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

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

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

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

2. Build the image using the commands below:

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

3. Program the full SD card image. For this, use SD card with capacity of at least 4 GB.

```
$ bzip2 -d -c nxp-image-real-time-edge-imx8mm-lpddr4-evk.wic.bz2 | pv | sudo
 dd of=/dev/sdx bs=1M && sync
# Note: find the right SD Card device name in your host machine and replace
 the “sdx”.
```

4. Start up M-core firmware under U-Boot:

```
=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_cm4_8m_buf.bin
=> cp.b 0x48000000 0x7e0000 0x20000
=> bootaux 0x7e0000
```

5. Boot up Linux with RPMSG DTB:

```
=> setenv fdtfile imx8mm-evk-rpmsg-8m-buf.dtb
=> run bsp_bootcmd
```

6. After Linux boots up, load `imx_rpmsg_tty.ko`

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

   Linux `imx_rpmsg_tty` driver sends a “hello world!” message when probed, and it is displayed on the FreeRTOS console.
7. Test string transmitting through device “`ttyRPMSG30`” from Linux prompt, the FreeRTOS console displays the received string. For example execute the following command:

   ```
   root@imx8mm-lpddr4-evk:~# stty -F /dev/ttyRPMSG30 -echo
   root@imx8mm-lpddr4-evk:~# echo “any-string” > /dev/ttyRPMSG30
   ```
8. In this demo, the single RPMSG buffer size is 1024B and the RPMSG header overhead is 16B. Therefore the transmitting string is split into up to 1008B fragments. Use the following commands to generate a file larger than 1KB to verify:

   ```
   root@imx8mm-lpddr4-evk:~# for i in {1..300}; do echo -n `seq -s "" 0 1 9` >>
    num.txt; done
   root@imx8mm-lpddr4-evk:~# echo `cat num.txt` > /dev/ttyRPMSG30
   ```

   The log displays the message shown below:

   ```
   Cortex-M4: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
   INFO: rpmsg_init             : RPMSG remote init ...
   INFO: rpmsg_remote_init      : waiting for link establish ...
   INFO: rpmsg_remote_init      : RPMSG link up
   INFO: rpmsg_create_ept       : Sending Nameservice string: rpmsg-virtual-tty-channel-1
   ept30: Get Message From Master Side : "hello world!" [len : 12]
   ept30: Get Message From Master Side :
    "01234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
   12345678901234567890123456789012345678901234567" [len : 1008]
   ept30: Get Message From Master Side : "8" [len : 1]
   ept30: Get Message From Master Side :
    "9012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
   01234567890123456789012345678901234567890123456" [len : 1008]
   ept30: Get Message From Master Side : "7" [len : 1]
   ept30: Get Message From Master Side : "890123456789012345678901234567" [len : 30]
   ept30: Get Message From Master Side :
    "8901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
   9012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789" [len :
   952]
   ept30: Get New Line From Master Side
   ```

3.4.4 RPMSG between Cortex-A Core and Cortex-A Core

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

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

There is not MU hardware mailbox that can be used between different Cortex-A cores. Therefore, a Generic Software mailbox is created for message notification between Cortex-A cores. The Generic Software mailbox uses shared memory to simulate MMIO registers that are used by the mailbox driver. Two unused SPI interrupts in GIC are used as notification interrupts between Cortex-A cores.

The RPMSG is supported both on FreeRTOS and Zephyr, on FreeRTOS the RPMSG-Lite is integrated and on Zephyr it leverages the RPMSG provided by OpenAMP.

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

Figure 30 illustrates the software setup for RPMSG between Linux on Cortex-A Core and RTOS on Cortex-A Core.

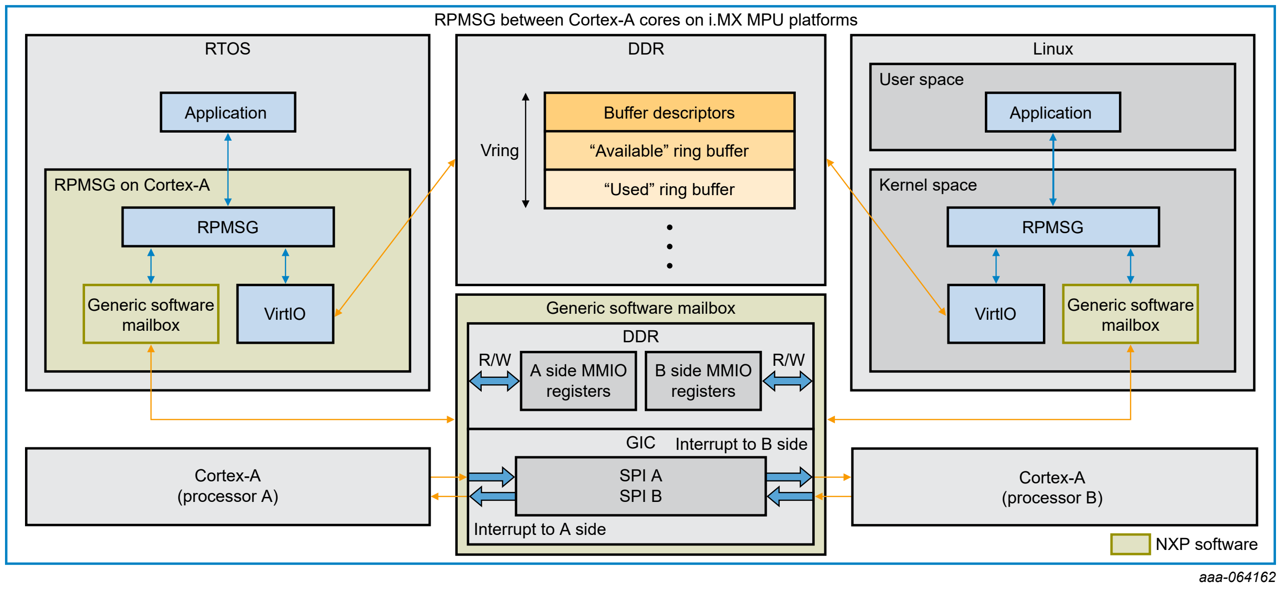


Figure 30. RPMSG between Linux and RTOS on different Cortex-A cores

3.4.4.1.1 Building the rpmsg_str_echo demo

RPMSG "rpmsg_str_echo" application runs on FreeRTOS, it is an application named in the repo: heterogeneous-multicore.

Refer to " Building Heterogeneous Multicore RTOS Application" for how to build RPMSG performance evaluation application.

3.4.4.1.2 Running the rpmsg_str_echo demo

- Setup UART Console

Refer to the steps described in **Setup UART console** in the Section Using U-Boot Commands to Manage Life Cycle of RTOS on Cortex-A Core.

- Start RTOS RPMSG example and Linux
  The RTOS can be booted up by U-Boot commands or Linux remoteproc:
  – Using U-Boot commands
  - For i.MX 8M Mini LPDDR4 EVK
    To start up `FreeRTOS` RPMSG example:

```
u-boot=> ext4load mmc 1:2 93c00000 /examples/heterogeneous-multicore/rpmsg-
str-echo-freertos/rpmsg_str_echo_ca53_RTOS0_UART4.bin
u-boot=> dcache flush; icache flush;
u-boot=> cpu 2 release 93c00000
```

    Boot up Linux with RPMSG DTB:

```
u-boot=> setenv fdtfile imx8mm-evk-multicore-rpmsg.dtb
u-boot=> run bsp_bootcmd
```

  - For i.MX 8M Plus LPDDR4 EVK
    To start up `FreeRTOS` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_ca53_RTOS0_UART4.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
```

    To start up `Zephyr` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-zephyr/rpmsg_str_echo_ca53_RTOS0_UART4.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
```

    Boot up Linux with RPMSG DTB:

```
u-boot=> setenv fdtfile imx8mp-evk-multicore-rpmsg.dtb
u-boot=> run bsp_bootcmd
```

  - For i.MX 93 11x11 and 14x14 EVK
    To start up `FreeRTOS` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_ca55_RTOS0_UART2.bin
u-boot=> dcache flush; icache flush; cpu 1 release 0xD0000000
```

    To start up `Zephyr` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-zephyr/rpmsg_str_echo_ca55_RTOS0_UART2.bin
u-boot=> dcache flush; icache flush; cpu 1 release 0xD0000000
```

    Boot up Linux with RPMSG DTB:

```
# Running on i.MX 93 11x11 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rpmsg.dtb
# Running on i.MX 93 14x14 EVK
u-boot=> setenv fdtfile imx93-14x14-evk-multicore-rpmsg.dtb
u-boot=> run bsp_bootcmd
```

  - For i.MX 95 15x15 and 19x19 EVK
    To start up `FreeRTOS` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_ca55_RTOS0_UART3.bin
u-boot=> dcache flush; icache flush; cpu 4 release 0xD0000000
```

To start up `Zephyr` RPMSG example:

```
u-boot=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-zephyr/rpmsg_str_echo_ca55_RTOS0_UART3.bin
u-boot=> dcache flush; icache flush; cpu 4 release 0xD0000000
```

Boot up Linux with RPMSG DTB:

```
# Running on i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rpmsg.dtb
# Running on i.MX 95 19x19 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rpmsg.dtb
u-boot=> run bsp_bootcmd
```

– Or **using Linux remoteproc**

– For i.MX 8M Mini LPDDR4 EVK
Boot up Linux with RPMSG DTB:

```
u-boot=> setenv fdtfile imx8mm-evk-multicore-rpmsg.dtb
u-boot=> run bsp_bootcmd
```

To start up `FreeRTOS` RPMSG example:

```
root@imx8mm-lpddr4-evk:/# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_ca53_RTOS0_UART4.elf > /sys/devices/platform/
remoteproc-ca53-2/remoteproc/remoteproc1/firmware
root@imx8mm-lpddr4-evk:/# echo start > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

– For i.MX 8M Plus LPDDR4 EVK
Boot up Linux with RPMSG DTB:

```
u-boot=> setenv fdtfile imx8mp-evk-multicore-rpmsg.dtb
u-boot=> run bsp_bootcmd
```

To start up `FreeRTOS` RPMSG example:

```
root@imx8mp-lpddr4-evk:/# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_ca53_RTOS0_UART4.elf > /sys/devices/platform/
remoteproc-ca53-2/remoteproc/remoteproc1/firmware
root@imx8mp-lpddr4-evk:/# echo start > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

To start up `Zephyr` RPMSG example:

```
root@imx8mp-lpddr4-evk:/# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-zephyr/rpmsg_str_echo_ca53_RTOS0_UART4.elf > /sys/devices/platform/
remoteproc-ca53-2/remoteproc/remoteproc1/firmware
root@imx8mp-lpddr4-evk:/# echo start > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

– For i.MX 93 11x11 and 14x14 EVK
Boot up Linux with RPMSG DTB:

```
# Running on i.MX 93 11x11 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rpmsg.dtb
# Running on i.MX 93 14x14 EVK
u-boot=> setenv fdtfile imx93-14x14-evk-multicore-rpmsg.dtb
u-boot=> run bsp_bootcmd
```

To start up `FreeRTOS` RPMSG example:

```
root@imx93evk:/# echo /examples/heterogeneous-multicore/rpmsg-str-echo-
freertos/rpmsg_str_echo_ca55_RTOS0_UART2.elf > /sys/devices/platform/
remoteproc-ca55-1/remoteproc/remoteproc0/firmware
root@imx93evk:/# echo start > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

To start up `Zephyr` RPMSG example:

```
root@imx93evk:/# echo /examples/heterogeneous-multicore/rpmsg-str-echo-
zephyr/rpmsg_str_echo_ca55_RTOS0_UART2.elf > /sys/devices/platform/
remoteproc-ca55-1/remoteproc/remoteproc0/firmware
root@imx93evk:/# echo start > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

– For i.MX 95 15x15 and 19x19 EVK
  Boot up Linux with RPMSG DTB:

```
# Running on i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rpmsg.dtb
# Running on i.MX 95 19x19 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rpmsg.dtb
u-boot=> run bsp_bootcmd
```

To start up `FreeRTOS` RPMSG example:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_ca55_RTOS0_UART3.elf > /sys/devices/
platform/remoteproc-ca55-4/remoteproc/remoteproc3/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/
remoteproc-ca55-4/remoteproc/remoteproc3/state
```

To start up `Zephyr` RPMSG example:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
rpmsg-str-echo-zephyr/rpmsg_str_echo_ca55_RTOS0_UART3.elf > /sys/devices/
platform/remoteproc-ca55-4/remoteproc/remoteproc3/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/
remoteproc-ca55-4/remoteproc/remoteproc3/state
```

If started the `FreeRTOS` application, then the RTOS console outputs is as follows (taking i.MX 8M Plus EVK as example):

```
Cortex-A53: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
RTOS1: RAM console@0xc1fff000
INFO: rpmsg_init            : RPMSG remote init ...
INFO: rpmsg_remote_init     : waiting for link establish ...
INFO: rpmsg_remote_init     : RPMSG link up
INFO: rpmsg_create_ept      : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
INFO: rpmsg_create_ept      : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
INFO: rpmsg_create_ept      : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
```

And if started the `Zephyr` application, the RTOS console shows (taking i.MX 8M Plus EVK as example) :

```
*** Booting Zephyr OS build v4.1.0-39-gb2ab2031bad8 ***

Cortex-A53: RTOS0: Multiple Endpoints RPMsg String Echo Zephyr Demo
```

• Run the demo

1. After Linux boots up, load `imx_rpmsg_tty.ko`, it creates 3 ttyRPMSG under `/dev`

```
root@imx8mm-lpddr4-evk:~# modprobe imx_rpmsg_tty
root@imx8mm-lpddr4-evk:~# ls /dev/ttyRPMSG*
/dev/ttyRPMSG3  /dev/ttyRPMSG4  /dev/ttyRPMSG5
```

2. Use minicom to open a console connecting one of the device `ttyRPMSG` on the Linux prompt as shown below:

```
root@imx8mm-lpddr4-evk:~# minicom -D /dev/ttyRPMSG3
```

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

3.4.4.2 RPMSG between Cortex-A RTOS and Cortex-A RTOS

Figure 31 illustrates the software setup for RPMSG between different RTOS on different Cortex-A Core.

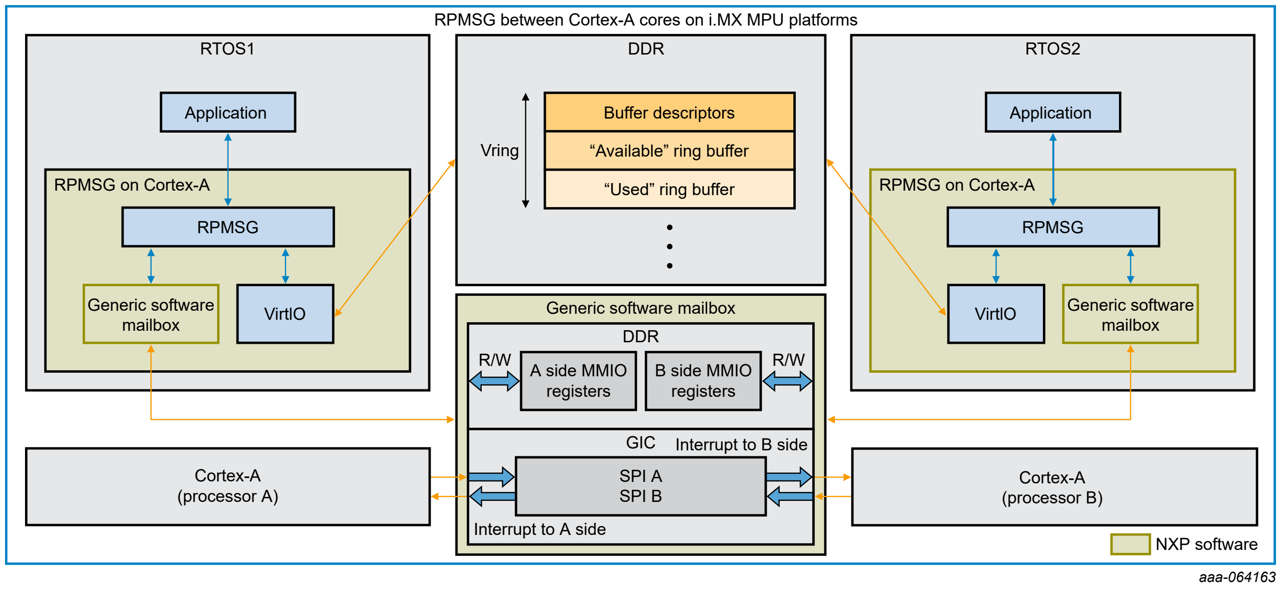


Figure 31. RPMSG between RTOS and RTOS on different Cortex-A cores

3.4.4.2.1 Running the RPMSG ping-pong application

The RPMSG ping-pong application demonstrates the RPMSG communication between two FreeRTOS systems running respectively on two Cortex-A core islands.

Can use U-Boot commands under U-Boot or remoteproc under Linux to start FreeRTOS, choose one the method:

- Using U-Boot Commands to start both the RPMSG master and remote RTOS instances
  – Follow the steps below to run the demo on i.MX 8M Plus LPDDR4 EVK:
  1. First, boot the remote peer FreeRTOS on core2:

```
u-boot=> load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/rpmsg-
pingpong-freertos/rpmsg_pingpong_remote_ca53_RTOS0_UART4.bin
u-boot=> dcache flush; icache flush;
u-boot=> cpu 2 release 0xC0000000
```

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

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

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

```
u-boot=> load mmc 1:2 0xC1000000 /examples/
heterogeneous-multicore/rpmsg-pingpong-freertos/
rpmsg_pingpong_master_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.bin
u-boot=> dcache flush; icache flush;
u-boot=> cpu 3 release 0xC1000000
```

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

```
RPMsg Ping-Pong FreeRTOS Demo: remote: running at 0xc0000000
master RAM_CONSOLE at 0xc1fff000
INFO: rpmsg_init             : RPMSG remote init ...
INFO: rpmsg_remote_init      : waiting for link establish ...
INFO: rpmsg_remote_init      : RPMSG link up
Waiting for ping ...
Received ping (0) ... sending pong (1)
Received ping (2) ... sending pong (3)
Received ping (4) ... sending pong (5)
Received ping (6) ... sending pong (7)
Received ping (8) ... sending pong (9)
Received ping (10) ... sending pong (11)
Received ping (12) ... sending pong (13)
Received ping (14) ... sending pong (15)
Received ping (16) ... sending pong (17)
Received ping (18) ... sending pong (19)
Received ping (20) ... sending pong (21)
Received ping (22) ... sending pong (23)
Received ping (24) ... sending pong (25)
Received ping (26) ... sending pong (27)
Received ping (28) ... sending pong (29)
Received ping (30) ... sending pong (31)
Received ping (32) ... sending pong (33)
Received ping (34) ... sending pong (35)
Received ping (36) ... sending pong (37)
Received ping (38) ... sending pong (39)
Received ping (40) ... sending pong (41)
Received ping (42) ... sending pong (43)
Received ping (44) ... sending pong (45)
Received ping (46) ... sending pong (47)
Received ping (48) ... sending pong (49)
Received ping (50) ... sending pong (51)
Received ping (52) ... sending pong (53)
Received ping (54) ... sending pong (55)
Received ping (56) ... sending pong (57)
Received ping (58) ... sending pong (59)
Received ping (60) ... sending pong (61)
Received ping (62) ... sending pong (63)
Received ping (64) ... sending pong (65)
Received ping (66) ... sending pong (67)
Received ping (68) ... sending pong (69)
Received ping (70) ... sending pong (71)
Received ping (72) ... sending pong (73)
Received ping (74) ... sending pong (75)
Received ping (76) ... sending pong (77)
Received ping (78) ... sending pong (79)
Received ping (80) ... sending pong (81)
Received ping (82) ... sending pong (83)
Received ping (84) ... sending pong (85)
Received ping (86) ... sending pong (87)
Received ping (88) ... sending pong (89)
Received ping (90) ... sending pong (91)
Received ping (92) ... sending pong (93)
Received ping (94) ... sending pong (95)
Received ping (96) ... sending pong (97)
Received ping (98) ... sending pong (99)
Received ping (100) ... sending pong (101)

RPMsg demo ends
```

The Master peer also displays the result on the RAM console:

```
u-boot=> dcache flush; md c1fff000 260
c1fff000: 734d5052 69502067 502d676e 20676e6f  RPMsg Ping-Pong
c1fff010: 65657246 534f5452 6d654420 6d203a6f  FreeRTOS Demo: m
c1fff020: 65747361 72203a72 696e6e75 6120676e  aster: running a
c1fff030: 78302074 30303163 30303030 4e490a0d  t 0xc1000000..IN
c1fff040: 203a4f46 736d7072 6e695f67 20207469  FO: rpmsg_init
c1fff050: 20202020 20202020 203a2020 534d5052            : RPMS
c1fff060: 616d2047 72657473 696e6920 2e2e2074  G master init ..
c1fff070: 490d0a2e 3a4f464e 6d707220 6d5f6773  ...INFO: rpmsg_m
c1fff080: 65747361 6e695f72 20207469 3a202020  aster_init     :
c1fff090: 4d505220 6d204753 65747361 534e2072   RPMSG master NS
c1fff0a0: 72657320 65636976 61657220 0d0a7964   service ready..
c1fff0b0: 74696157 20676e69 20726f66 6120534e  Waiting for NS a
c1fff0c0: 756f6e6e 2065636e 0d2e2e2e 464e490a  nnounce .....INF
c1fff0d0: 72203a4f 67736d70 6d616e5f 72657365  O: rpmsg_nameser
c1fff0e0: 65636976 7361745f 63203a6b 74616572  vice_task: creat
c1fff0f0: 45206465 28205450 3a637273 202c3120  ed EPT (src: 1,
c1fff100: 3a747364 29303320 65530d0a 6e69646e  dst: 30)..Sendin
c1fff110: 69702067 2820676e 2e202930 72202e2e  g ping (0) ... r
c1fff120: 69656365 20646576 676e6f70 29312820  eceived pong (1)
c1fff130: 65530a0d 6e69646e 69702067 2820676e  ..Sending ping (
c1fff140: 2e202932 72202e2e 69656365 20646576  2) ... received
c1fff150: 676e6f70 29332820 65530a0d 6e69646e  pong (3)..Sendin
c1fff160: 69702067 2820676e 2e202934 72202e2e  g ping (4) ... r
c1fff170: 69656365 20646576 676e6f70 29352820  eceived pong (5)
c1fff180: 65530a0d 6e69646e 69702067 2820676e  ..Sending ping (
c1fff190: 2e202936 72202e2e 69656365 20646576  6) ... received
c1fff1a0: 676e6f70 29372820 65530a0d 6e69646e  pong (7)..Sendin
c1fff1b0: 69702067 2820676e 2e202938 72202e2e  g ping (8) ... r
c1fff1c0: 69656365 20646576 676e6f70 29392820  eceived pong (9)
c1fff1d0: 65530a0d 6e69646e 69702067 2820676e  ..Sending ping (
c1fff1e0: 20293031 202e2e2e 65636572 64657669  10) ... received
c1fff1f0: 6e6f7020 31282067 0a0d2931 646e6553   pong (11)..Send
c1fff200: 20676e69 676e6970 32312820 2e2e2029  ing ping (12) ..
c1fff210: 6572202e 76696563 70206465 20676e6f  . received pong
c1fff220: 29333128 65530a0d 6e69646e 69702067  (13)..Sending pi
c1fff230: 2820676e 20293431 202e2e2e 65636572  ng (14) ... rece
c1fff240: 64657669 6e6f7020 31282067 0a0d2935  ived pong (15)..
c1fff250: 646e6553 20676e69 676e6970 36312820  Sending ping (16
c1fff260: 2e2e2029 6572202e 76696563 70206465  ) ... received p
c1fff270: 20676e6f 29373128 65530a0d 6e69646e  ong (17)..Sendin
```

```
c1fff280: 69702067 2820676e 20293831 202e2e2e  g ping (18) ...
c1fff290: 65636572 64657669 6e6f7020 31282067  received pong (1
c1fff2a0: 0a0d2939 646e6553 20676e69 676e6970  9)..Sending ping
c1fff2b0: 30322820 2e2e2029 6572202e 76696563   (20) ... receiv
c1fff2c0: 70206465 20676e6f 29313228 65530a0d  ed pong (21)..Se
c1fff2d0: 6e69646e 69702067 2820676e 20293232  nding ping (22)
c1fff2e0: 202e2e2e 65636572 64657669 6e6f7020  ... received pon
c1fff2f0: 32282067 0a0d2933 646e6553 20676e69  g (23)..Sending
c1fff300: 676e6970 34322820 2e2e2029 6572202e  ping (24) ... re
c1fff310: 76696563 70206465 20676e6f 29353228  ceived pong (25)
c1fff320: 65530a0d 6e69646e 69702067 2820676e  ..Sending ping (
c1fff330: 20293632 202e2e2e 65636572 64657669  26) ... received
c1fff340: 6e6f7020 32282067 0a0d2937 646e6553   pong (27)..Send
c1fff350: 20676e69 676e6970 38322820 2e2e2029  ing ping (28) ..
c1fff360: 6572202e 76696563 70206465 20676e6f  . received pong
c1fff370: 29393228 65530a0d 6e69646e 69702067  (29)..Sending pi
c1fff380: 2820676e 20293033 202e2e2e 65636572  ng (30) ... rece
c1fff390: 64657669 6e6f7020 33282067 0a0d2931  ived pong (31)..
c1fff3a0: 646e6553 20676e69 676e6970 32332820  Sending ping (32
c1fff3b0: 2e2e2029 6572202e 76696563 70206465  ) ... received p
c1fff3c0: 20676e6f 29333328 65530a0d 6e69646e  ong (33)..Sendin
c1fff3d0: 69702067 2820676e 20293433 202e2e2e  g ping (34) ...
c1fff3e0: 65636572 64657669 6e6f7020 33282067  received pong (3
c1fff3f0: 0a0d2935 646e6553 20676e69 676e6970  5)..Sending ping
c1fff400: 36332820 2e2e2029 6572202e 76696563   (36) ... receiv
c1fff410: 70206465 20676e6f 29373328 65530a0d  ed pong (37)..Se
c1fff420: 6e69646e 69702067 2820676e 20293833  nding ping (38)
c1fff430: 202e2e2e 65636572 64657669 6e6f7020  ... received pon
c1fff440: 33282067 0a0d2939 646e6553 20676e69  g (39)..Sending
c1fff450: 676e6970 30342820 2e2e2029 6572202e  ping (40) ... re
c1fff460: 76696563 70206465 20676e6f 29313428  ceived pong (41)
c1fff470: 65530a0d 6e69646e 69702067 2820676e  ..Sending ping (
c1fff480: 20293234 202e2e2e 65636572 64657669  42) ... received
c1fff490: 6e6f7020 34282067 0a0d2933 646e6553   pong (43)..Send
c1fff4a0: 20676e69 676e6970 34342820 2e2e2029  ing ping (44) ..
c1fff4b0: 6572202e 76696563 70206465 20676e6f  . received pong
c1fff4c0: 29353428 65530a0d 6e69646e 69702067  (45)..Sending pi
c1fff4d0: 2820676e 20293634 202e2e2e 65636572  ng (46) ... rece
c1fff4e0: 64657669 6e6f7020 34282067 0a0d2937  ived pong (47)..
c1fff4f0: 646e6553 20676e69 676e6970 38342820  Sending ping (48
c1fff500: 2e2e2029 6572202e 76696563 70206465  ) ... received p
c1fff510: 20676e6f 29393428 65530a0d 6e69646e  ong (49)..Sendin
c1fff520: 69702067 2820676e 20293035 202e2e2e  g ping (50) ...
c1fff530: 65636572 64657669 6e6f7020 35282067  received pong (5
c1fff540: 0a0d2931 646e6553 20676e69 676e6970  1)..Sending ping
c1fff550: 32352820 2e2e2029 6572202e 76696563   (52) ... receiv
c1fff560: 70206465 20676e6f 29333528 65530a0d  ed pong (53)..Se
c1fff570: 6e69646e 69702067 2820676e 20293435  nding ping (54)
c1fff580: 202e2e2e 65636572 64657669 6e6f7020  ... received pon
c1fff590: 35282067 0a0d2935 646e6553 20676e69  g (55)..Sending
c1fff5a0: 676e6970 36352820 2e2e2029 6572202e  ping (56) ... re
c1fff5b0: 76696563 70206465 20676e6f 29373528  ceived pong (57)
c1fff5c0: 65530a0d 6e69646e 69702067 2820676e  ..Sending ping (
c1fff5d0: 20293835 202e2e2e 65636572 64657669  58) ... received
c1fff5e0: 6e6f7020 35282067 0a0d2939 646e6553   pong (59)..Send
c1fff5f0: 20676e69 676e6970 30362820 2e2e2029  ing ping (60) ..
c1fff600: 6572202e 76696563 70206465 20676e6f  . received pong
c1fff610: 29313628 65530a0d 6e69646e 69702067  (61)..Sending pi
c1fff620: 2820676e 20293236 202e2e2e 65636572  ng (62) ... rece
c1fff630: 64657669 6e6f7020 36282067 0a0d2933  ived pong (63)..
```

```
c1fff640: 646e6553 20676e69 676e6970 34362820  Sending ping (64
c1fff650: 2e2e2029 6572202e 76696563 70206465  ) ... received p
c1fff660: 20676e6f 29353628 65530a0d 6e69646e  ong (65)..Sendin
c1fff670: 69702067 2820676e 20293636 202e2e2e  g ping (66) ...
c1fff680: 65636572 64657669 6e6f7020 36282067  received pong (6
c1fff690: 0a0d2937 646e6553 20676e69 676e6970  7)..Sending ping
c1fff6a0: 38362820 2e2e2029 6572202e 76696563   (68) ... receiv
c1fff6b0: 70206465 20676e6f 29393628 65530a0d  ed pong (69)..Se
c1fff6c0: 6e69646e 69702067 2820676e 20293037  nding ping (70)
c1fff6d0: 202e2e2e 65636572 64657669 6e6f7020  ... received pon
c1fff6e0: 37282067 0a0d2931 646e6553 20676e69  g (71)..Sending
c1fff6f0: 676e6970 32372820 2e2e2029 6572202e  ping (72) ... re
c1fff700: 76696563 70206465 20676e6f 29333728  ceived pong (73)
c1fff710: 65530a0d 6e69646e 69702067 2820676e  ..Sending ping (
c1fff720: 20293437 202e2e2e 65636572 64657669  74) ... received
c1fff730: 6e6f7020 37282067 0a0d2935 646e6553   pong (75)..Send
c1fff740: 20676e69 676e6970 36372820 2e2e2029  ing ping (76) ..
c1fff750: 6572202e 76696563 70206465 20676e6f  . received pong
c1fff760: 29373728 65530a0d 6e69646e 69702067  (77)..Sending pi
c1fff770: 2820676e 20293837 202e2e2e 65636572  ng (78) ... rece
c1fff780: 64657669 6e6f7020 37282067 0a0d2939  ived pong (79)..
c1fff790: 646e6553 20676e69 676e6970 30382820  Sending ping (80
c1fff7a0: 2e2e2029 6572202e 76696563 70206465  ) ... received p
c1fff7b0: 20676e6f 29313828 65530a0d 6e69646e  ong (81)..Sendin
c1fff7c0: 69702067 2820676e 20293238 202e2e2e  g ping (82) ...
c1fff7d0: 65636572 64657669 6e6f7020 38282067  received pong (8
c1fff7e0: 0a0d2933 646e6553 20676e69 676e6970  3)..Sending ping
c1fff7f0: 34382820 2e2e2029 6572202e 76696563   (84) ... receiv
c1fff800: 70206465 20676e6f 29353828 65530a0d  ed pong (85)..Se
c1fff810: 6e69646e 69702067 2820676e 20293638  nding ping (86)
c1fff820: 202e2e2e 65636572 64657669 6e6f7020  ... received pon
c1fff830: 38282067 0a0d2937 646e6553 20676e69  g (87)..Sending
c1fff840: 676e6970 38382820 2e2e2029 6572202e  ping (88) ... re
c1fff850: 76696563 70206465 20676e6f 29393828  ceived pong (89)
c1fff860: 65530a0d 6e69646e 69702067 2820676e  ..Sending ping (
c1fff870: 20293039 202e2e2e 65636572 64657669  90) ... received
c1fff880: 6e6f7020 39282067 0a0d2931 646e6553   pong (91)..Send
c1fff890: 20676e69 676e6970 32392820 2e2e2029  ing ping (92) ..
c1fff8a0: 6572202e 76696563 70206465 20676e6f  . received pong
c1fff8b0: 29333928 65530a0d 6e69646e 69702067  (93)..Sending pi
c1fff8c0: 2820676e 20293439 202e2e2e 65636572  ng (94) ... rece
c1fff8d0: 64657669 6e6f7020 39282067 0a0d2935  ived pong (95)..
c1fff8e0: 646e6553 20676e69 676e6970 36392820  Sending ping (96
c1fff8f0: 2e2e2029 6572202e 76696563 70206465  ) ... received p
c1fff900: 20676e6f 29373928 65530a0d 6e69646e  ong (97)..Sendin
c1fff910: 69702067 2820676e 20293839 202e2e2e  g ping (98) ...
c1fff920: 65636572 64657669 6e6f7020 39282067  received pong (9
c1fff930: 0a0d2939 646e6553 20676e69 676e6970  9)..Sending ping
c1fff940: 30312820 2e202930 72202e2e 69656365   (100) ... recei
c1fff950: 20646576 676e6f70 30312820 0a0d2931  ved pong (101)..
c1fff960: 50520a0d 2067734d 6f6d6564 646e6520  ..RPMsg demo end
c1fff970: 000a0d73 00000000 00000000 00000000  s...............
```

– Follow the steps below to run the demo on i.MX 93 11x11 EVK and i.MX 93 14x14 EVK:

1. First, boot the remote peer FreeRTOS on core1 with UART2:

```
u-boot=> load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/rpmsg-
pingpong-freertos/rpmsg_pingpong_remote_ca55_RTOS0_UART2.bin
u-boot=> dcache flush; icache flush;
u-boot=> cpu 1 release 0xd0000000
```

The Remote peer boots up and waits for RPMSG link up, the logs on UART2 console are dsiplayed as the following:

```
RPMsg Ping-Pong FreeRTOS Demo: remote: running at 0xd0000000
master RAM_CONSOLE at 0xd1fff000
INFO: rpmsg_init              : RPMSG remote init ...
INFO: rpmsg_remote_init       : waiting for link establish ...
```

2. As a second step, boot the Master peer FreeRTOS on core0 with UART1.

```
u-boot=> load mmc 1:2 0xd1000000 /examples/heterogeneous-multicore/rpmsg-
pingpong-freertos/rpmsg_pingpong_master_RTOS1_ca55_UART1.bin
u-boot=> dcache flush; icache flush;
u-boot=> go 0xd1000000
```

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

```
RPMsg Ping-Pong FreeRTOS Demo: remote: running at 0xd0000000
master RAM_CONSOLE at 0xd1fff000
INFO: rpmsg_init              : RPMSG remote init ...
INFO: rpmsg_remote_init       : waiting for link establish ...
INFO: rpmsg_remote_init       : RPMSG link up
Waiting for ping ...
Received ping (0) ... sending pong (1)
Received ping (2) ... sending pong (3)
Received ping (4) ... sending pong (5)
Received ping (6) ... sending pong (7)
Received ping (8) ... sending pong (9)
Received ping (10) ... sending pong (11)
Received ping (12) ... sending pong (13)
Received ping (14) ... sending pong (15)
Received ping (16) ... sending pong (17)
Received ping (18) ... sending pong (19)
Received ping (20) ... sending pong (21)
Received ping (22) ... sending pong (23)
Received ping (24) ... sending pong (25)
Received ping (26) ... sending pong (27)
Received ping (28) ... sending pong (29)
Received ping (30) ... sending pong (31)
Received ping (32) ... sending pong (33)
Received ping (34) ... sending pong (35)
Received ping (36) ... sending pong (37)
Received ping (38) ... sending pong (39)
Received ping (40) ... sending pong (41)
Received ping (42) ... sending pong (43)
Received ping (44) ... sending pong (45)
Received ping (46) ... sending pong (47)
Received ping (48) ... sending pong (49)
Received ping (50) ... sending pong (51)
Received ping (52) ... sending pong (53)
Received ping (54) ... sending pong (55)
Received ping (56) ... sending pong (57)
Received ping (58) ... sending pong (59)
Received ping (60) ... sending pong (61)
Received ping (62) ... sending pong (63)
Received ping (64) ... sending pong (65)
Received ping (66) ... sending pong (67)
Received ping (68) ... sending pong (69)
Received ping (70) ... sending pong (71)
Received ping (72) ... sending pong (73)
Received ping (74) ... sending pong (75)
Received ping (76) ... sending pong (77)
Received ping (78) ... sending pong (79)
Received ping (80) ... sending pong (81)
Received ping (82) ... sending pong (83)
Received ping (84) ... sending pong (85)
Received ping (86) ... sending pong (87)
Received ping (88) ... sending pong (89)
Received ping (90) ... sending pong (91)
Received ping (92) ... sending pong (93)
Received ping (94) ... sending pong (95)
Received ping (96) ... sending pong (97)
Received ping (98) ... sending pong (99)
Received ping (100) ... sending pong (101)

RPMsg demo ends
```

The Master peer also displays the result on the UART1:

```
## Starting application at 0xD1000000 ...
RPMsg Ping-Pong FreeRTOS Demo: master: running at 0xd1000000
INFO: rpmsg_init           : RPMSG master init ...
INFO: rpmsg_master_init    : RPMSG master NS service ready
Waiting for NS announce ...
INFO: rpmsg_nameservice_task: created EPT (src: 1, dst: 30)
Sending ping (0) ... received pong (1)
Sending ping (2) ... received pong (3)
Sending ping (4) ... received pong (5)
Sending ping (6) ... received pong (7)
Sending ping (8) ... received pong (9)
Sending ping (10) ... received pong (11)
Sending ping (12) ... received pong (13)
Sending ping (14) ... received pong (15)
Sending ping (16) ... received pong (17)
Sending ping (18) ... received pong (19)
Sending ping (20) ... received pong (21)
Sending ping (22) ... received pong (23)
Sending ping (24) ... received pong (25)
Sending ping (26) ... received pong (27)
Sending ping (28) ... received pong (29)
Sending ping (30) ... received pong (31)
Sending ping (32) ... received pong (33)
Sending ping (34) ... received pong (35)
Sending ping (36) ... received pong (37)
Sending ping (38) ... received pong (39)
Sending ping (40) ... received pong (41)
Sending ping (42) ... received pong (43)
Sending ping (44) ... received pong (45)
Sending ping (46) ... received pong (47)
Sending ping (48) ... received pong (49)
Sending ping (50) ... received pong (51)
Sending ping (52) ... received pong (53)
Sending ping (54) ... received pong (55)
Sending ping (56) ... received pong (57)
Sending ping (58) ... received pong (59)
Sending ping (60) ... received pong (61)
Sending ping (62) ... received pong (63)
Sending ping (64) ... received pong (65)
Sending ping (66) ... received pong (67)
Sending ping (68) ... received pong (69)
Sending ping (70) ... received pong (71)
Sending ping (72) ... received pong (73)
Sending ping (74) ... received pong (75)
Sending ping (76) ... received pong (77)
Sending ping (78) ... received pong (79)
Sending ping (80) ... received pong (81)
Sending ping (82) ... received pong (83)
Sending ping (84) ... received pong (85)
Sending ping (86) ... received pong (87)
Sending ping (88) ... received pong (89)
Sending ping (90) ... received pong (91)
Sending ping (92) ... received pong (93)
Sending ping (94) ... received pong (95)
Sending ping (96) ... received pong (97)
Sending ping (98) ... received pong (99)
Sending ping (100) ... received pong (101)

RPMsg demo ends
```

– Follow the steps below to run the demo on i.MX 95 15x15 EVK and i.MX 95 19x19 EVK:

1. First, boot the remote peer FreeRTOS on core4 with UART3:

```
u-boot=> load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/rpmsg-
pingpong-freertos/rpmsg_pingpong_remote_ca55_RTOS0_UART3.bin
u-boot=> dcache flush; icache flush;
u-boot=> cpu 4 release 0xd0000000
```

The Remote peer boots up and waits for RPMSG link up, the logs on UART3 console are dsiplayed as the following:

```
RPMsg Ping-Pong FreeRTOS Demo: remote: running at 0xd0000000
master RAM_CONSOLE at 0xd1fff000
INFO: rpmsg_init            : RPMSG remote init ...
INFO: rpmsg_remote_init     : waiting for link establish ...
```

2. As a second step, boot the Master peer FreeRTOS on core5 with UART1.

```
u-boot=> load mmc 1:2 0xd1000000 /examples/heterogeneous-multicore/rpmsg-
pingpong-freertos/rpmsg_pingpong_master_ca55_RTOS1_UART1.bin
u-boot=> dcache flush; icache flush;
u-boot=> cpu 5 release 0xd0000000
```

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

```
RPMsg Ping-Pong FreeRTOS Demo: remote: running at 0xd0000000
master RAM_CONSOLE at 0xd1fff000
INFO: rpmsg_init            : RPMSG remote init ...
INFO: rpmsg_remote_init     : waiting for link establish ...
INFO: rpmsg_remote_init     : RPMSG link up
Waiting for ping ...
Received ping (0) ... sending pong (1)
Received ping (2) ... sending pong (3)
Received ping (4) ... sending pong (5)
Received ping (6) ... sending pong (7)
Received ping (8) ... sending pong (9)
Received ping (10) ... sending pong (11)
Received ping (12) ... sending pong (13)
Received ping (14) ... sending pong (15)
Received ping (16) ... sending pong (17)
Received ping (18) ... sending pong (19)
Received ping (20) ... sending pong (21)
Received ping (22) ... sending pong (23)
Received ping (24) ... sending pong (25)
Received ping (26) ... sending pong (27)
Received ping (28) ... sending pong (29)
Received ping (30) ... sending pong (31)
Received ping (32) ... sending pong (33)
Received ping (34) ... sending pong (35)
Received ping (36) ... sending pong (37)
Received ping (38) ... sending pong (39)
Received ping (40) ... sending pong (41)
Received ping (42) ... sending pong (43)
Received ping (44) ... sending pong (45)
Received ping (46) ... sending pong (47)
Received ping (48) ... sending pong (49)
Received ping (50) ... sending pong (51)
Received ping (52) ... sending pong (53)
Received ping (54) ... sending pong (55)
Received ping (56) ... sending pong (57)
Received ping (58) ... sending pong (59)
Received ping (60) ... sending pong (61)
Received ping (62) ... sending pong (63)
Received ping (64) ... sending pong (65)
Received ping (66) ... sending pong (67)
Received ping (68) ... sending pong (69)
Received ping (70) ... sending pong (71)
Received ping (72) ... sending pong (73)
Received ping (74) ... sending pong (75)
Received ping (76) ... sending pong (77)
Received ping (78) ... sending pong (79)
Received ping (80) ... sending pong (81)
Received ping (82) ... sending pong (83)
Received ping (84) ... sending pong (85)
Received ping (86) ... sending pong (87)
Received ping (88) ... sending pong (89)
Received ping (90) ... sending pong (91)
Received ping (92) ... sending pong (93)
Received ping (94) ... sending pong (95)
Received ping (96) ... sending pong (97)
Received ping (98) ... sending pong (99)
Received ping (100) ... sending pong (101)

RPMsg demo ends
```

The Master peer also displays the result on the UART1:

```
## Starting application at 0xD1000000 ...
RPMsg Ping-Pong FreeRTOS Demo: master: running at 0xd1000000
INFO: rpmsg_init            : RPMSG master init ...
INFO: rpmsg_master_init     : RPMSG master NS service ready
Waiting for NS announce ...
INFO: rpmsg_nameservice_task: created EPT (src: 1, dst: 30)
Sending ping (0) ... received pong (1)
Sending ping (2) ... received pong (3)
Sending ping (4) ... received pong (5)
Sending ping (6) ... received pong (7)
Sending ping (8) ... received pong (9)
Sending ping (10) ... received pong (11)
Sending ping (12) ... received pong (13)
Sending ping (14) ... received pong (15)
Sending ping (16) ... received pong (17)
Sending ping (18) ... received pong (19)
Sending ping (20) ... received pong (21)
Sending ping (22) ... received pong (23)
Sending ping (24) ... received pong (25)
Sending ping (26) ... received pong (27)
Sending ping (28) ... received pong (29)
Sending ping (30) ... received pong (31)
Sending ping (32) ... received pong (33)
Sending ping (34) ... received pong (35)
Sending ping (36) ... received pong (37)
Sending ping (38) ... received pong (39)
Sending ping (40) ... received pong (41)
Sending ping (42) ... received pong (43)
Sending ping (44) ... received pong (45)
Sending ping (46) ... received pong (47)
Sending ping (48) ... received pong (49)
Sending ping (50) ... received pong (51)
Sending ping (52) ... received pong (53)
Sending ping (54) ... received pong (55)
Sending ping (56) ... received pong (57)
Sending ping (58) ... received pong (59)
Sending ping (60) ... received pong (61)
Sending ping (62) ... received pong (63)
Sending ping (64) ... received pong (65)
Sending ping (66) ... received pong (67)
Sending ping (68) ... received pong (69)
Sending ping (70) ... received pong (71)
Sending ping (72) ... received pong (73)
Sending ping (74) ... received pong (75)
Sending ping (76) ... received pong (77)
Sending ping (78) ... received pong (79)
Sending ping (80) ... received pong (81)
Sending ping (82) ... received pong (83)
Sending ping (84) ... received pong (85)
Sending ping (86) ... received pong (87)
Sending ping (88) ... received pong (89)
Sending ping (90) ... received pong (91)
Sending ping (92) ... received pong (93)
Sending ping (94) ... received pong (95)
Sending ping (96) ... received pong (97)
Sending ping (98) ... received pong (99)
Sending ping (100) ... received pong (101)

RPMsg demo ends
```

- Or using Linux remoteproc to start both the RPMSG master and remote RTOS instances
  - Follow the steps below to run the demo on i.MX 8M Plus LPDDR4 EVK:
    1. Start up Linux:

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

    2. After Linux boot up, start remote peer FreeRTOS on core2, it use UART4 as debug console:

       ```
       root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-
       pingpong-freertos/rpmsg_pingpong_remote_ca53_RTOS0_UART4.elf > /sys/
       devices/platform/remoteproc-ca53-2/remoteproc/remoteproc1/firmware
       root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
       ca53-2/remoteproc/remoteproc1/state
       ```

3. Then start the Master peer FreeRTOS on core3, which uses the RAM console as debug console:

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-
pingpong-freertos/
rpmsg_pingpong_master_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.elf >  /sys/
devices/platform/remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

4. And you can use the tool to check RAM Console log:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xc1fff000 -r 1
RAM Console@0xc1fff000:
RPMsg Ping-Pong FreeRTOS Demo: master: running at 0xc1000000
INFO: rpmsg_init            : RPMSG master init ...
INFO: rpmsg_master_init     : RPMSG master NS service ready
Waiting for NS announce ...
INFO: rpmsg_nameservice_task: created EPT (src: 1, dst: 30)
Sending ping (0) ... received pong (1)
Sending ping (2) ... received pong (3)
Sending ping (4) ... received pong (5)
Sending ping (6) ... received pong (7)
Sending ping (8) ... received pong (9)
Sending ping (10) ... received pong (11)
Sending ping (12) ... received pong (13)
Sending ping (14) ... received pong (15)
Sending ping (16) ... received pong (17)
Sending ping (18) ... received pong (19)
Sending ping (20) ... received pong (21)
Sending ping (22) ... received pong (23)
Sending ping (24) ... received pong (25)
Sending ping (26) ... received pong (27)
Sending ping (28) ... received pong (29)
Sending ping (30) ... received pong (31)
Sending ping (32) ... received pong (33)
Sending ping (34) ... received pong (35)
Sending ping (36) ... received pong (37)
Sending ping (38) ... received pong (39)
Sending ping (40) ... received pong (41)
Sending ping (42) ... received pong (43)
Sending ping (44) ... received pong (45)
Sending ping (46) ... received pong (47)
Sending ping (48) ... received pong (49)
Sending ping (50) ... received pong (51)
Sending ping (52) ... received pong (53)
Sending ping (54) ... received pong (55)
Sending ping (56) ... received pong (57)
Sending ping (58) ... received pong (59)
Sending ping (60) ... received pong (61)
Sending ping (62) ... received pong (63)
Sending ping (64) ... received pong (65)
Sending ping (66) ... received pong (67)
Sending ping (68) ... received pong (69)
Sending ping (70) ... received pong (71)
Sending ping (72) ... received pong (73)
Sending ping (74) ... received pong (75)
Sending ping (76) ... received pong (77)
Sending ping (78) ... received pong (79)
Sending ping (80) ... received pong (81)
Sending ping (82) ... received pong (83)
Sending ping (84) ... received pong (85)
Sending ping (86) ... received pong (87)
Sending ping (88) ... received pong (89)
Sending ping (90) ... received pong (91)
Sending ping (92) ... received pong (93)
Sending ping (94) ... received pong (95)
Sending ping (96) ... received pong (97)
Sending ping (98) ... received pong (99)
Sending ping (100) ... received pong (101)

RPMsg demo ends
```

– Follow the steps below to run the demo on i.MX 95 15x15 EVK and i.MX 95 19x19 EVK:

1. Start up Linux:

```
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rtos.dtb; # for i.MX 95
 15x15 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rtos.dtb; # for i.MX 95
 19x19 EVK
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run bsp_bootcmd
```

2. After Linux boot up, start remote peer FreeRTOS on core4, it use UART3 as debug console:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
rpmsg-pingpong-freertos/rpmsg_pingpong_remote_ca55_RTOS0_UART3.elf > /sys/
devices/platform/remoteproc-ca55-4/remoteproc/remoteproc3/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/
remoteproc-ca55-4/remoteproc/remoteproc3/state
```

3. Then start the Master peer FreeRTOS on core5, which uses the UART1 as debug console:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
rpmsg-pingpong-freertos/rpmsg_pingpong_master_ca55_RTOS1_UART1.elf > /sys/
devices/platform/remoteproc-ca55-5/remoteproc/remoteproc4/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/
remoteproc-ca55-5/remoteproc/remoteproc4/state
```

3.4.5 Complex RPMSG on MPU

3.4.5.1 Overview

To demonstrate a typical RPMSG use case on the MPU platform, a complex RPMsg str-echo application is available in the repository: heterogeneous-multicore.

Figure 32 shows the application setup:

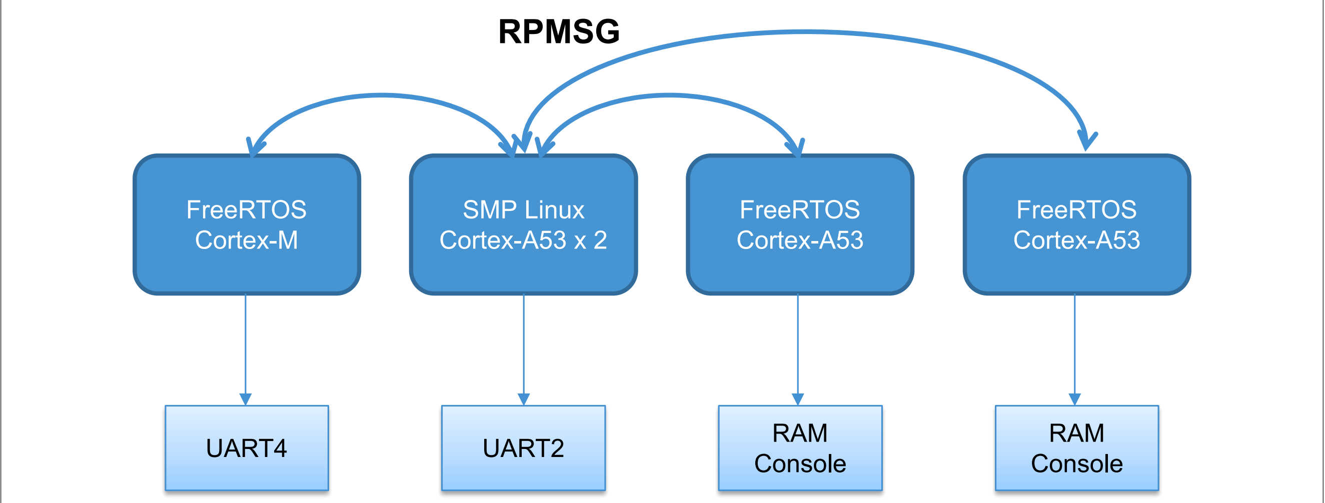


Figure 32. Complex RPMsg set up on MPU

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

3.4.5.2 Building the Complex str_echo application

RPMSG complex str_echo application runs on FreeRTOS, it is an application named ""rpmsg_str_echo" in the repo: heterogeneous-multicore.

Refer to "Building Heterogeneous Multicore RTOS Application" for how to build the application.

3.4.5.3 Running the Complex str_echo application

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

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

The following RTOS images are provided to run the application:

```
/examples/heterogeneous-multicore/rpmsg-str-echo-freertos/
├── rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.bin
├── rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.elf
├── rpmsg_str_echo_ca53_RTOS0_UART4.bin
├── rpmsg_str_echo_ca53_RTOS0_UART4.elf
├── rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.bin
├── rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.elf
├── rpmsg_str_echo_cm7.bin
└── rpmsg_str_echo_cm7.elf
```

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

Can use U-Boot commands under U-Boot or remoteproc under Linux to start FreeRTOS, choose one the method:

- Using U-Boot Commands to start RTOS, then start up Linux
  1. Boot the First Cortex-A Core RTOS on A53 Core2:

```
u-boot=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.bin
u-boot=> dcache flush; icache flush; cpu 2 release 0xC0000000
```

  2. Boot the Second Cortex-A Core RTOS on A53 Core3:

```
u-boot=> ext4load mmc 1:2 0xC1000000 /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.bin
u-boot=> dcache flush; icache flush; cpu 3 release 0xC1000000
```

  3. Boot M7 Core RTOS:

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

     The below log is displayed for the UART4 console:

```
Cortex-M7: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
INFO: rpmsg_init             : RPMSG init ...
INFO: rpmsg_init             : waiting for link establish ...
```

  4. Boot up Linux using the commands:

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

  5. When Linux is up, install tty driver module

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

6. Then, check the RAM console log of Cortex-A RTOS0 as shown below.

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xC0FFF000 -r 1
RAM Console@0xc0fff000:

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

7. Check the RAM console log of Cortex-A RTOS1:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xC1FFF000 -r 1
RAM Console@0xc1fff000:

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

The following log is displayed for the UART4 of Cortex-M Core 4:

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

It creates the following RPMsg devices:

```
root@imx8mp-lpddr4-evk:~# ls /dev/ttyRPMSG*
/dev/ttyRPMSG0  /dev/ttyRPMSG1  /dev/ttyRPMSG2  /dev/ttyRPMSG3  /dev/
ttyRPMSG4  /dev/ttyRPMSG5  /dev/ttyRPMSG6  /dev/ttyRPMSG7  /dev/ttyRPMSG8
```

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

- Or start up Linux, then using remoteproc to start RTOS
  1. Start up Linux

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

  2. Start the first A53 RTOS

After Linux boot up, start the first A53 Core RTOS on A53 Core2:

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0xc0fff000.elf >  /sys/
devices/platform/remoteproc-ca53-2/remoteproc/remoteproc1/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-2/remoteproc/remoteproc1/state
```

The RAM Console log for the first A53 RTOS can be checked using the below command:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xc0fff000 -r 1
```

3. Start the second A53 RTOS
   Then start the second A53 Core RTOS on A53 Core3:

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0xc1fff000.elf >  /sys/
devices/platform/remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mp-lpddr4-evk:~# echo start >  /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

   The RAM Console log for the second A53 RTOS can be checked with command:

```
root@imx8mp-lpddr4-evk:~# ram_console_dump -a 0xc1fff000 -r 1
```

4. Start M7 Core RTOS

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_cm7.elf > /sys//devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys//devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/state
```

5. Install tty driver module:

```
root@imx8mp-lpddr4-evk:~# modprobe imx_rpmsg_tty
```

   It creates the following RPMsg devices:

```
root@imx8mp-lpddr4-evk:~# ls /dev/ttyRPMSG*
/dev/ttyRPMSG0  /dev/ttyRPMSG1  /dev/ttyRPMSG2  /dev/ttyRPMSG3  /dev/
ttyRPMSG4  /dev/ttyRPMSG5  /dev/ttyRPMSG6  /dev/ttyRPMSG7  /dev/ttyRPMSG8
```

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

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

```
root@imx8mp-lpddr4-evk:~# for i in {0..8}; do stty -F /dev/ttyRPMSG$i -echo;
 done
root@imx8mp-lpddr4-evk:~# echo "adfad" > /dev/ttyRPMSG2
```

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

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

  Or use a minicom to open one RPMsg endpoint. It echoes back the character inputted in the minicom console:

```
root@imx8mp-lpddr4-evk:~# minicom -D /dev/ttyRPMSG6
```

Then, input some characters typed in using the minicom console. This character is sent to the RTOS endpoint from Linux. Then the application running on RTOS sends the characters back to Linux. Finally all characters are echoed back in the minicom console. For example, if you input the characters "`hello world!`", the below log would be displayed:

```
Welcome to minicom 2.8

OPTIONS: I18n
Compiled on Jan  1 2021, 17:45:55.
Port /dev/ttyRPMSG6, 08:36:41

Press CTRL-A Z for help on special keys

hello world!
```

Known Issues:

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

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

The following RTOS images are provided to run the application:

```
/examples/heterogeneous-multicore/rpmsg-str-echo-freertos/
├── rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0x94bff000.bin
├── rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0x94bff000.elf
├── rpmsg_str_echo_ca53_RTOS0_UART4.bin
├── rpmsg_str_echo_ca53_RTOS0_UART4.elf
├── rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0x95bff000.bin
├── rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0x95bff000.elf
├── rpmsg_str_echo_cm4.bin
└── rpmsg_str_echo_cm4.elf
```

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

Can use U-Boot commands under U-Boot or remoteproc under Linux to start FreeRTOS, choose one the method:

- Using U-Boot Commands to start RTOS, then start up Linux
  1. Boot the First Cortex-A Core RTOS

     ```
     u-boot=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/
     rpmsg-str-echo-freertos/rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0x94bff000.bin
     u-boot=> dcache flush; icache flush; cpu 2 release 0x93C00000
     ```
  2. Boot the Second Cortex-A Core RTOS

     ```
     u-boot=> ext4load mmc 1:2 0x94C00000 /examples/heterogeneous-multicore/
     rpmsg-str-echo-freertos/rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0x95bff000.bin
     u-boot=> dcache flush; icache flush; cpu 3 release 0x94C00000
     ```
  3. Boot Cortex-M Core RTOS

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

The log for UART4 Console is displayed below:

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

4. Boot Linux up

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

5. When Linux is up, install tty driver module

```
root@imx8mm-lpddr4-evk:~# modprobe imx_rpmsg_tty
[   21.770356] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.3: new
 channel: 0x400 -> 0x3!
[   21.770576] Install rpmsg tty driver!
[   21.773539] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.4: new
 channel: 0x401 -> 0x4!
[   21.773804] Install rpmsg tty driver!
[   21.774034] imx_rpmsg_tty virtio0.rpmsg-virtual-tty-channel-1.-1.5: new
 channel: 0x402 -> 0x5!
[   21.774156] Install rpmsg tty driver!
[   21.774275] imx_rpmsg_tty virtio1.rpmsg-virtual-tty-channel-1.-1.6: new
 channel: 0x400 -> 0x6!
[   21.774360] Install rpmsg tty driver!
[   21.774443] imx_rpmsg_tty virtio1.rpmsg-virtual-tty-channel-1.-1.7: new
 channel: 0x401 -> 0x7!
[   21.774530] Install rpmsg tty driver!
[   21.774586] imx_rpmsg_tty virtio1.rpmsg-virtual-tty-channel-1.-1.8: new
 channel: 0x402 -> 0x8!
[   21.774663] Install rpmsg tty driver!
[   21.774726] imx_rpmsg_tty virtio2.rpmsg-virtual-tty-channel-1.-1.0: new
 channel: 0x400 -> 0x0!
[   21.774810] Install rpmsg tty driver!
[   21.774880] imx_rpmsg_tty virtio2.rpmsg-virtual-tty-channel-1.-1.1: new
 channel: 0x401 -> 0x1!
[   21.774960] Install rpmsg tty driver!
[   21.775022] imx_rpmsg_tty virtio2.rpmsg-virtual-tty-channel-1.-1.2: new
 channel: 0x402 -> 0x2!
[   21.775111] Install rpmsg tty driver!
```

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

```
root@imx8mm-lpddr4-evk:~# ram_console_dump 0x94BFF000
RAM Console@0x94bff000:

Cortex-A53: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
RTOS1: RAM console@0x95Bff000
INFO: rpmsg_init              : RPMSG remote init ...
INFO: rpmsg_remote_init       : waiting for link establish ...
INFO: rpmsg_remote_init       : RPMSG link up
ept3: Get Message From Master Side : "hello world!" [len : 12]
ept4: Get Message From Master Side : "hello world!" [len : 12]
ept5: Get Message From Master Side : "hello world!" [len : 12]
 > ...........................................................
```

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

```
root@imx8mm-lpddr4-evk:~# ram_console_dump -a 0x95BFF000 -r 1
RAM Console@0x95bff000:
```

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

The following is the log displayed for Cortex-M Core's UART4 console:

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

The following RPMsg devices are created:

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

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

- Or start up Linux, then using remoteproc to start RTOS
  1. Start up Linux

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

  2. Start the first A53 RTOS
     After Linux boot up, start the first A53 Core RTOS on A53 Core2:

     ```
     root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-str-
     echo-freertos/rpmsg_str_echo_ca53_RTOS0_RAM_CONSOLE-0x94bff000.elf >  /sys/
     devices/platform/remoteproc-ca53-2/remoteproc/remoteproc1/firmware
     root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
     ca53-2/remoteproc/remoteproc1/state
     ```

     The RAM Console log for the first A53 RTOS can be checked with command:

     ```
     root@imx8mm-lpddr4-evk:~# ram_console_dump -a 0x94bff000 -r 1
     ```

  3. Start the second A53 RTOS
     Then start the second A53 Core RTOS on A53 Core3:

     ```
     root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-str-
     echo-freertos/rpmsg_str_echo_ca53_RTOS1_RAM_CONSOLE-0x95bff000.elf >  /sys/
     devices/platform/remoteproc-ca53-3/remoteproc/remoteproc2/firmware
     root@imx8mm-lpddr4-evk:~# echo start >  /sys/devices/platform/remoteproc-
     ca53-3/remoteproc/remoteproc2/state
     ```

     The RAM Console log for the second A53 RTOS can be checked with command:

     ```
     root@imx8mm-lpddr4-evk:~# ram_console_dump -a 0x95bff000 -r 1
     ```

4. Start M4 Core RTOS

```
root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/rpmsg-str-
echo-freertos/rpmsg_str_echo_cm4.elf > /sys//devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys//devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/state
```

5. Install tty driver module:

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

It creates the following RPMsg devices:

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

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

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

```
root@imx8mm-lpddr4-evk:~# for i in {0..8}; do stty -F /dev/ttyRPMSG$i -echo;
 done
root@imx8mm-lpddr4-evk:~# echo "adfad" > /dev/ttyRPMSG2
```

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

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

Or use a minicom to open one RPMsg endpoint. It echoes back the character inputted in the minicom console:

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

Then input some characters typed in using the minicom console. This character is sent to the RTOS endpoint from Linux. Then the application running on RTOS sends the characters back to Linux. Finally all characters are echoed back in the minicom console. For example, if you input the characters "hello world!", the below log would be displayed:

```
Welcome to minicom 2.8

OPTIONS: I18n
Compiled on Jan  1 2021, 17:45:55.
Port /dev/ttyRPMSG6, 08:36:41

Press CTRL-A Z for help on special keys

hello world!
```

Known Issues:

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

3.4.5.3.3 Running the application on i.MX 95 15x15 and 19x19 EVK

The following RTOS images are provided to run the application:

```
/examples/heterogeneous-multicore/rpmsg-str-echo-freertos/
├── rpmsg_str_echo_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.bin
├── rpmsg_str_echo_ca55_RTOS0_RAM_CONSOLE-0xd0fff000.elf
├── rpmsg_str_echo_ca55_RTOS0_UART3.bin
├── rpmsg_str_echo_ca55_RTOS0_UART3.elf
├── rpmsg_str_echo_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin
├── rpmsg_str_echo_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.elf
├── rpmsg_str_echo_ca55_RTOS1_UART1.bin
└── rpmsg_str_echo_ca55_RTOS1_UART1.elf
```

Use U-Boot commands or Linux remoteproc to start the RTOS instances:

- Using U-Boot commands to start RTOS, then start up Linux
  1. Boot the first Cortex-A Core RTOS

     ```
     u-boot=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/
     rpmsg-str-echo-freertos/rpmsg_str_echo_ca55_RTOS0_UART3.bin
     u-boot=> dcache flush; icache flush; cpu 4 release 0xd0000000
     ```

  2. Boot the second Cortex-A Core RTOS

     ```
     u-boot=> ext4load mmc 1:2 0xd1000000 /examples/heterogeneous-multicore/
     rpmsg-str-echo-freertos/rpmsg_str_echo_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.bin
     u-boot=> dcache flush; icache flush; cpu 5 release 0xd1000000
     ```

  3. Boot Linux up

     ```
     # Running on i.MX 95 15x15 EVK
     u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rpmsg.dtb
     # Running on i.MX 95 19x19 EVK
     u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rpmsg.dtb
     u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
     u-boot=> run bsp_bootcmd
     ```

  4. When Linux is up, install tty driver module

     ```
     root@imx95-19x19-lpddr5-evk:~# modprobe imx_rpmsg_tty
     ```

  5. Then check Cortex-A RTOS0's UART3 console:

     ```
     Cortex-A55: RTOS0: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
     RTOS1: RAM console@0xd1fff000
     INFO: rpmsg_init            : RPMSG remote init ...
     INFO: rpmsg_remote_init     : waiting for link establish ...
     INFO: rpmsg_remote_init     : RPMSG link up
     INFO: rpmsg_create_ept      : Sending Nameservice string: rpmsg-virtual-tty-
     channel-1
     INFO: rpmsg_create_ept      : Sending Nameservice string: rpmsg-virtual-tty-
     channel-1
     INFO: rpmsg_create_ept      : Sending Nameservice string: rpmsg-virtual-tty-
     channel-1
     ept3: Get Message From Master Side : "hello world!" [len : 12]
     ept4: Get Message From Master Side : "hello world!" [len : 12]
     ept5: Get Message From Master Side : "hello world!" [len : 12]
     ```

  6. And check the Cortex-A RTOS1's RAM console logs using the `ram_console_dump` tool under Linux:

     ```
     root@imx95-19x19-lpddr5-evk:~# ram_console_dump -a 0xd1fff000 -r 1
     ```

```
RAM Console@0xd1fff000:

Cortex-A55: RTOS1: Multiple Endpoints RPMsg String Echo FreeRTOS Demo...
INFO: rpmsg_init             : RPMSG remote init ...
INFO: rpmsg_remote_init      : waiting for link establish ...
INFO: rpmsg_remote_init      : RPMSG link up
INFO: rpmsg_create_ept       : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
INFO: rpmsg_create_ept       : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
INFO: rpmsg_create_ept       : Sending Nameservice string: rpmsg-virtual-tty-
channel-1
ept6: Get Message From Master Side : "hello world!" [len : 12]
ept7: Get Message From Master Side : "hello world!" [len : 12]
ept8: Get Message From Master Side : "hello world!" [len : 12]
 >
```

- Or start up Linux, then using remoteproc to start RTOS
  1. Start up Linux

```
# Running on i.MX 95 15x15 EVK
u-boot=> setenv fdtfile imx95-15x15-evk-multicore-rpmsg.dtb
# Running on i.MX 95 19x19 EVK
u-boot=> setenv fdtfile imx95-19x19-evk-multicore-rpmsg.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run bsp_bootcmd
```

  2. Start the first A55 RTOS
     After Linux boot up, start the first A-Core RTOS on A55 Core4:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
rpmsg-str-echo-freertos/rpmsg_str_echo_ca55_RTOS0_UART3.elf >  /sys/devices/
platform/remoteproc-ca55-4/remoteproc/remoteproc3/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/
remoteproc-ca55-4/remoteproc/remoteproc3/state
```

  3. Start the second A55 RTOS
     Then start the second A-Core RTOS on A55 Core5:

```
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/rpmsg-
str-echo-freertos/rpmsg_str_echo_ca55_RTOS1_RAM_CONSOLE-0xd1fff000.elf >  /
sys/devices/platform/remoteproc-ca55-5/remoteproc/remoteproc4/firmware
root@imx95-19x19-lpddr5-evk:~# echo start >  /sys/devices/platform/
remoteproc-ca55-5/remoteproc/remoteproc4/state
```

     The RAM Console log for the second A-Core RTOS can be checked using the command:

```
root@imx95-19x19-lpddr5-evk:~# ram_console_dump -a 0xd1fff000 -r 1
```

  4. Install tty driver module:

```
root@imx95-19x19-lpddr5-evk:~# modprobe imx_rpmsg_tty
```

- Test RPMSG Communication
  1. After installing the `imx_rpmsg_tty.ko`, it creates the following RPMSG TTY devices:

```
root@imx95-19x19-lpddr5-evk:~# ls /dev/ttyRPMSG*
/dev/ttyRPMSG3  /dev/ttyRPMSG4  /dev/ttyRPMSG5  /dev/ttyRPMSG6  /dev/
ttyRPMSG7  /dev/ttyRPMSG8
```

     The char devices /dev/ttyRPMSG3 ~ 5 are three endpoints connected to Cortex-A Core RTOS0, the /dev/ttyRPMSG6 ~ 8 are three endpoints connected to Cortex-A Core RTOS1.

2. Use `echo` or `minicom` to verify the RPMSG communication between Linux RPMSG master and the RTOS RPMSG remote.
   For example, use `echo` to send a sample string to the RTOS0's RPMSG endpoint3:
   ```
   # Disable the local seting "echo" of all the RPMSG TTY devices
   root@imx95-19x19-lpddr5-evk:~# for i in {3..8}; do stty -F /dev/ttyRPMSG$i -
   echo; done
   root@imx95-19x19-lpddr5-evk:~# echo "hello" > /dev/ttyRPMSG3
   ```
   Then in the RTOS0's console, it displays the string received:
   ```
   ept3: Get Message From Master Side : "hello" [len : 5]
   ept3: Get New Line From Master Side
   ```
   Or use the `minicom` to open one of these RPMSG TTY devices, such as the RPMSG endpoint6 of RTOS1:
   ```
   root@imx95-19x19-lpddr5-evk:~# minicom -D /dev/ttyRPMSG6
   ```
   Then input some characters in the minicom console. The input character is sent to the RPMSG endpoint6 of RTOS1 from Linux. Then the application running on RTOS1 echoes the received character back to Linux. For example, if you input the characters "hello world!", the below log would be displayed:
   ```
   Welcome to minicom 2.8

   OPTIONS: I18n
   Compiled on Jan  1 2021, 17:45:55.
   Port /dev/ttyRPMSG6, 08:36:41

   Press CTRL-A Z for help on special keys

   hello world!
   ```

3.5 RPMSG based resource sharing

3.5.1 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 shares 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.

3.5.2 Software architecture and design

This chapter describes different software architectures based on different technologies.

3.5.3 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.

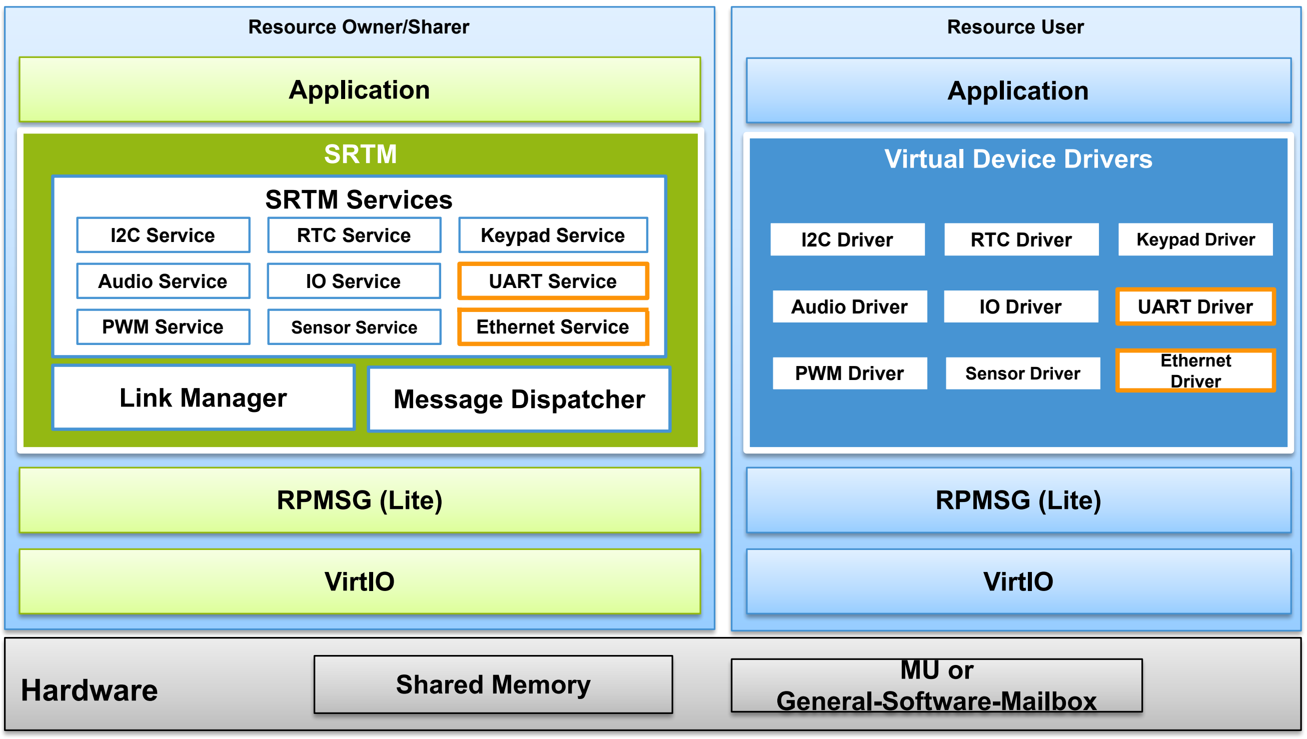


Figure 33. Resource sharing software architecture

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.

3.5.3.1 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.
- The virtual UART driver on Linux provides a standard UART device service to applications.
- Multiple virtual UART ports are provided in Linux.
- Each virtual UART port in Linux can map to a dedicated physical UART on FreeRTOS.
- Multiple virtual UART ports can be mapped to the same physical UART.

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

It includes the following software components:

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

Figure 34 illustrates the software architecture of a UART sharing design.

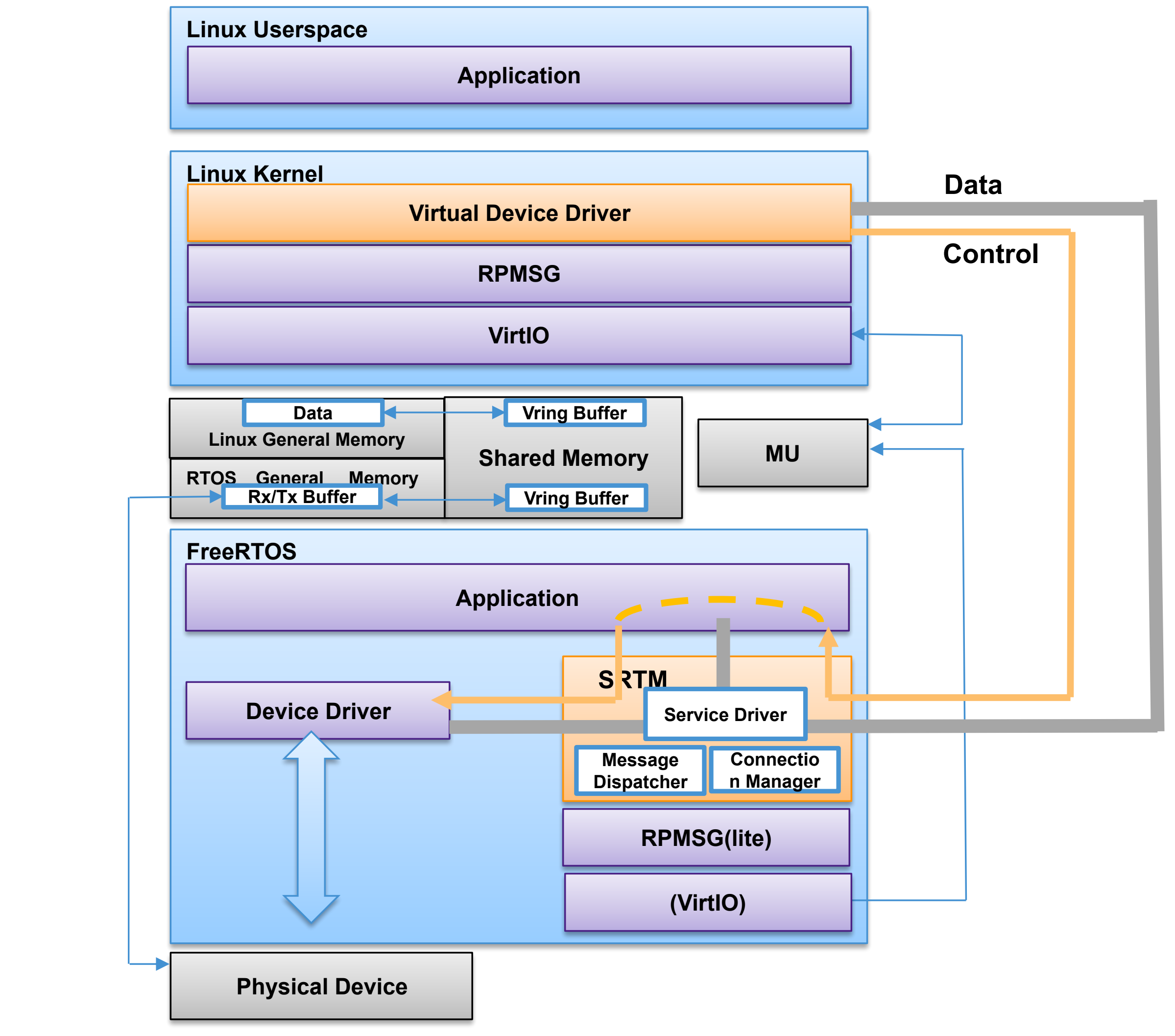


Figure 34. UART sharing software architecture

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 Table 35.

Table 35. Packet format for UART sharing

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	,			

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

- The "`Start flags`" field is used to figure out the start of data packets. Users can configure start flags with specified characters and size. The default start flags are 4 bytes: "$UT,".
- The "`Address`" field is reused by receive from transmit directions. For the receive direction (blue colored path in Figure 34), it is the destination address or ID of target virtual device. For the transmit direction (orange colored path in Figure 34), it is the source address or ID from which virtual device is transmitted.

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

3.5.4 Source code files and configuration

1. **Source code files**:
   The source files for different software components are listed in the Table 36:

   Table 36. 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, the Real-time Edge kernel builds the virtual UART driver as module (`rpmsg_tty.ko`) by enabling the configurable 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 the A-core have 1:1 mapping to physical UART on the M-core.
      - Each physical UART connects to a different device.
      - Each virtual UART uses a dedicated RPMsg endpoint.

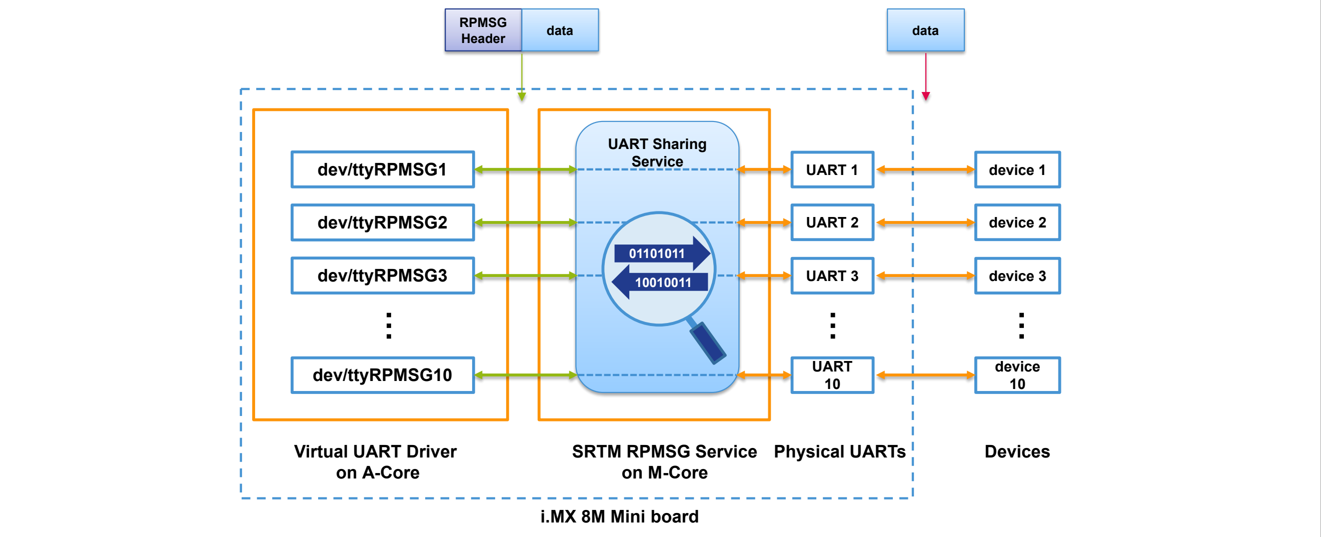


Figure 35. Virtual UART to physical UART 1:1 mapping

   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.

- A multiple UART Header is used to establish multiple virtual UART channels on a single physical UART connect. For details about multiple UART Headers, refer to the section Section 3.5.3.1.

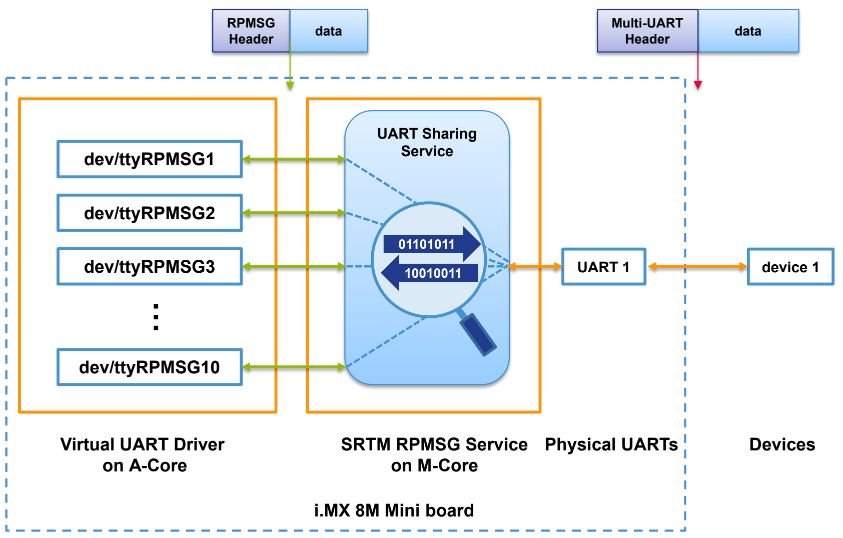


Figure 36. Virtual UART to physical UART n:1 mapping

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.

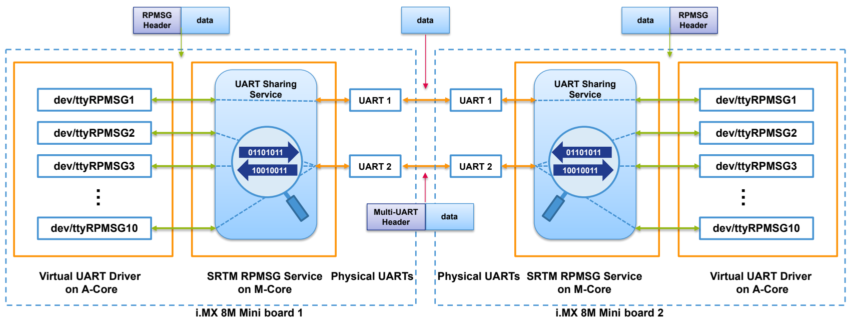


Figure 37. Virtual UART to physical UART flexible mapping

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`" specifies the physical UART instance ID that this virtual UART maps to. If the property of "`bus_id`" is not configured, the message sent from Linux to this virtual UART is display on M-core's debug console directly.*
- *Physical UART ID is configured in the FreeRTOS application "`rpmsg_lite_uart_sharing_rtos`".*
- *On i.MX 8M Mini LPDDR4 EVK, physical UART3 can be used, so all virtual UART ports are mapped to physical UART3 by default.*

- *On i.MX 93 EVK, physical LPUART5 can be used, so all virtual UART ports are mapped to physical LPUART5 by default.*
- *If `flags` is set with the value `IMX_SRTM_UART_SUPPORT_MULTI_UART_MSG_FLAG`, the multiple virtual UART is mapped to a single physical UART instance specified by `bus_id` (that implies that multiple virtual UART protocol packet headers are used).*
- *If `flags` is not set, this virtual UART is mapped 1:1 with physical UART instance specified by `bus_id`.*

By default, there are 11 virtual UARTs in the dtb file `imx8mm-evk-rpmsg.dtb`" for i.MX 8M Mini LPDDR4 EVK and `imx93-11x11-evk-rpmsg.dtb` for i.MX 93 EVK.

- *The virtual UART `0` to `9` are n:1 mapped to physical UART.*
- *The virtual UART `10` has no `bus_id` and displays messages sent from Linux to M-core's debug console.*

3.5.5 Building and running the demo on i.MX 8M Mini LPDDR4 EVK

3.5.5.1 Hardware setup for i.MX 8M Mini EVK

Use flying wire to connect UART3 between two i.MX 8M Mini EVK boards. UART3's pin is provided in J1003 connector; use the following pin connection between the two boards.

Table 37. 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

3.5.5.2 Building the demo images

The demo images "`rpmsg_uart_sharing_cm4.bin`" are by default compiled with the i.MX 8M Mini LPDDR4 EVK target image compiling, and are installed into the "`/examples`" directory of the target rootfs.

Or the image can be built separately by using the following Yocto command:

```
DISTRO=nxp-real-time-edge MACHINE=imx8mm-lpddr4-evk bitbake rpmsg-uart-sharing
```

The image can be found on directory "`<image-build-dir>/tmp/deploy/images/imx8mmevk/examples/`" on building host.

3.5.5.3 Running the i.MX 8M Mini EVK demo

1. Connect two i.MX 8M Mini EVK boards by following the steps in section of "Hardware Setup".
2. Connect two i.MX 8M 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 the above change permanent, execute the following command once:
   ```
   u-boot => saveenv
   ```
6. Then, use the following commands to download and run FreeRTOS image:
   ```
   u-boot => ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/
   rpmsg-uart-sharing-freertos/rpmsg_uart_sharing_cm4.bin;
   u-boot => cp.b 0x48000000 0x7e0000 20000; bootaux 0x7e0000
   ```
   Then, FreeRTOS debug console displays the following log:
   ```
   #################### RPMSG UART SHARING DEMO ####################
           Build Time: Mar 2 2022--09:38:19
           ********************************
           Wait the Linux kernel boot up to create the link between M core and A
    core.
   ```

```
         ********************************
```

7. Then, boot the Linux kernel by executing the following commands:

   ```
   u-boot => setenv jh_clk clk_ignore_unused
   u-boot => boot
   ```

   After the Linux kernel boots up, the below line on the FreeRTOS debug console 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*
      ```

      If the default `dtb` file `imx8mm-evk-rpmsg.dtb` is used, 11 device files from “`/dev/ttyRPMSG0`” to “`/dev/ttyRPMSG10`” must be displayed. The “`/dev/ttyRPMSG0`” to “`/dev/ttyRPMSG9`” have n:1 mapping to physical UART3. The “`/dev/ttyRPMSG10`” is without “`bus_id`” and displays the message sent from Linux to M-core’s debug console.
   b. Check each virtual UART from “`/dev/ttyRPMSG0`” to “`/dev/ttyRPMSG9`” is connected to peer virtual UART between two boards, for example, use "minicom" to open and configure the same virtual UART on both boards:

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

      Then, configure the UART as shown in the following figures:

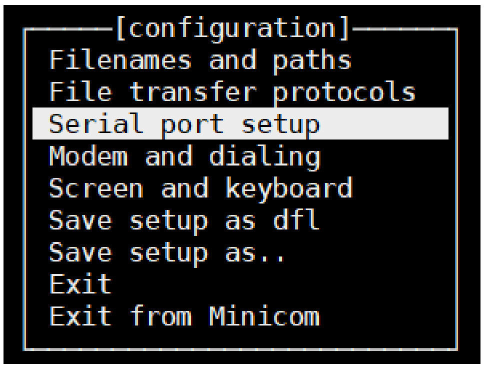
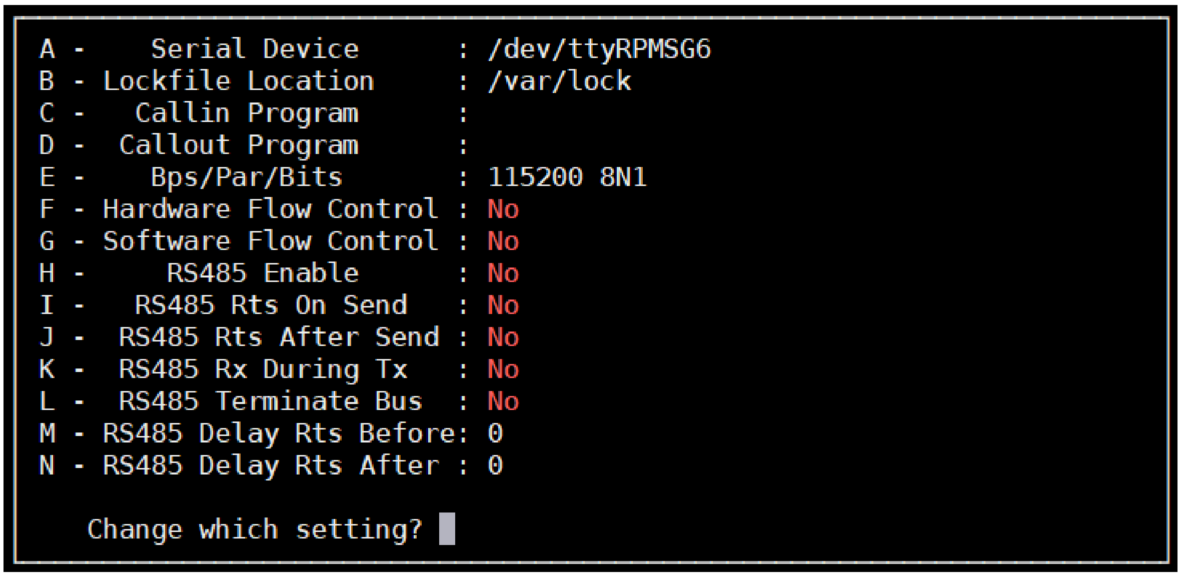


Figure 38. Configuring RPMSG Virtual UART Step1



```
A -    Serial Device      : /dev/ttyRPMSG6
B - Lockfile Location     : /var/lock
C -   Callin Program      :
D -  Callout Program      :
E -    Bps/Par/Bits       : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No
H -     RS485 Enable      : No
I -   RS485 Rts On Send   : No
J -  RS485 Rts After Send : No
K -  RS485 Rx During Tx   : No
L -  RS485 Terminate Bus  : No
M - RS485 Delay Rts Before: 0
N - RS485 Delay Rts After : 0

   Change which setting?
```

Figure 39. Configuring RPMSG Virtual UART Step2

Save the settings and back to minicom main window, then input any characters in one board's minicom window. These characters would be displayed on the other board's minicom window.

3.5.6 Building and running the demo on i.MX 93 EVK and i.MX 93 14x14 EVK

3.5.6.1 Hardware setup for i.MX 93 EVK and i.MX 93 14x14 EVK

Use flying wire to connect LPUART5 between two i.MX 93 EVK or i.MX 93 14x14 EVK boards.

- On i.MX 93 EVK, LPUART5's pin is provided in J1001 connector.
- On i.MX 93 14x14 EVK, LPUART5's pin is provided in J16 connector.

Use the following pin connection between the two boards.

Table 38. Pin connections between two i.MX 93 EVK or i.MX 93 14x14 EVK boards

Board 1		Connection	Board 2	
Pin	Function		Pin	Function
30	GND	<->	30	GND
28	LPUART5_RX	<->	27	LPUART5_TX
27	LPUART5_TX	<->	28	LPUART5_RX

3.5.6.2 Building the demo images

The demo image "`rpmsg_uart_sharing_cm33.bin`" is by default compiled with the i.MX 93 EVK and and i.MX 93 14x14 EVK target image compiling, and are installed into the "`/examples`" directory of the target rootfs.

Or the image can be built separately by using the following Yocto command:

```
# For i.MX 93 EVK
DISTRO=nxp-real-time-edge MACHINE=imx93evk source real-time-edge-setup-env.sh -b
 <build_dir>
```

```
# For i.MX93 14x14 EVK
DISTRO=nxp-real-time-edge MACHINE=imx93-14x14-lpddr4x-evk source real-time-edge-
setup-env.sh -b <build_dir>

bitbake rpmsg-uart-sharing
```

The image can be found on directory "`<image-build-dir>/tmp/deploy/images/<MACHINE>/examples/`" on building host.

3.5.6.3 Running the demo on i.MX 93 EVK and i.MX 93 14x14 EVK

1. Connect two i.MX 93 EVK or i.MX 93 14x14 EVK boards by following the steps listed in Section 3.5.6.1.
2. Connect two i.MX 93 EVK or i.MX 93 14x14 EVK boards to your PC via USB cable between the USB-UART connector and the PC USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number, four debug consoles for each board. Use the third one for the Linux debug console and the fourth one for the FreeRTOS debug console.
4. Deploy Real-time Edge release root files in SD card and modify the on-board switch to boot from MicroSD card.
5. Power on the board and enter into U-Boot command line. Then, execute the following command:

   ```
   # Running on i.MX 93 EVK
   u-boot => setenv fdtfile imx93-11x11-evk-uart-sharing-cm33.dtb
   # Running on i.MX 93 14x14 EVK
   u-boot => setenv fdtfile imx93-14x14-evk-uart-sharing-cm33.dtb
   ```

   To make changes permanent, execute the following commands once (after `setenv` above):

   ```
   u-boot => saveenv
   ```

6. Then, use the following command to download and run FreeRTOS image:

   ```
   u-boot => ext4load mmc 1:2 0x80000000 /examples/heterogeneous-multicore/
   rpmsg-uart-sharing-freertos/rpmsg_uart_sharing_cm33.bin
   u-boot => cp.b 0x80000000 0x201e0000 0x10000
   u-boot => bootaux 0x1ffe0000 0
   ```

   Then, FreeRTOS debug console would display the following log:

   ```
   ####################  RPMSG UART SHARING DEMO  ####################
       Build Time: Apr  5 2011 23:00:00
   Start SRTM communication
   ********************************
   Wait for the Linux kernel boot up to create the link between M core and A
    core.
   ```

   ****

7. And then, boot Linux kernel by executing the following command:

   ```
   u-boot => setenv jh_clk clk_ignore_unused
   u-boot => boot
   ```

8. After the Linux kernel boots up, in the FreeRTOS console, an extra line of log as shown below indicates that RPMSG connection between Cortex-A core and Cortex-M core has been established:

   ```
   Task A is working now.
   ```

Execute the above steps (1 to 7) on each of the two boards.

9. After Linux boots up, enter Linux command line, use the following commands to test the demo:
   a. Check device files are available:

   ```
   root @imx93evk:~# ls /dev/ttyRPMSG*
   ```

   There should be 11 device files from "`/dev/ttyRPMSG0`" to "`/dev/ttyRPMSG10`". The "`/dev/ttyRPMSG0`" to "`/dev/ttyRPMSG9`" have n:1 mapping to physical LPUART5, "`/dev/ttyRPMSG10`" is without "`bus_id`" and displays the message sent from Linux to M-core's debug console.
   b. Check each virtual UART from "`/dev/ttyRPMSG0`" to "`/dev/ttyRPMSG9`" is connected to peer virtual UART between two boards, for example, execute the following on the first board:
      - Use "minicom" to open and configure the same virtual UART on both boards:

      ```
      root@imx93evk:~# minicom -s -D /dev/ttyRPMSG6
      ```

      Then configure the UART as shown in the following figures:

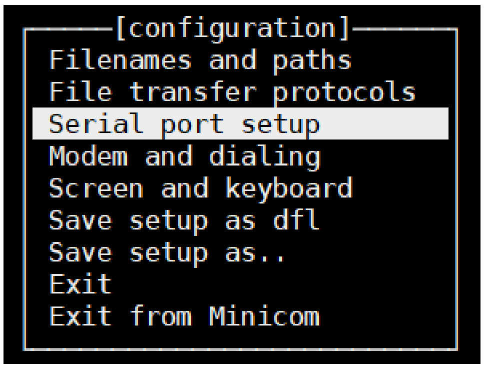
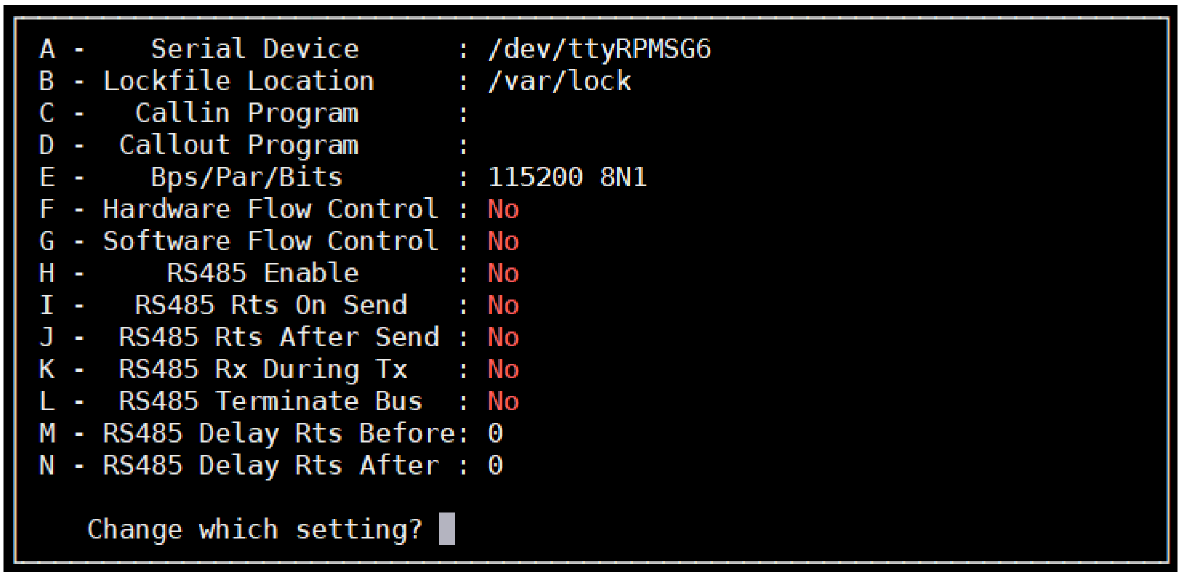


Figure 40. Configure RPMSG Virtual UART Step1



```
A -    Serial Device      : /dev/ttyRPMSG6
B - Lockfile Location     : /var/lock
C -   Callin Program      :
D -  Callout Program      :
E -    Bps/Par/Bits       : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No
H -     RS485 Enable      : No
I -   RS485 Rts On Send   : No
J -  RS485 Rts After Send : No
K -  RS485 Rx During Tx   : No
L -  RS485 Terminate Bus  : No
M - RS485 Delay Rts Before: 0
N - RS485 Delay Rts After : 0

   Change which setting?
```

Figure 41. Configure RPMSG Virtual UART Step2

Save the settings and go back to minicom main window, then input any characters in one board's minicom window. Then, the characters would be displayed on the other board's minicom window.

3.5.7 Building and running the demo on i.MX 93 QSB

3.5.7.1 Hardware setup for i.MX 93 QSB

Use flying wire to connect LPUART5 between two i.MX 93 QSB boards. LPUART5's pin is provided in J1401 connector. Use the following pin connection between the two boards.

Table 39. PIN connection between two i.MX 93 QSB boards

i.MX 93 QSB Board1		Connection	i.MX 93 QSB Board2	
Pin	Function		Pin	Function
30	GND	<->	30	GND
28	LPUART5_RX	<->	27	LPUART5_TX
27	LPUART5_TX	<->	28	LPUART5_RX

3.5.7.2 Building the demo images

The demo image "`rpmsg_uart_sharing_cm33.bin`" is by default compiled with the i.MX 93 QSB target image compiling, and are installed into the "`/examples`" directory of the target rootfs.

Or the image can be built separately by using the following Yocto command:

```
DISTRO=nxp-real-time-edge MACHINE=imx93-9x9-lpddr4-qsb source real-time-edge-
setup-env.sh -b <build_dir>

bitbake rpmsg-uart-sharing
```

The image can be found on directory "`<image-build-dir>/tmp/deploy/images/imx93-9x9-lpddr4-qsb/examples/`" on building host.

3.5.7.3 Running the demo on i.MX 93 QSB

1. Connect two i.MX 93 QSB boards by following the steps listed in Section 3.5.7.1.
2. Connect two i.MX 93 QSB boards to your PC via USB cable between the USB-UART connector and the PC USB connector.
3. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number, four debug consoles for each board. Use the third one for the Linux debug console and the fourth one for the FreeRTOS debug console.
4. Deploy Real-time Edge release root files in SD card and modify the on-board switch to boot from MicroSD card.
5. Power on the board and enter into U-Boot command line. Then, execute the following command:
   ```
   u-boot => setenv fdtfile imx93-9x9-qsb-uart-sharing-cm33.dtb
   ```
   To make changes permanent, execute the following commands once (after `setenv` above):
   ```
   u-boot => saveenv
   ```
6. Then, use the following command to download and run FreeRTOS image:
   ```
   u-boot => ext4load mmc 1:2 0x80000000 /examples/rpmsg-uart-sharing-freertos/
   rpmsg_uart_sharing_cm33.bin
   u-boot => cp.b 0x80000000 0x201e0000 0x10000
   u-boot => bootaux 0x1ffe0000 0
   ```

Then, FreeRTOS debug console would display the following log:

```
####################  RPMSG UART SHARING DEMO  ####################
 Build Time: Oct 30 2023 11:20:34
 Start SRTM communication
 ********************************
  Wait for the Linux kernel boot up to create the link between M core and A
 core.
```

****

7. And then, boot Linux kernel by executing the following command:

```
u-boot => setenv jh_clk clk_ignore_unused
u-boot => boot
```

8. After the Linux kernel boots up, in the FreeRTOS console, an extra line of log as shown below indicates that RPMSG connection between Cortex-A core and Cortex-M core has been established:

```
Task A is working now.
```

   Execute the above steps (1 to 7) on each i.MX 93 QSB board.

9. After Linux boots up, enter Linux command line using the following commands to test the demo:
   a. Check the device files are available:

   ```
   root@imx93-9x9-lpddr4-qsb:~# ls /dev/ttyRPMSG*
   ```

   There should be 11 device files from “`/dev/ttyRPMSG0`” to “`/dev/ttyRPMSG10`” if the default dtb file `imx93-9x9-qsb-uart-sharing-cm33.dtb` is used. The “`/dev/ttyRPMSG0`” to “`/dev/ttyRPMSG9`” have n:1 mapping to physical LPUART5, “`/dev/ttyRPMSG10`” is without “`bus_id`” and displays the message sent from Linux to M-core’s debug console.

   b. Check each virtual UART from “`/dev/ttyRPMSG0`” to “`/dev/ttyRPMSG9`” is connected to peer virtual UART between two boards, for example, ues "minicom" to open and configure the same virtual UART on both boards:

   ```
   root@imx93-9x9-lpddr4-qsb:~# minicom -s -D /dev/ttyRPMSG6
   ```

   Then configure the UART as shown in the following figures:

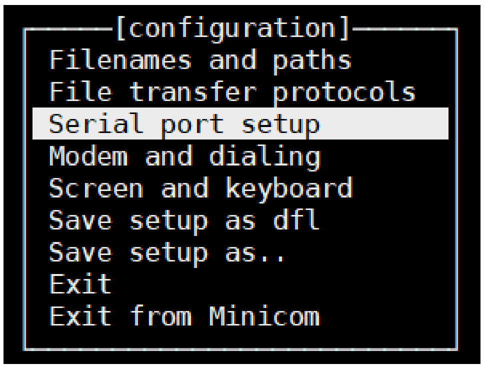
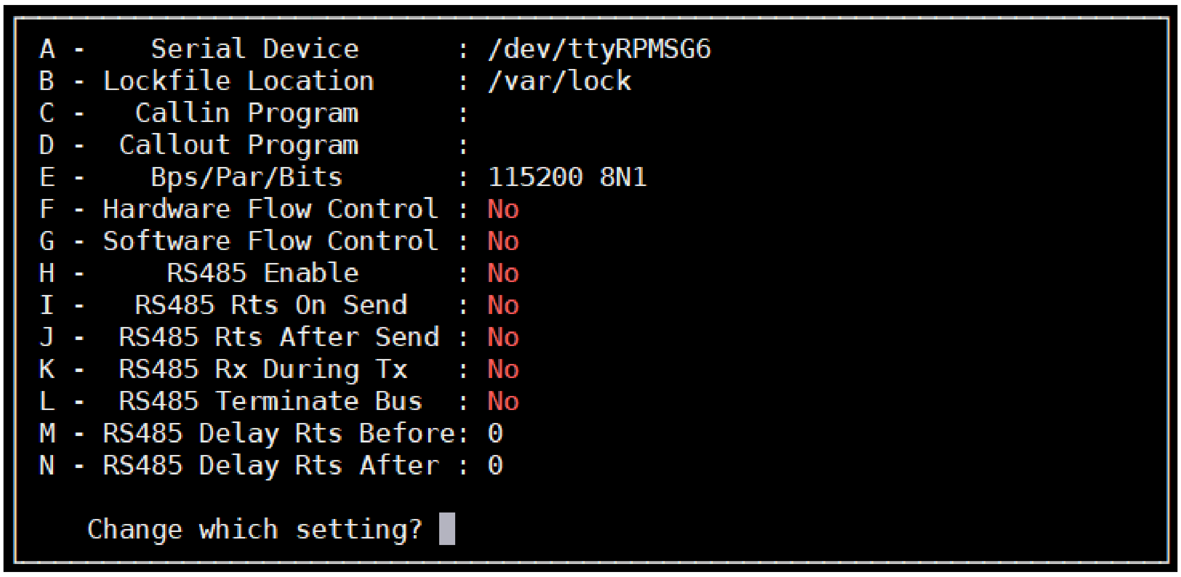


Figure 42. Configure RPMSG Virtual UART Step1



```
A -    Serial Device      : /dev/ttyRPMSG6
B - Lockfile Location     : /var/lock
C -   Callin Program      :
D -  Callout Program      :
E -    Bps/Par/Bits       : 115200 8N1
F - Hardware Flow Control : No
G - Software Flow Control : No
H -     RS485 Enable      : No
I -   RS485 Rts On Send   : No
J -  RS485 Rts After Send : No
K -  RS485 Rx During Tx   : No
L -  RS485 Terminate Bus  : No
M - RS485 Delay Rts Before: 0
N - RS485 Delay Rts After : 0

   Change which setting?
```

Figure 43. Configure RPMSG Virtual UART Step2

Save the settings and go back to minicom main window, then input any characters in one board's minicom window. Then, the characters would be displayed on the other board's minicom window.

3.6 Heterogeneous Multicore VirtIO and networking sharing

3.6.1 Heterogeneous Multicore VirtIO

Heterogeneous Multicore VirtIO applies para-virtualization VirtIO technology to build resource sharing between Heterogeneous asymmetric multiprocessing (AMP). The main difference from para-virtualization VirtIO is that Heterogeneous Multicore VirtIO does not use and depend on any hypervisor. Therefore, it can be used for resource sharing between Cortex-A and Cortex-M cores, or between multiple Cortex-A cores.

The VirtIO is a standard for para-virtualization to provide high-performance IO device virtualization for VM. The Figure 44 shows the architecture of the VirtIO solution.

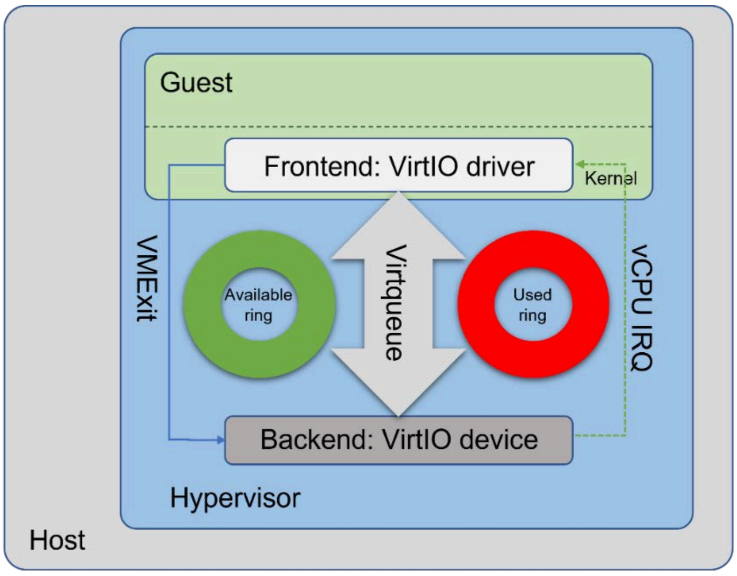


Figure 44. Para-virtualization VirtIO on hypervisor

The front-end VirtIO driver runs in the guest kernel space, and the backend VirtIO device runs in the Hypervisor. The Virtqueue and Vring provide data transfer capability via shared memory. The hypervisor emulates the device logic by VMExit and injecting vCPU IRQ. Therefore, para-virtualization VirtIO is used for resource sharing between Virtual Machine guest OS and host OS, and it depends on the Hypervisor to run the VirtIO backend.

The Heterogeneous Multicore VirtIO in Real Time Edge uses VirtIO technology. However, it runs VirtIO backend on any CPU core including Cortex-A core and Cortex-M core. The VirtIO front-end runs on any other CPU core. This technology uses VirtIO to establish communication between the front-end and backend, which run on different CPU cores. Therefore, VirtIO can be used to share hardware resources between different CPU cores.

In the current implementation, the backend runs on RTOS and owns the hardware resources, such as peripherals. The front-end runs on Linux. As there is no hypervisor providing VMExit and vCPU IRQ injecting mechanism, the hardware or software mailbox between the front-end and back-end is needed. Figure 45 shows the architecture of the Heterogeneous Multicore VirtIO.

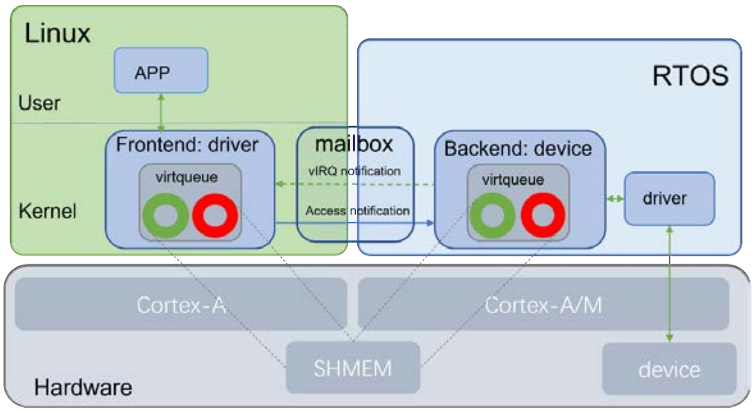


Figure 45. Heterogeneous Multicore VirtIO

Heterogeneous Multicore VirtIO uses shared memory to build Vring structure and data buffers. The shared memory must be coherent between front end and Backend.

3.6.2 Heterogeneous Multicore VirtIO performance evaluation

3.6.2.1 Overview

A VirtIO transmission device is introduced for evaluating the performance between the frontend and backend through virtqueues. There are 2 virtqueues for transmitting and receiving directions separately, and the device configuration registers are used to configure and control the test cases.

The Table 40 lists the combination of supported cases.

Table 40. Heterogeneous Multicore VirtIO performance evaluation

Direction	Pkt size	Frontend Buffer copy	Backend Buffer copy
TX (Linux -> FreeRTOS)	Max 2KB	Y/N	Y/N
RX (FreeRTOS -> Linux)	Max 2KB	Y/N	Y/N

3.6.2.2 Building VirtIO performance evaluation backend application

VirtIO performance evaluation backend application runs on FreeRTOS, it is an application named “virtio_perf” in the repo: heterogeneous-multicore.

Refer to "Building Heterogeneous Multicore RTOS Application" for steps to build the application.

3.6.2.3 Running VirtIO performance testing

The VirtIO performance testing supports running on i.MX 8M Mini LPDDR4 EVK and i.MX 8M Plus LPDDR4 EVK.

Note: *The released image does not display the CPU load status of the RTOS side during the evaluation, to display the CPU load status, build the debug type image manually referring to the steps listed in Section 3.2.2.1.*

Perform the following steps for VirtIO performance testing:

1. **Set up the UART console for the frontend and backend**
   Connect the DEBUG UART slot on the board to your PC through the USB Cable. On the PC, this step creates two USB serial ports (port0 and port1) for i.MX 8M Mini EVK board, and four USB serial ports (port0 ~ port3) for i.MX 8M Plus EVK board. Open two UART consoles for UART port0 and port1 on i.MX 8M Mini EVK board or port2 and port3 on i.MX 8M Plus EVK board with the following setup:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit

   The first UART console is used for Linux that runs the VirtIO frontend, the another is used for RTOS that runs the VirtIO backend.
2. **Boot up VirtIO backend FreeRTOS and frontend Linux**
   Heterogeneous Multicore VirtIO Backend can run on Cortex-A core or Cortex-M core to evaluate different use cases.
   - **For i.MX 8M Mini LPDDR4 EVK**

Users can use U-Boot commands or Linux remoteproc to start the backend RTOS. Choose one method to start RTOS and Linux:

– Using U-Boot Commands to start RTOS, then start Linux

a. Run the backend on Cortex-M Core
   On the U-Boot command line, execute the following commands to boot Cortex-M core with backend firmware:

```
u-boot => ext4load mmc 1:2 0x48000000 /examples/heterogeneous-
multicore/virtio-perf-freertos/virtio_perf_cm4.bin
u-boot => cp.b 0x48000000 0x7e0000 0x20000
u-boot => bootaux 0x7e0000
```

   Then, boot Linux kernel:

```
u-boot => setenv fdtfile imx8mm-evk-virtio-perf-cm4.dtb
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

b. Or run the backend on Cortex-A Core
   Execute the following command in U-Boot command line:

```
u-boot => ext4load mmc 1:2 0x93c00000 /examples/heterogeneous-
multicore/virtio-perf-freertos/virtio_perf_ca53.bin
u-boot => dcache flush; icache flush; cpu 3 release 0x93c00000
```

   Then, boot Linux kernel:

```
u-boot => setenv fdtfile imx8mm-evk-virtio-perf-ca53.dtb
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

– Or start Linux, then use remoteproc to start RTOS

a. Run the backend on Cortex-M Core
   Boot up Linux:

```
u-boot => run prepare_mcore; mw.l 0xb8400000 0 1
u-boot => setenv fdtfile imx8mm-evk-virtio-perf-cm4.dtb
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

   After Linux boot up, start backend RTOS running on Cortex-M Core by using remoteproc under Linux:

```
root@imx8mm-lpddr4-evk:~# modprobe -r virtio_trans
root@imx8mm-lpddr4-evk:~# modprobe -r virtio_mmio
root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/
virtio-perf-freertos/virtio_perf_cm4.elf > /sys/devices/platform/
imx8mm-cm4/remoteproc/remoteproc0/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mm-
cm4/remoteproc/remoteproc0/state
root@imx8mm-lpddr4-evk:~# modprobe virtio_mmio
root@imx8mm-lpddr4-evk:~# modprobe virtio_trans
```

b. Or run the backend on Cortex-A Core
   Boot up Linux:

```
u-boot => setenv fdtfile imx8mm-evk-virtio-perf-ca53.dtb ; mw.l
 0xb8400000 0 1
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

After Linux boot up, start backend RTOS running on Cortex-A Core by using remoteproc under Linux:

```
root@imx8mm-lpddr4-evk:~# modprobe -r virtio_trans
root@imx8mm-lpddr4-evk:~# modprobe -r virtio_mmio
root@imx8mm-lpddr4-evk:~# echo /examples/heterogeneous-multicore/
virtio-perf-freertos/virtio_perf_ca53.elf > /sys/devices/platform/
remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/
remoteproc-ca53-3/remoteproc/remoteproc2/state
root@imx8mm-lpddr4-evk:~# modprobe virtio_mmio
root@imx8mm-lpddr4-evk:~# modprobe virtio_trans
```

- **For i.MX 8M Plus LPDDR4 EVK**
  Can use U-Boot commands or Linux remoteproc to start backend RTOS, choose one method to start RTOS and Linux:
  - **Using U-Boot Commands to start RTOS, then start Linux**
    a. Run the backend on Cortex-M Core
    On the U-Boot command line, execute the following commands to boot Cortex-M core with backend firmware:

```
u-boot => ext4load mmc 1:2 0x48000000 /examples/heterogeneous-
multicore/virtio-perf-freertos/virtio_perf_cm7.bin
u-boot => cp.b 0x48000000 0x7e0000 0x20000
u-boot => bootaux 0x7e0000
```

Then, boot Linux kernel:

```
u-boot => setenv fdtfile imx8mp-evk-virtio-perf-cm7.dtb
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

b. Run the backend on Cortex-A Core
Execute the following command in U-Boot command line:

```
u-boot => ext4load mmc 1:2 0xc0000000 /examples/heterogeneous-
multicore/virtio-perf-freertos/virtio_perf_ca53.bin
u-boot => dcache flush; icache flush; cpu 3 release 0xc0000000
```

Then, boot Linux kernel:

```
u-boot => setenv fdtfile imx8mp-evk-virtio-perf-ca53.dtb
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

  - **Or start Linux, then use remoteproc to start RTOS**
    a. Run the backend on Cortex-M Core
    Boot up Linux:

```
u-boot => run prepare_mcore; mw.l 0xb8400000 0 1
u-boot => setenv fdtfile imx8mp-evk-virtio-perf-cm7.dtb
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

After Linux boot up, start backend RTOS running on Cortex-M Core by using remoteproc under Linux:

```
root@imx8mp-lpddr4-evk:~# modprobe -r virtio_trans
root@imx8mp-lpddr4-evk:~# modprobe -r virtio_mmio
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/
virtio-perf-freertos/virtio_perf_cm7.elf > /sys/devices/platform/
imx8mp-cm7/remoteproc/remoteproc0/firmware
```

```
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mp-
cm7/remoteproc/remoteproc0/state
root@imx8mp-lpddr4-evk:~# modprobe virtio_mmio
root@imx8mp-lpddr4-evk:~# modprobe virtio_trans
```

b. Run the backend on Cortex-A Core
   Boot up Linux:

```
u-boot => setenv fdtfile imx8mp-evk-virtio-perf-ca53.dtb; mw.l
 0xfc700000 0 1
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

   After Linux boots up, start backend RTOS running on Cortex-A Core by using remoteproc under Linux:

```
root@imx8mp-lpddr4-evk:~# modprobe -r virtio_trans
root@imx8mp-lpddr4-evk:~# modprobe -r virtio_mmio
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/
virtio-perf-freertos/virtio_perf_ca53.elf > /sys/devices/platform/
remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/
remoteproc-ca53-3/remoteproc/remoteproc2/state
root@imx8mp-lpddr4-evk:~# modprobe virtio_mmio
root@imx8mp-lpddr4-evk:~# modprobe virtio_trans
```

3. **Use the “`vt_test.sh`” tool in Linux to start the performance testing use case.**
   The following is the help information of the tool.

```
root@imx8mm-lpddr4-evk:~# vt_test.sh -h
USAGE: vt_test.sh [-h] [-s pkt_size] [-r regression] [-t type] [-b backend
 copy] [-f frontend copy]
-s: Packet size: max 2048 Bytes, default: 64 Bytes
-r: Regression times: default: 1000
-t: Test type: 0: TX (frontend to backend); 1: RX (backend to frontend)
-b: Backend copy buffer option:  0: not copy; 1: copy
-f: Frontend copy buffer option: 0: not copy; 1: copy
-h: This USAGE info
```

   a. “-s” specifies the packet size to be used for testing, such as “-s 64”, it uses 64-byte packets for testing.
   b. “-t” specifies the testing direction:
      - “-t 0” means the test sends packets from frontend (Linux) to backend(RTOS on A-Core or M-Core),
      - “-t 1” means the test sends packets from backend (RTOS on A-Core or M-Core) to frontend (Linux).
   c. “-r” specifies the regression times:
      - “-r 100000” indicates that the test case sends 100000 packets with the direction from backend to frontend or frontend to backend, which is specified by the “-t” parameter.
   d. “-b” specifies whether there is a memory copy in the backend:
      - For the memory copy case, use “-b 1”, for enabling memory copy from Vring buffer to user application buffer when receiving packet from fontend, or copy from user application buffer to Vring buffer when transmitting packets to frontend.
      - For no memory copy case “-b 0”, specifies there is no memory copy in backend for each packet receiving or transmitting.
   e. “-f” specifies whether there is a memory copy in the frontend.

   For example, see the command below:

```
vt_test.sh -s 64 -r 1000000 -t 0 -b 0 -f 0
```

The above test case transmits 1000000 packets from frontend (Linux) to backend (RTOS on A-Core or M-Core), each packet size is 64 bytes, there is no memory copy both on frontend and backend. The test log is as follows:

```
root@imx8mm-lpddr4-evk:~# vt_test.sh -s 64 -r 1000000 -t 0 -b 0 -f 0
[20.561527] *********************************************************
[20.561539] Front-end: interrupt mode
[20.561543] Back-end:  interrupt mode
[20.561544] Front-end: do NOT copy buffer
[20.561546] Back-end:  do NOTcopy buffer
[20.561547]  Test case: TX
[20.561547]  pkt_size: <64>
[20.561547]  regress times: <1000000>
[21.868494] tx_test: pkt_size (64 B), pkt_cnt (1000000), period (1298108 µs)
```

The log shows that 1000000 packets are transmitted from frontend to backend in 1298108 µs. Therefore, the performance is 770 kpps or 394 Mbit/s.

3.6.3 Heterogeneous Multicore VirtIO network sharing

3.6.3.1 Overview

Figure 46 shows the heterogeneous Multicore VirtIO network sharing architecture.

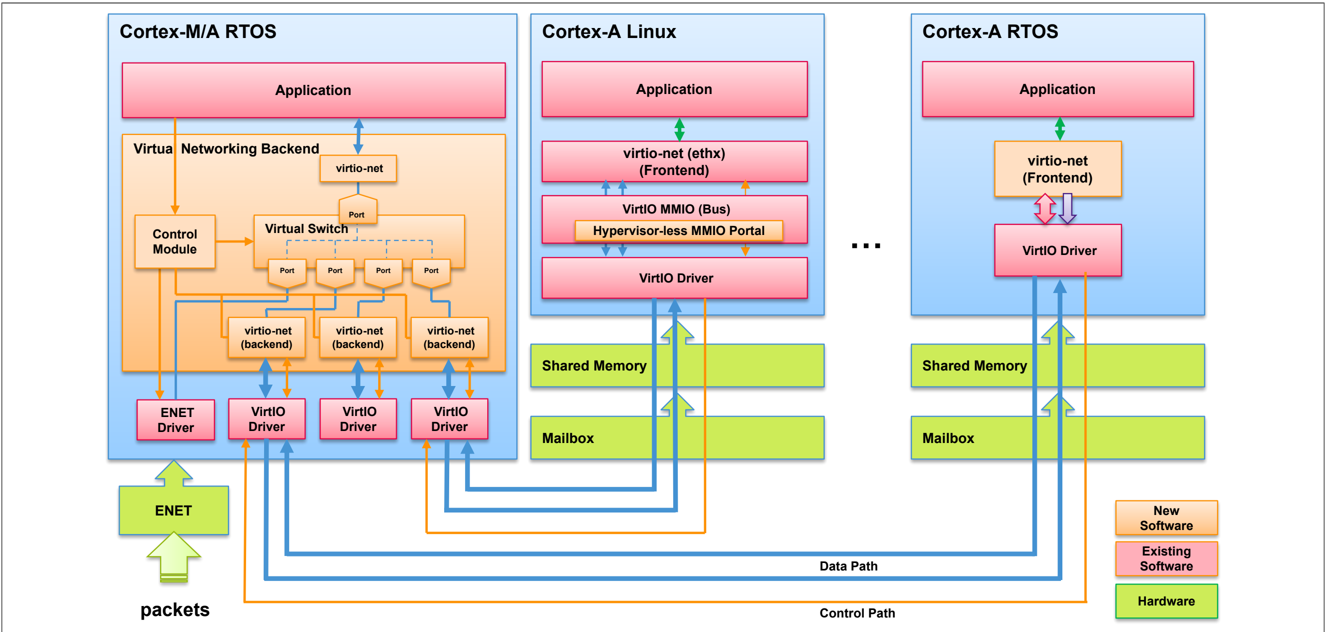


Figure 46. Heterogeneous Multicore VirtIO Networking Sharing

The virtual networking frontend runs on Cortex-A core, the frontend in Linux reuses the existing "`drivers/net/virtio_net.c`" driver by selecting kernel configuration item "`CONFIG_VIRTIO_NET`". RTOS frontend driver is not enabled in this release.

The Virtual Networking Backend can run in RTOS on Cortex-A core or Cortex-M core. The virtio-net backend drivers use Heterogeneous Multicore VirtIO to communicate with virtio-net frontend. It includes two data paths to handle data packets receiving/transmitting of frontend and one control path to handle control requirements from the frontend.

A Virtual Switch in the backend is used to switch packets from different ports. The switch ports include one “remote port” and many “local ports”. In general, a “remote port” is a physical Ethernet port such as physical ENET port, which is used to receive/transmit packets from/to the physical Ethernet port. Here, “local port” refers to the virtual software port, such as the local port for virtio-net backend used to receive/transmit from/to virtio-net backend. In fact, the packet is from/to virtio-net frontend through data path of Heterogeneous Multicore VirtIO and another type of “local port” is used to connect to “virt-net” on RTOS to provide virtual Ethernet interface for RTOS locally.

In the current implementation, an Ethernet L2 switch acts as a virtual switch. Each “local port” has a different MAC address, so the packets received from a “remote port” can be switched to the destination “local port” according to destination MAC address in the networking packets. The packets whose destination MAC address does not match any “local port” are discarded, except broadcast packets. For the packets received from the “local port”, the switch tries to check whether destination MAC address matches the address of other “local ports”. If a matching entry is found, the packet is switched to the matched “local port”, so that the virtual switch can implement switching packets between “local ports” locally. If a match is not found, the packets are sent to the external by “remote port”. That is to say Virtual Switch supports “local switch” and “remote switch”.

The Virtual Switch can connect to multiple “local port”, that is to say single physical Ethernet port, controlled by CPU Core running backend, can be shared with multiple OS running on different CPU Core by though multiple front-end, and local virtual Ethernet driver in back-end also provide Ethernet service for back-end CPU core locally.

3.6.3.2 Building VirtIO network sharing backend application

VirtIO network sharing backend application runs on FreeRTOS, it is an application named “virtio_net_backend” in the repo: heterogeneous-multicore.

Refer to "Building Heterogeneous Multicore RTOS Application" for how to build the application.

3.6.3.3 Running VirtIO network sharing

VirtIO Network Sharing supports running on i.MX 8M Mini, i.MX 8M Plus, and i.MX 93 platforms.

1. **Set up the UART console for the frontend and backend**
   Connect the DEBUG UART slot on the board to your PC through the USB Cable. On the PC, this step creates two USB serial ports (port0 and port1) for i.MX 8M Mini EVK board, and four USB serial ports (port0 ~ port3) for i.MX 8M Plus EVK board and i.MX 93 EVK board. Open two UART consoles for UART port0 and port1 on i.MX 8M Mini EVK board or port2 and port3 on i.MX 8M Plus EVK board and i.MX 93 EVK board UART with the following setup:
   - 115200
   - No parity
   - 8 data bits
   - 1 stop bit

   The first UART console is used for Linux that runs the VirtIO frontend. The UART console is used for RTOS, which runs the VirtIO backend.
2. **Hardware setup**
   Connect the ENET port on the selected EVK board to a networking switch or another board by using an Ethernet cable.
   For i.MX 8M Mini EVK board, there is a single Ethernet port on the board. So, use this port for testing.
   For i.MX 8M Plus EVK board and i.MX 93 EVK board, there are two Ethernet ports are on the board. The first one, which is close to DEBUG USB port is ENET port and it is used for VirtIO Networking Sharing. So connect this port to the networking link. Another Ethernet port is an ENET QoS port and it is not used for this demo.
3. **Booting backend and frontend**

Heterogeneous Multicore VirtIO backend can run on Cortex-A core or Cortex-M core.

- **For i.MX 8M Mini LPDDR4 EVK**
  Can use U-Boot commands or Linux remoteproc to start backend RTOS, choose one method to start RTOS and Linux:
  - **Use U-Boot Commands to start RTOS, then start Linux**
    a. Run the backend on a Cortex-M core
    On U-Boot command prompt, execute the following commands to boot Cortex-M core with firmware:

```
u-boot => ext4load mmc 1:2 0x48000000 /examples/heterogeneous-
multicore/virtio-net-backend-freertos/virtio_net_backend_cm4.bin
u-boot => cp.b 0x48000000 0x7e0000 0x20000
u-boot => bootaux 0x7e0000
```

    Then boot Linux kernel:

```
u-boot => setenv fdtfile imx8mm-evk-virtio-net-cm4.dtb
u-boot => setenv mmcargs $mmcargs mem=2048MB clk_ignore_unused
u-boot => run bsp_bootcmd
```

    b. Run the backend on a Cortex-A core
    Execute the following command in the U-Boot command line:

```
u-boot => ext4load mmc 1:2 0x93c00000 /examples/heterogeneous-
multicore/virtio-net-backend-freertos/virtio_net_backend_ca53.bin
u-boot => dcache flush; icache flush;
u-boot => cpu 3 release 0x93c00000
```

    Then, boot the Linux kernel:

```
u-boot => setenv fdtfile imx8mm-evk-virtio-net-ca53.dtb
u-boot => setenv mmcargs $mmcargs maxcpus=3 clk_ignore_unused
u-boot => run bsp_bootcmd
```

  - **Or start Linux, then use remoteproc to start RTOS**
    a. Run the backend on Cortex-M core
    Boot up Linux:

```
u-boot => run prepare_mcore; mw.l 0xb8400000 0 1
u-boot => setenv fdtfile imx8mm-evk-virtio-net-cm4.dtb
u-boot => setenv mmcargs $mmcargs mem=2048MB clk_ignore_unused
u-boot => run bsp_bootcmd
```

    After Linux boot up, start backend RTOS running on Cortex-M Core by using remoteproc under Linux:

```
root@imx8mm-lpddr4-evk:/# modprobe -r virtio_net
root@imx8mm-lpddr4-evk:/# modprobe -r virtio_mmio
root@imx8mm-lpddr4-evk:/# echo /examples/heterogeneous-multicore/
virtio-net-backend-freertos/virtio_net_backend_cm4.elf  > /sys/devices/
platform/imx8mm-cm4/remoteproc/remoteproc0/firmware
root@imx8mm-lpddr4-evk:/# echo start > /sys/devices/platform/imx8mm-
cm4/remoteproc/remoteproc0/state
root@imx8mm-lpddr4-evk:/# modprobe virtio_mmio
root@imx8mm-lpddr4-evk:/# modprobe virtio_net
```

    b. Run the backend on Cortex-A core
    Boot up Linux:

```
u-boot => setenv fdtfile imx8mm-evk-virtio-net-ca53.dtb; mw.l
 0xb8400000 0 1
u-boot => setenv mmcargs $mmcargs maxcpus=3 clk_ignore_unused
```

```
u-boot => run bsp_bootcmd
```

After Linux boot up, start backend RTOS running on Cortex-A Core by using remoteproc under Linux:

```
root@imx8mm-lpddr4-evk:/# modprobe -r virtio_net
root@imx8mm-lpddr4-evk:/# modprobe -r virtio_mmio
root@imx8mm-lpddr4-evk:/# echo /examples/heterogeneous-multicore/
virtio-net-backend-freertos/virtio_net_backend_ca53.elf  > /sys/
devices/platform/remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mm-lpddr4-evk:/# echo start >/sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
root@imx8mm-lpddr4-evk:/# modprobe virtio_mmio
root@imx8mm-lpddr4-evk:/# modprobe virtio_net
```

- **For i.MX 8M Plus LPDDR4 EVK**
  Can use U-Boot commands or Linux remoteproc to start backend RTOS, choose one method to start RTOS and Linux:
  - **Using U-Boot Commands to start RTOS, then start Linux**
    a. Run the backend on Cortex-M core
    U-Boot command prompt, execute the following commands to boot Cortex-M core with firmware:

```
u-boot => ext4load mmc 1:2 0x48000000 /examples/heterogeneous-
multicore/virtio-net-backend-freertos/virtio_net_backend_cm7.bin
u-boot => cp.b 0x48000000 0x7e0000 0x20000
u-boot => bootaux 0x7e0000
```

    Then, boot Linux kernel:

```
u-boot => setenv fdtfile imx8mp-evk-virtio-net-cm7.dtb
u-boot => setenv mmcargs $mmcargs mem=2048MB clk_ignore_unused
u-boot => run bsp_bootcmd
```

    b. Run the backend on Cortex-A core
    Execute the following command in U-Boot command line:

```
u-boot => ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-
multicore/virtio-net-backend-freertos/virtio_net_backend_ca53.bin
u-boot => dcache flush; icache flush;
u-boot => cpu 3 release 0xC0000000
```

    Then boot Linux kernel:

```
u-boot => setenv fdtfile imx8mp-evk-virtio-net-ca53.dtb
u-boot => setenv mmcargs $mmcargs maxcpus=3 clk_ignore_unused
u-boot => run bsp_bootcmd
```

  - **Or start Linux, then use remoteproc to start RTOS**
    a. Run the backend on Cortex-M core
    Boot up Linux:

```
u-boot => run prepare_mcore; mw.l 0xb8400000 0 1
u-boot => setenv fdtfile imx8mp-evk-virtio-net-cm7.dtb
u-boot => setenv mmcargs $mmcargs mem=2048MB clk_ignore_unused
u-boot => run bsp_bootcmd
```

    After Linux boot up, start backend RTOS running on Cortex-M Core by using remoteproc under Linux:

```
root@imx8mp-lpddr4-evk:/# modprobe -r virtio_net
root@imx8mp-lpddr4-evk:/# modprobe -r virtio_mmio
```

```
root@imx8mp-lpddr4-evk:/# echo /examples/heterogeneous-multicore/
virtio-net-backend-freertos/virtio_net_backend_cm7.elf  > /sys/devices/
platform/imx8mp-cm7/remoteproc/remoteproc0/firmware
root@imx8mp-lpddr4-evk:/# echo start > /sys/devices/platform/imx8mp-
cm7/remoteproc/remoteproc0/state
root@imx8mp-lpddr4-evk:/# modprobe virtio_mmio
root@imx8mp-lpddr4-evk:/# modprobe virtio_net
```

b. Run the backend on Cortex-A core
Boot up Linux:

```
u-boot => setenv fdtfile imx8mp-evk-virtio-net-ca53.dtb; mw.l
 0xfc700000 0 1
u-boot => setenv mmcargs $mmcargs maxcpus=3 clk_ignore_unused
u-boot => run bsp_bootcmd
```

After Linux boot up, start backend RTOS running on Cortex-A Core by using remoteproc under Linux:

```
root@imx8mp-lpddr4-evk:/# modprobe -r virtio_net
root@imx8mp-lpddr4-evk:/# modprobe -r virtio_mmio
root@imx8mp-lpddr4-evk:/# echo /examples/heterogeneous-multicore/
virtio-net-backend-freertos/virtio_net_backend_ca53.elf  > /sys/
devices/platform/remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mp-lpddr4-evk:/# echo start >/sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
root@imx8mp-lpddr4-evk:/# modprobe virtio_mmio
root@imx8mp-lpddr4-evk:/# modprobe virtio_net
```

- **for i.MX 93 EVK**
  Can use U-Boot commands or Linux remoteproc to start backend RTOS, choose one method to start RTOS and Linux:
  – Using U-Boot Commands to start RTOS, then start Linux

a. Run the backend on Cortex-M core
On U-Boot command prompt, execute the following commands to boot Cortex-M core with firmware:

```
u-boot => ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-
multicore/virtio-net-backend-freertos/virtio_net_backend_cm33.bin
u-boot => cp.b 0xd0000000 0x201e0000 20000
u-boot => bootaux 0x1ffe0000
```

Then boot Linux kernel:

```
# For i.MX 93 EVK
u-boot => setenv fdtfile imx93-11x11-evk-virtio-net-cm33.dtb
# For i.MX 93 14x14 EVK
u-boot => setenv fdtfile imx93-14x14-evk-virtio-net-cm33.dtb

u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

b. Run the backend on Cortex-A core
Execute the following command in the U-Boot command line:

```
u-boot => ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-
multicore/virtio-net-backend-freertos/virtio_net_backend_ca55.bin
u-boot => dcache flush && icache flush
u-boot => cpu 1 release 0xd0000000
```

Then boot Linux kernel:

```
# For i.MX 93 EVK
```

```
u-boot => setenv fdtfile imx93-11x11-evk-virtio-net-ca55.dtb
# For i.MX 93 14x14 EVK
u-boot => setenv fdtfile imx93-14x14-evk-virtio-net-ca55.dtb

u-boot => setenv mmcargs $mmcargs maxcpus=1 clk_ignore_unused
u-boot => run bsp_bootcmd
```

– **Or start Linux, then use remoteproc to start RTOS**

a. Run the backend on Cortex-M core
Boot up Linux:

```
# For i.MX 93 EVK
u-boot => setenv fdtfile imx93-11x11-evk-virtio-net-cm33.dtb
# For i.MX 93 14x14 EVK
u-boot => setenv fdtfile imx93-14x14-evk-virtio-net-cm33.dtb

u-boot => run prepare_mcore; mw.l 0xa8400000 0 1
u-boot => setenv mmcargs $mmcargs clk_ignore_unused
u-boot => run bsp_bootcmd
```

After Linux boot up, start backend RTOS running on Cortex-M Core by using remoteproc under Linux:

```
root@imx93evk:/sys# modprobe -r virtio_net
root@imx93evk:/sys# modprobe -r virtio_mmio
root@imx93evk:/sys# echo /examples/heterogeneous-multicore/virtio-net-
backend-freertos/virtio_net_backend_cm33.elf  > /sys/devices/platform/
remoteproc-cm33/remoteproc/remoteproc0/firmware
root@imx93evk:/sys# echo start > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc0/state
root@imx93evk:/sys# modprobe virtio_mmio
root@imx93evk:/sys# modprobe virtio_net
```

b. Run the backend on Cortex-A core
Boot up Linux:

```
# For i.MX 93 EVK
u-boot => setenv fdtfile imx93-11x11-evk-virtio-net-ca55.dtb
# For i.MX 93 14x14 EVK
u-boot => setenv fdtfile imx93-14x14-evk-virtio-net-ca55.dtb

u-boot => setenv mmcargs $mmcargs maxcpus=1 clk_ignore_unused; mw.l
 0xfc700000 0 1
u-boot => run bsp_bootcmd
```

After Linux boot up, start backend RTOS running on Cortex-A Core by using remoteproc under Linux:

```
root@imx93evk:/sys# modprobe -r virtio_net
root@imx93evk:/sys# modprobe -r virtio_mmio
root@imx93evk:/sys# echo /examples/heterogeneous-multicore/virtio-net-
backend-freertos/virtio_net_backend_ca55.elf  > /sys/devices/platform/
remoteproc-ca55-1/remoteproc/remoteproc0/firmware
root@imx93evk:/sys# echo start > /sys/devices/platform/remoteproc-
ca55-1/remoteproc/remoteproc0/state
root@imx93evk:/sys# modprobe virtio_mmio
root@imx93evk:/sys# modprobe virtio_net
```

4. **Evaluate Networking Sharing**
After backend starts, the second UART console displays the following backend log:

```
Starting Virtio networking backend...
```

```
virtio network device initialization succeed!
Switch enabled with enet remote port succeed!
ENET: PHY link is up with speed 1000M full-duplex
```

After the kernel boots up, use “`ifconfig`” and “`ping`” commands to verify the virtual networking interface. In the following log, "`eth0`" is virtio_net interface, but it may be different on different platform. So, use "ethtool" to find out the virtio_net interface that is using "virtio_net" driver, and the default MAC address of virtio_net interface is "00:04:9f:00:01:02".

```
root@imx8mm-lpddr4-evk:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.107  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fd08:d7d5:e652::733  prefixlen 128  scopeid 0x0<global>
        inet6 fd08:d7d5:e652:0:201:2ff:fe03:405  prefixlen 64  scopeid
 0x0<global>
        inet6 fe80::201:2ff:fe03:405  prefixlen 64  scopeid 0x20<link>
        ether 00:04:9f:00:01:02  txqueuelen 1000  (Ethernet)
        RX packets 54  bytes 5544 (5.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 143  bytes 20887 (20.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@imx8mm-lpddr4-evk:~# ethtool -i eth0
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: b8400000.virtio_net
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no

root@imx8mm-lpddr4-evk:~# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.888 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.541 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=2.13 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=2.29 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=1.73 ms
```

Use the following command to change the MAC address of `virtio_net`:

```
root@imx8mm-lpddr4-evk:~# ifconfig eth0 hw ether 00:04:9f:00:01:03
```

Attention: *While running M-core backend on i.MX 93 EVK and i.MX 93 14x14 EVK platforms, changing the MAC address causes the M-core to be stuck.*

3.7 NETC PSI-VSI networking sharing

3.7.1 Overview

The NETC presents itself as a multi-function PCIe Root Complex Integrated Endpoint (RCiEP), and the NIC functionality in NETC is known as EtherNET Controller (ENETC). ENETC supports virtualization/isolation based on PCIe Single Root IO Virtualization (SR-IOV). This example lets RTOS own the PF (PSI) and PCIe configuration space and Linux own the VFs (VSIs).

Figure 47 shows the NETC networking sharing architecture.

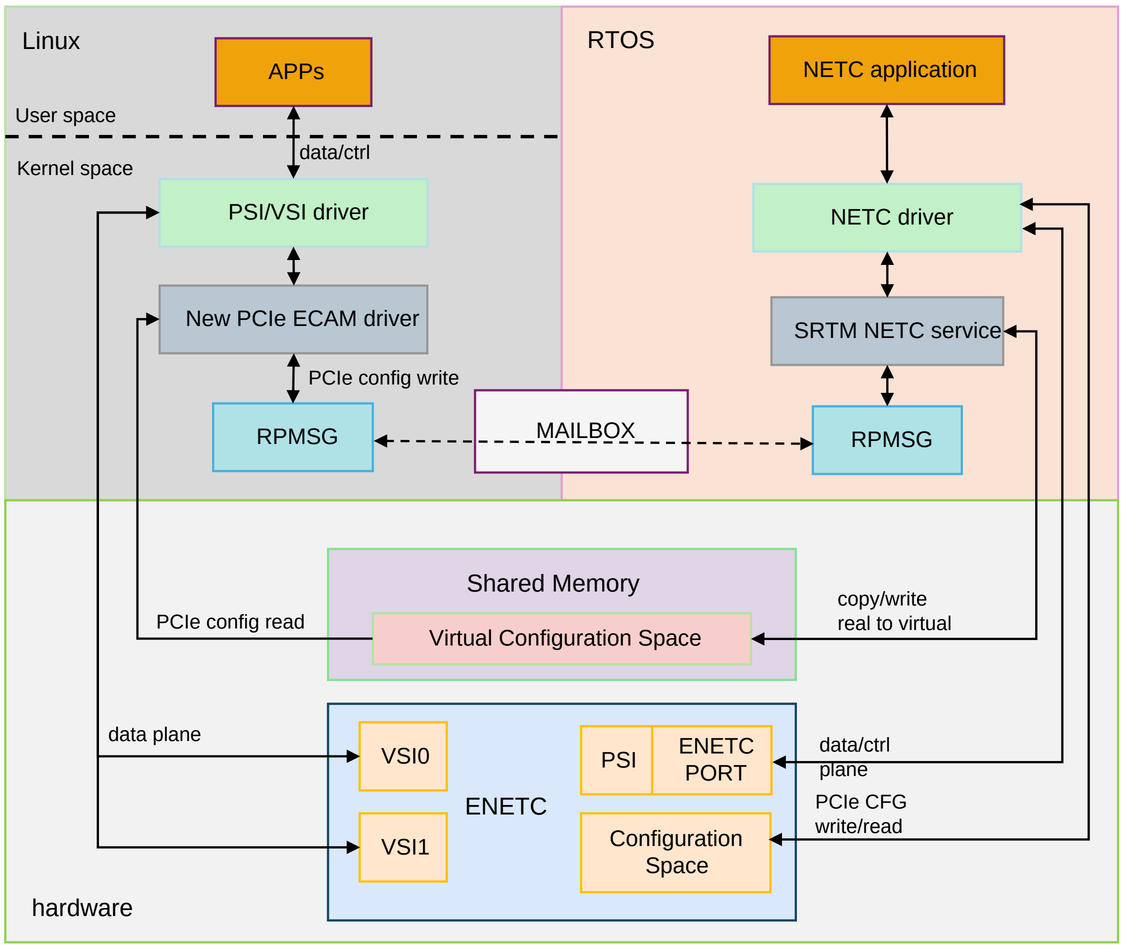


Figure 47. NETC PSI/VSI networking sharing architecture

- **Hardware Resource Allocation**
  - NETC Resource Allocation
    1. RTOS: Owns PSI (PF), other NETC functions, and real PCIe Configuration Space.
    2. Linux: Owns VSI0 and VSI1 (two VFs).
  - Shared Memory Resource
    Virtual Configuration Space: Located in shared memory between Linux and RTOS, preinitialized by RTOS.
- **Software Components**
  - Linux:
    1. New PCIe ECAM Driver: Used to enumerate the proxy ENETC PF device, it redirects PCIe config write requests to RPMSG requests; and PCIe config read operations directly read the Virtual Configuration Space.
    2. RPMSG client: Provide data channel for PCIe write requests to send to RTOS side SRTM service.
    3. ENETC PSI driver: Re-use the exist ENETC PF driver with update for support proxy PF device, which is only used to enable the PCIe SR-IOV VFs.
    4. ENETC VSI driver: Re-use the exist ENETC VF driver.
  - RTOS:
    1. NETC Driver: Control all the NETC hardware resources except the VSIs owned by Linux.
    2. SRTM NETC Service: Process Linux config write requests, deliver the requests to the NETC driver to write into the real PCIe Configuration Space and then update Virtual Configuration Space as well.

In the ENETC, the data plane of VSI does not introduce any extra data-copy, so the shared VSI NIC under Linux can get the same performance and the native ENETC NIC. At the same time, the control plane are still through the RTOS NETC driver. This architecture leverages Virtual Config Space + RPMSG/SRTM layering to allow Linux to reuse the generic PCI stack while ensuring the NETC remains securely controlled by RTOS.

3.7.2 Running NETC PSI-VSI networking sharing on i.MX 95 19x19 EVK

On i.MX 95 EVK, it is sharing the ENETC2 between Cortex-M7 RTOS, which owns the PSI, and Cortex-A55 Linux, which owns the 2 VSIs.

- **Hardware setup**
  1. Set up the UART console.
     Refer to the steps listed in *Setup UART console* in the Section 3.8.1.
  2. Connect the ENETC 10G port (J28) to a ethernet switch/router.
  3. Program the SD card with the dedicated boot image:

     ```
     $ dd if=nxp-image-real-time-edge-imx95-19x19-lpddr5-evk.rootfs.wic of=/dev/
     sd<x> bs=1M conv=fsync
     $ dd if=imx-boot-variant-netc-imx95-19x19-lpddr5-evk-sd.bin-flash_netc of=/
     dev/sd<x> bs=1k seek=32 conv=fsync
     ```
  4. Install the SD card and power on the board.
- **Running the NETC networking sharing**
  1. After the board is power up, the FreeRTOS will be started automatically on Cortex-M7.

     ```
     Initializing PHY...
     [LINK STATE] netif=0, state=up, speed=1000M_full

     ************************************************
      DHCP example
     ************************************************
      DHCP state       : SELECTING
      DHCP state       : REQUESTING
      DHCP state       : CHECKING
      DHCP state       : BOUND

      IPv4 Address     : 192.168.0.4
      IPv4 Subnet mask : 255.255.255.0
      IPv4 Gateway     : 192.168.0.1
     ```
  2. Boot up Linux with the dedicated device tree:

     ```
     u-boot=> setenv fdtfile imx95-19x19-evk-netc-rpmsg.dtb
     u-boot=> run bsp_bootcmd
     ```
  3. Enable the VSIs of ENETC2 on the Linux prompt:
     a. Confirm the proxy PF driver at the Linux side. This proxy driver only provides the SR-IOV interfaces to enable/disable VFs.

     ```
     root@imx95-19x19-lpddr5-evk:~# lspci
     0002:00:10.0 Ethernet controller: Philips Semiconductors Device 080b (rev
      04)
     ```
     b. Enable the VSIs:

     ```
     root@imx95-19x19-lpddr5-evk:~# echo 1 > /sys/bus/pci/
     devices/0002\:00\:10.0/sriov_numvfs  # Enable one of the VSIs
     root@imx95-19x19-lpddr5-evk:~# echo 2 > /sys/bus/pci/
     devices/0002\:00\:10.0/sriov_numvfs  # Enable all the 2 VSIs
     ```

4. Check the ethernet interfaces of the enabled VSI(s).

```
root@imx95-19x19-lpddr5-evk:~# ifconfig
eth0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet 192.168.0.7  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::200:faff:fefa:dda0  prefixlen 64  scopeid 0x20<link>
        ether 00:00:fa:fa:dd:a0  txqueuelen 1000  (Ethernet)
        RX packets 60  bytes 8953 (8.7 KiB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 62  bytes 9626 (9.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet 192.168.0.9  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::200:faff:fefa:dda1  prefixlen 64  scopeid 0x20<link>
        ether 00:00:fa:fa:dd:a1  txqueuelen 1000  (Ethernet)
        RX packets 53  bytes 7885 (7.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 56  bytes 10438 (10.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

5. Disable the VSIs:

```
root@imx95-19x19-lpddr5-evk:~# echo 0 > /sys/bus/pci/devices/0002\:00\:10.0/
sriov_numvfs
```

3.8 Unified Life Cycle Management

Heterogeneous Multicore Framework provides Unified Life Cycle Management for both the Cortex-A and the Cortex-M cores.

Real-time Edge supports bootstrapping the **native Zephyr** and **native FreeRTOS** on the **Cortex-A core** and **Cortex-M core** with **U-Boot** command and with the **RemoteProc** under Linux as listed in the below table.

Table 41. Bootstrapping options for Cortex-A and CortexM cores

Core Type	U-Boot	RemoteProc on Linux
NativeZephyr on **M** core	Y	Y
NativeFreeRTOS on **M** core	Y	Y
NativeZephyr on **A** core	Y	Y
NativeFreeRTOS on **A** core	Y	Y

3.8.1 Using U-Boot commands to manage the life cycle of RTOS on the Cortex-A core

Multiple U-Boot commands can be used to start or stop the Cortex-A core RTOS.

- "`cpu`" command
  The following is the help information of the U-Boot `cpu` command:

```
u-boot=> cpu
cpu - Multiprocessor CPU boot manipulation and release

Usage:
cpu <num> reset                 - Reset cpu <num>
cpu status                      - Status of all cpus
cpu <num> status                - Status of cpu <num>
cpu <num> disable               - Disable cpu <num>
cpu <num> release <addr> [args] - Release cpu <num> at <addr> with [args]
```

Except `cpu <num> reset` command, all other cpu commands have been implemented on i.MX 8M Mini, i.MX 8M Plus, i.MX 93, and i.MX 95 platforms.
In general, U-Boot runs on the primary core (Core0) of Cortex-A Cores. Therefore, users have the following options:

- Use the command `cpu <num> release <addr> [args]` to start RTOS on any other secondary Cortex-A Cores (Cores except Core0) from a specified memory address. For this, you must load the RTOS binary images into the corresponding memory space.
- Use the `cpu <num> disable` command to power off any secondary core that runs RTOS.
- Use the `cpu status` or the `cpu <num> status` commands to check the status (running or power off) for all or a specified Cortex-A Core.

- `go` command
  Following is the help information of the U-Boot `go` command:

```
u-boot=> go
go - start application at address 'addr'

Usage:
go addr [arg ...]
    - start application at address 'addr'
      passing 'arg' as arguments
```

  The `go` command can be used to start RTOS on the primary core (Core0) but it does not return to the U-Boot command line.

In summary, use the U-Boot `go` command to boot the RTOS from Core0. Use the U-Boot command `cpu` to boot or to power off the RTOS running on other Cortex-A Cores except the Core0.

The below example shows how to run the `hello_world` example on the following boards:

- i.MX 8M Mini EVK
- i.MX 8M Plus EVK
- i.MX 91 EVK and i.MX 91 QSB
- i.MX 93 11x11 EVK and i.MX 93 14x14 EVK
- i.MX 943 15x15 and 19x19 EVK
- i.MX 95 15x15 EVK and i.MX 95 19x19 EVK

1. **Set up the UART console for Native RTOS**
   Connect DEBUG UART slot on the board to your PC through the USB Cable. This step creates some USB serial ports on the PC:
   - i.MX 8M Mini EVK (port 0~1): port1 is for U-Boot/Linux or RTOS; port0 for RTOS.
   - i.MX 8M Plus EVK (port 0~3): port2 is for U-Boot/Linux or RTOS; port3 for RTOS.
   - i.MX 91 EVK and QSB (port 0~3): port2 is for U-Boot/Linux or RTOS.
   - i.MX 93 11x11 and 14x14 EVK (port 0~3): port2 is for U-Boot/Linux or RTOS; port3 for RTOS.
   - i.MX 943 15x15 and 19x19 EVK (port 0~3): port2 is for U-Boot/Linux or RTOS; port0 for RTOS.
     Note: *This platform uses multiplexing between JTAG and UART8. Therefore, run the below '`bcu`' command on the PC to select UART8:*

```
$ bcu lsftdi
$ bcu set_gpio fta_jtag_host_en 0 -board=imx943evk19b1 -id=<location_id>
```

   - i.MX95 15x15 and 19x19 EVK (port 0~3): port2 is for U-Boot/Linux or RTOS; port0 for RTOS.

   Open the UART consoles needed with the following setup:
   - 115200
   - No parity

- 8 data bits
- 1 stop bit

2. **Booting Native FreeRTOS Image**
   After powering up the board and entering U-Boot command line, execute the following U-Boot commands to run the `hello_world` example.
   - For i.MX 8M Mini EVK
     Using the commands below to start RTOS from Core3 (the first Core is Core0):

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca53_RTOS0_UART4.bin
=> dcache flush; icache flush
=> cpu 3 release 0x93C00000
```

After the preceding steps are followed, the UART4 console displays the following RTOS log:

```
Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
```

Use the following command to power off the RTOS running on Core3 (the first Core is Core0):

```
u-boot=> cpu 3 disable
```

   - For i.MX 8M Plus EVK
     Using the commands below to start RTOS from Core3 (the first Core is Core0):

```
=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca53_RTOS0_UART4.bin
=> dcache flush; icache flush;
=> cpu 3 release 0xC0000000
```

After the preceding steps are followed, the UART4 console displays the following RTOS log:

```
Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MP-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
```

Use the following command to power off the RTOS running on Core3 (the first Core is Core0):

```
u-boot=> cpu 3 disable
```

   - For i.MX 91 EVK and QSB board:
     Using the commands below to start RTOS:

```
=> ext4load mmc 1:2 0x80000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca55_RTOS0_UART1.bin
=> dcache flush; icache flush;
=> go 0x80000000
```

After the preceding steps are followed, the UART1 console displays the following RTOS log:

```
Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX91-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core0 (MPID: 0x0)
```

   - For i.MX 93 11x11 and 14x14 EVK board:

Using the commands below to start RTOS from Core1 (the first Core is Core0):

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca55_RTOS0_UART2.bin
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

After the preceding steps are followed, the UART2 console displays the following RTOS log:

```
Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX93-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core1 (MPID: 0x100)
```

Use the following command to poweroff the RTOS running on Core1 (the first Core is Core0):

```
u-boot=> cpu 1 disable
```

- For i.MX943 15x15 and 19x19 EVK board:
  Using the commands below to start RTOS from Core3 (the first Core is Core0):

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca55_RTOS0_UART8.bin
=> dcache flush; icache flush;
=> cpu 3 release 0xD0000000
```

After the preceding steps are followed, the UART8 console displays the following RTOS log:

```
Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX943-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 4 times from Cortex-A55 core3 (MPID: 0x300)
FreeRTOS_thread_0: hello 5 times from Cortex-A55 core3 (MPID: 0x300)
```

- For i.MX95 15x15 and 19x19 EVK board:
  Using the commands below to start RTOS from Core5 (the first Core is Core0):

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca55_RTOS0_UART3.bin
=> dcache flush; icache flush;
=> cpu 5 release 0xD0000000
```

After the preceding steps are followed, the UART3 console displays the following RTOS log:

```
Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX95-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core5 (MPID: 0x500)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core5 (MPID: 0x500)
```

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

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca53_RTOS0_UART4.bin
=> dcache flush; icache flush;
=> go 0x93C00000
```

After the preceding steps are followed, the UART4 console displays the following RTOS log:

```
Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core0 (MPID: 0x0)
```

3. **Booting Native Zephyr Image**
   After powering up the board and entering U-Boot command line, execute the following U-Boot commands to run the `hello_world` example.
   - For i.MX 8M Mini EVK
     Using below commands to boot `hello_world` example on the Cortex-A Core3 (the first Core is Core0):

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca53_RTOS0_UART4.bin
=> dcache flush; icache flush
=> cpu 3 release 0x93C00000
```

After the preceding steps are followed, the UART4 console displays the following RTOS log:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mm_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
```

Use the following command to power off the RTOS running on Core3 (the first Core is Core0):

```
u-boot=> cpu 3 disable
```

   - For i.MX 8M Plus EVK
     Using below commands to boot `hello_world` example on the Cortex-A Core3 (the first Core is Core0):

```
=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca53_RTOS0_UART4.bin
=> dcache flush; icache flush;
=> cpu 3 release 0xC0000000
```

After the preceding steps are followed, the UART4 console displays the following RTOS log:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mp_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
Zephyr_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
```

Use the following command to poweroff the RTOS running on Core3 (the first Core is Core0):

```
u-boot=> cpu 3 disable
```

   - For i.MX 91 EVK and QSB board:
     Using the commands below to start RTOS:

```
=> ext4load mmc 1:2 0x80000000 /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca55_RTOS0_UART1.bin
=> dcache flush; icache flush;
=> go 0x80000000
```

After the preceding steps are followed, the UART1 console displays the following RTOS log:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
```

```
Cortex-A55: RTOS0: Hello World! Real-time Edge on imx91_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core0 (MPID: 0x0)
Zephyr_thread_0: hello 1 times from Cortex-A55 core0 (MPID: 0x0)
Zephyr_thread_0: hello 2 times from Cortex-A55 core0 (MPID: 0x0)
Zephyr_thread_0: hello 3 times from Cortex-A55 core0 (MPID: 0x0)
```

- For i.MX 93 11x11 and 14x14 EVK board:
  Using the commands below to start RTOS from Core1 (the first Core is Core0):

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca55_RTOS0_UART2.bin
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

After the preceding steps are followed, the UART2 console displays the following RTOS log:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A55: RTOS0: Hello World! Real-time Edge on imx93_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core1 (MPID: 0x100)
Zephyr_thread_0: hello 1 times from Cortex-A55 core1 (MPID: 0x100)
Zephyr_thread_0: hello 2 times from Cortex-A55 core1 (MPID: 0x100)
Zephyr_thread_0: hello 3 times from Cortex-A55 core1 (MPID: 0x100)
```

Use the following command to poweroff the RTOS running on Core1 (the first Core is Core0):

```
u-boot=> cpu 1 disable
```

- For i.MX943 15x15 and 19x19 EVK board:
  Using the commands below to start RTOS from Core3 (the first Core is Core0):

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca55_RTOS0_UART8.bin
=> dcache flush; icache flush;
=> cpu 3 release 0xD0000000
```

After the preceding steps are followed, the UART8 console displays the following RTOS log:

```
*** Booting Zephyr OS build v4.2.0-18435-g97b8474ac1a0 ***
Cortex-A55: RTOS0: Hello World! Real-time Edge on imx943_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 1 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 2 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 3 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 4 times from Cortex-A55 core3 (MPID: 0x300)
Zephyr_thread_0: hello 5 times from Cortex-A55 core3 (MPID: 0x300)
```

- For i.MX95 15x15 and 19x19 EVK board:
  Using the commands below to start RTOS from Core5 (the first Core is Core0):

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca55_RTOS0_UART3.bin
=> dcache flush; icache flush;
=> cpu 5 release 0xD0000000
```

After the preceding steps are followed, the UART3 console displays the following RTOS log:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A55: RTOS0: Hello World! Real-time Edge on imx95_evk
Zephyr_thread_0: hello 0 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 1 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 2 times from Cortex-A55 core5 (MPID: 0x500)
Zephyr_thread_0: hello 3 times from Cortex-A55 core5 (MPID: 0x500)
```

- go command example

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

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/hello-world-
zephyr/hello_world_ca53_RTOS0_UART4.bin
=> dcache flush; icache flush
=> go 0x93C00000
```

After the preceding steps are followed, the UART4 console displays the following RTOS log:

```
*** Booting Zephyr OS build v4.2.0-18438-gf719c4b2d3d7 ***
Cortex-A53: RTOS0: Hello World! Real-time Edge on imx8mm_evk
Zephyr_thread_0: hello 0 times from Cortex-A53 core0 (MPID: 0x0)
Zephyr_thread_0: hello 1 times from Cortex-A53 core0 (MPID: 0x0)
Zephyr_thread_0: hello 2 times from Cortex-A53 core0 (MPID: 0x0)
Zephyr_thread_0: hello 3 times from Cortex-A53 core0 (MPID: 0x0)
```

3.8.2 Using RemoteProc under Linux to Manage Life Cycle of RTOS on Cortex-A Core

Apart from the U-Boot commands in U-Boot, another Cortex-A Core RTOS (native FreeRTOS and native Zephyr) management tool is remoteproc under Linux. Using remoteproc can dynamically start or stop native FreeRTOS or native Zephyr under Linux. This Cortex-A Core RTOS life cycle management method has limitations. The limitation is that Linux must run on at least a single Cortex-A Core to manage RTOS running on all the other Cortex-A Cores.

Remoteproc (Remote Processor Framework) under Linux is used to control remote processors on which different instances of operating systems are running. For example, it can be used to power on, load firmware, or to power off the remote processors. For SMP Linux, which it runs on multiple Cortex-A Cores, each Cortex-A Core can be used as a remote processor, and can use remoteproc to bring up or bring down another RTOS instance on it. To run another RTOS on it, remoteproc first removes this Cortex-A Core from SMP Linux by using CPU hotplug, and brings it to be down. Then, remoteproc brings it up again with a new RTOS instance to run it. Remoteproc can also bring this CPU Core running RTOS to be down and then plug it back to SMP Linux by using CPU hotplug.

- **Linux Device Tree Configuration**

To use remoteproc to manage life cycle on Cortex-A Core, you must add device nodes in Linux dts. For example, the following dts nodes are defined in `imx8m-rproc-ca53.dtsi` which defines four remoteproc instances. In each instance `fsl,cpus-bits` defines CPU Core bitmask to specify which CPU Core(s) can be managed by this instance. The `memory-region` is used to specify the reserved memory space used by RTOS to be run on this instance. So, the first instance in the following example runs on Core1, the second one runs on Core2, the third one runs on Core3. The fourth one runs on Core2 and Core3 and it can be used to run SMP RTOS. However, since the CPU Core used by the fourth one is duplicated with the second one and the third one, so we cannot run RTOS simultaneously on the second one or third one with the fourth one.

```
         ca53_1: remoteproc-ca53-1 {
                compatible = "fsl,imx-rproc-psci";
                /* bitmask:0b0010, assign A53 Core 1 */
                fsl,cpus-bits = <0x2>;
                memory-region = <&rtos_ca53_reserved>;
        };

        ca53_2: remoteproc-ca53-2 {
                compatible = "fsl,imx-rproc-psci";
                /* bitmask:0b0100, assign A53 Core 2 */
                fsl,cpus-bits = <0x4>;
                memory-region = <&rtos_ca53_reserved>;
        };

        ca53_3: remoteproc-ca53-3 {
                compatible = "fsl,imx-rproc-psci";
                /* bitmask:0b1000, assign A53 Core 3 */
                fsl,cpus-bits = <0x8>;
                memory-region = <&rtos_ca53_reserved>;
        };

        ca53_2_3: remoteproc-ca53-2-3 {
                compatible = "fsl,imx-rproc-psci";
                /* bitmask:0b1100, assign A53 Core 2 and Core 3 */
                fsl,cpus-bits = <0xc>;
                memory-region = <&rtos_ca53_reserved>;
        };
```

- **Hardware Resource Allocation between RTOS and Linux**

To run a flexible AMP system with multiple RTOSes and Linux running together on the single SoC simultaneously, the hardware resources must be allocated to different OSes carefully. This allocation is necessary to avoid conflicts and the hardware resources include memory and peripherals. Refer to `imx8mp-evk-multicore-rtos.dts` in Real-time Edge Linux as an example. The memory used by RTOS must be added to the `reserved-memory` in Linux dts. In addition, the Linux dts node 'UART4' must be disabled as it is used as the RTOS debug console.

- **Using remoteproc to start or stop RTOS on Cortex-A Core**
  When running multiple OSes on different Cortex-A Cores on one platform, you must avoid GIC to be re-configured to crash the OS that has already been started. For Zephyr, you must enable "`CONFIG_GIC_SAFE_CONFIG=y`". For the Linux kernel, you must enable "`CONFIG_GIC_GENTLE_CONFIG=y`", and for FreeRTOS in Real-time Edge, a similar feature has already enabled by default.

Take the Heterogeneous Multicore FreeRTOS application `hello_world` as example, follow the following steps to start or stop the RTOS by using remoteproc.

1. **Setup UART console for Native RTOS**
   Refer to the steps described in **Setup UART console** in the Section Section 3.8.1.
2. **Starting the Linux kernel**
   To run RTOS, a device tree that enables remoteproc and has compatible resource allocation between RTOS and Linux must be used.
   To do this, when U-Boot is executing, stop at the U-Boot prompt with a terminal emulator connected to the serial port and execute the following commands (based on the board and the application):
   - For i.MX 8M Mini EVK

   ```
   => setenv fdtfile imx8mm-evk-multicore-rtos.dtb
   => setenv mmcargs $mmcargs clk_ignore_unused
   => run bsp_bootcmd
   ```

   - For i.MX 8M Plus EVK

   ```
   => setenv fdtfile imx8mp-evk-multicore-rtos.dtb
   => setenv mmcargs $mmcargs clk_ignore_unused
   => run bsp_bootcmd
   ```

   - For i.MX 93 11x11 and 14x14 EVK

   ```
   # For i.MX 93 11x11 EVK
   => setenv fdtfile imx93-11x11-evk-multicore-rtos.dtb
   # For i.MX 93 14x14 EVK
   => setenv fdtfile imx93-14x14-evk-multicore-rtos.dtb
   ```

```
=> setenv mmcargs $mmcargs clk_ignore_unused
=> run bsp_bootcmd
```

- For i.MX 95 15x15 19x19 EVK

```
# For i.MX 95 15x15 EVK
=> setenv fdtfile imx95-15x15-evk-multicore-rtos.dtb
# For i.MX 95 19x19 EVK
=> setenv fdtfile imx95-19x19-evk-multicore-rtos.dtb
=> setenv mmcargs $mmcargs clk_ignore_unused
=> run bsp_bootcmd
```

3. **Using remoteproc**
   When Linux boots up, log in Linux.
   The `hello_world` application is located at the following directory of the root file system:

```
/examples/heterogeneous-multicore/hello-world-freertos #FreeRTOS Images
/examples/heterogeneous-multicore/hello-world-zephyr #Zephyr Images
```

   The remoteproc sys portal is available in the following directories:
   - For i.MX 8M Mini EVK and i.MX 8M Plus EVK:

```
/sys/devices/platform/remoteproc-ca53-1/remoteproc/remoteproc0
/sys/devices/platform/remoteproc-ca53-2/remoteproc/remoteproc1
/sys/devices/platform/remoteproc-ca53-3/remoteproc/remoteproc2
/sys/devices/platform/remoteproc-ca53-2-3/remoteproc/remoteproc3
```

     These four remoteproc portals correspond to the remoteproc instances defined in the Linux device tree (`imx8m-rproc-ca53.dtsi`).
   - For i.MX 93 11x11 and 14x14 EVK:

```
/sys/devices/platform/remoteproc-ca55-1/remoteproc/remoteproc0
```

     This remoteproc portal corresponds to the remoteproc instance defined in the Linux device tree (`imx93-rproc-ca55.dtsi`)
   - For i.MX 95 (15x15 and 19x19) EVK:

```
/sys/devices/platform/remoteproc-ca55-1/remoteproc/remoteproc0
/sys/devices/platform/remoteproc-ca55-2/remoteproc/remoteproc1
/sys/devices/platform/remoteproc-ca55-3/remoteproc/remoteproc2
/sys/devices/platform/remoteproc-ca55-4/remoteproc/remoteproc3
/sys/devices/platform/remoteproc-ca55-5/remoteproc/remoteproc4
/sys/devices/platform/remoteproc-ca55-4-5/remoteproc/remoteproc5
```

     This remoteproc portal is the corresponding remoteproc instance defined in the Linux device tree (`imx95-rproc-ca55.dtsi`)

   a. **Start RTOS**
      To start RTOS on a specified CPU Core, first you must specify the elf RTOS image by echoing the image path and name. If the RTOS image is located in the `/lib/firmware/` directory, only specify the image name without the image path to the corresponding remoteproc instance's `firmware` portal. Then, echo `start` to the corresponding remoteproc instance's `state` portal.
      For example, use the following commands to start FreeRTOS `hello_world` application on Cortex-A Core3 (the first Core is Core0) on i.MX 8M Mini EVK and i.MX 8M Plus EVK boards:

```
root@imx8mp-lpddr4-evk:~# echo /examples/heterogeneous-multicore/hello-world-freertos/
hello_world_ca53_RTOS0_UART4.elf > /sys/devices/platform/remoteproc-ca53-3/remoteproc/
remoteproc2/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-ca53-3/remoteproc/
remoteproc2/state
```

Then, the following RTOS log displays on the RTOS UART console on i.MX 8M Plus EVK. (A similar log is obtained for the i.MX 8M Mini EVK board):

```
Cortex-A53: RTOS0: Hello world! Real-time Edge on MIMX8MP-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 1 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 2 times from Cortex-A53 core3 (MPID: 0x3)
FreeRTOS_thread_0: hello 3 times from Cortex-A53 core3 (MPID: 0x3)
```

Or Use the following commands to start FreeRTOS `hello_world` application on Cortex-A Core1 (first Core is Core0) on i.MX 93 11x11 and 14x14 EVK:

```
root@imx93evk:~# echo /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_ca55_RTOS0_UART2.elf > /sys/devices/platform/
remoteproc-ca55-1/remoteproc/remoteproc0/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

Then, the following RTOS log is displayed on the RTOS UART console on i.MX 93 11x11 and 14x14 EVK:

```
Cortex-A55: RTOS0: Hello world! Real-time Edge on MIMX93-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 1 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 2 times from Cortex-A55 core1 (MPID: 0x100)
FreeRTOS_thread_0: hello 3 times from Cortex-A55 core1 (MPID: 0x100)
```

b. **Stop RTOS**
To stop RTOS on a specified CPU core, echo `stop` to the corresponding `state` portal of the remoteproc instance.
For example, use the following command to stop RTOS running on Cortex-A Core3 (first Core is Core0) on i.MX 8M Mini EVK and i.MX 8M Plus EVK:

```
root@imx8mp-lpddr4-evk:~# echo stop > /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

Or use the following command to stop RTOS running on Cortex-A Core1 (the first Core is Core0) on i.MX 93 11x11 and 14x14 EVK:

```
root@imx93evk:~# echo stop > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

- **Cortex-A Core dynamically allocated between Linux and RTOS**
Before running RTOS by using remoteproc, SMP Linux kernel uses all Cortex-A Cores by default. After running RTOS on the specified Cortex-A Core by using remoteproc, these Cortex-A core(s) are hot removed from SMP Linux kernel and RTOS runs on them. After using the `remoteproc stop` command to stop the RTOS running on Cortex-A Core, these CPU Cores once used by RTOS are hot added back to SMP Linux kernel.
Can use the following command to check which CPU Core are used by SMP Linux kernel currently:

```
root@imx93evk:~# cat /proc/cpuinfo
```

Note: *The remoteproc portal instance may change according to the different device tree configuration used. Therefore, use the `name` under each remoteproc instance portal to check which instance must be used.*

3.8.3 Using U-Boot Commands to Manage Life Cycle of RTOS on Cortex-M Core

U-boot command "`bootaux`" is used to boot Cortex-M Core RTOS Image from U-Boot, for example, after the board is booted into the U-Boot console.

- Use the following command to boot Arm Cortex-M core on i.MX 8M Mini LPDDR4 EVK board, it uses UART4:

```
=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm4.bin; cp.b 0x48000000 0x7e0000 20000;
=> bootaux 0x7e0000
```

- Use the following command on i.MX 8M Plus LPDDR4 EVK board, it uses UART4:

```
=> ext4load mmc 1:2 0x48000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm7.bin; cp.b 0x48000000 0x7e0000 20000;
=> bootaux 0x7e0000
```

- Use the following command on i.MX 93 EVK board, it uses UART2:

```
=> ext4load mmc 1:2 0xd0000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm33.bin; cp.b 0xd0000000 0x201e0000 20000;
=> bootaux 0x1ffe0000
```

- Use the following command on i.MX 95 15x15 EVK and i.MX95 19x19 EVK board, it uses UART3:

```
=> ext4load mmc 1:2 0x90000000 /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm7.bin;
=> cp.b 0x90000000 0x203c0000 ${filesize}
=> bootaux 0 1
```

- Then, the below RTOS log is displayed on the UART console (taking i.MX 8M Mini EVK as an example)

```
Cortex-M4: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M4 core0 (MPID: 0x0)
```

3.8.4 Using RemoteProc under Linux to Manage Life Cycle of RTOS on Cortex-M Core

Use Linux remoteproc to start and stop RTOS on Cortex-M Core, taking the Heterogeneous Multicore `hello_world` application as an example:

- Booting Arm Cortex-M core on i.MX 8M Mini EVK, i.MX 8M Plus EVK, i.MX 93 EVK, i.MX 95 15x15 EVK and i.MX 95 19x19 EVK boards:
  1. Starting Linux kernel:

     ```
     => run prepare_mcore
     => setenv fdtfile imx8mm-evk-multicore-rtos.dtb # for i.MX 8M Mini EVK
     => setenv fdtfile imx8mp-evk-multicore-rtos.dtb # for i.MX 8M Plus EVK
     => setenv fdtfile imx93-11x11-evk-multicore-rtos.dtb # for i.MX 93 EVK
     => setenv fdtfile imx95-15x15-evk-multicore-rtos.dtb; # for i.MX 95 15x15
      EVK
     => setenv fdtfile imx95-19x19-evk-multicore-rtos.dtb; # for i.MX 95 19x19
      EVK
     => boot
     ```

  2. After Linux booting up, boot up Cortex-M core RTOS application:

     ```
     # i.MX 8M Mini EVK uses UART4
     root@imx8mm-lpddr4-evk:~# echo -n /examples/heterogeneous-multicore/hello-
     world-freertos/hello_world_cm4.elf > /sys/devices/platform/imx8mm-cm4/
     remoteproc/remoteproc4/firmware
     ```

```
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/state

# i.MX 8M Plus EVK uses UART4
root@imx8mp-lpddr4-evk:~# echo -n /examples/heterogeneous-multicore/hello-
world-freertos/hello_world_cm7.elf > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/state

# i.MX 93 EVK uses UART2
root@imx93evk:~# echo -n /examples/heterogeneous-multicore/hello-world-
freertos/hello_world_cm33.elf > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/state

# i.MX 95 15x15 EVK and i.MX 95 19x19 EVK use UART3
root@imx95-19x19-lpddr5-evk:~# echo /examples/heterogeneous-multicore/
hello-world-freertos/hello_world_cm7.elf > /sys/devices/platform/imx95-cm7/
remoteproc/remoteproc7/firmware
root@imx95-19x19-lpddr5-evk:~# echo start > /sys/devices/platform/imx95-cm7/
remoteproc/remoteproc7/state
```

Then, the below RTOS log is displayed on the UART console (taking i.MX 8M Mini EVK as an example):

```
Cortex-M4: RTOS0: Hello world! Real-time Edge on MIMX8MM-EVK
FreeRTOS_thread_0: hello 0 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 1 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 2 times from Cortex-M4 core0 (MPID: 0x0)
FreeRTOS_thread_0: hello 3 times from Cortex-M4 core0 (MPID: 0x0)
```

3. Then the Cortex-M core RTOS application can be shut down:

```
# i.MX 8M Mini EVK
root@imx8mm-lpddr4-evk:~# echo stop > /sys/devices/platform/imx8mm-cm4/
remoteproc/remoteproc4/state

# i.MX 8M Plus EVK
root@imx8mp-lpddr4-evk:~# echo stop > /sys/devices/platform/imx8mp-cm7/
remoteproc/remoteproc4/state

# i.MX 93 EVK
root@imx93evk:~# echo stop > /sys/devices/platform/remoteproc-cm33/
remoteproc/remoteproc1/state

# i.MX 95 15x15 EVK and i.MX 95 19x19 EVK
root@imx95-19x19-lpddr5-evk:~# echo stop > /sys/devices/platform/imx95-cm7/
remoteproc/remoteproc7/state
```

Note: *The remoteproc portal instance ID `/sys/devices/platform/remoteproc-cm7/remoteproc/remoteproc4` might be different, depending on the device tree configuration. Therefore, use the `name` under each remoteproc instance portal to check which instance must be used.*

3.9 Industrial applications

3.9.1 EtherCAT MainDevice stack SOEM

This section describes an overview of the SEOM application running on Cortex-A and Cortex-M cores.

3.9.1.1 Overview

The Heterogeneous Multicore framework provides a flexible mechanism to run Preemp-RT Linux or RTOS on different cores with Industrial applications. SOEM can run on FreeRTOS to scale down to the Cortex-M core and scale up to the Cortex-A core. Taking i.MX 8M Plus platform as an example, it is equipped with four Cortex-A53 cores and one Cortex-M7 core. Table 42 describes the use cases that can be executed on this MPU platform.

Table 42. SOEM use cases

SOEM examples	M7	A53	A53	A53	A53
digital_io	baremetal/FreeRTOS	FreeRTOS/Zephyr			
servo_motor	baremetal/FreeRTOS	FreeRTOS/Zephyr			
servo_motor_rt1180	baremetal/FreeRTOS	FreeRTOS/Zephyr			

The below platforms support SOEM:

- i.MX 8M Plus LPDDR4 EVK platform
- i.MX 8M Mini LPDDR4 EVK platform
- i.MX 93 EVK platform
- i.MX 95-15x15-lpddr4x EVK platform
- i.MX 95-19x19-lpddr5 EVK platform
- i.MX 943-19x19-lpddr4 EVK platform
- i.MX 943-19x19-lpddr5 EVK platform

Three examples are provided in this release:

- SOEM **`digital_io`** example for IO control using BECKHOFF EK1100 as EtherCAT SubDevice
- SOEM **`servo_motor`** example using Inovance SV680 servo as EtherCAT SubDevice
- SOEM **`servo_motor_rt1180`** example using **NXP XSERVO-MTR-PSG Board** as EtherCAT SubDevice
  For detailed information of SOEM, refer to Section 6.1.4.

3.9.1.2 Running SOEM Application on Cortex-A core

For detailed information, refer to Section 6.1.4 for steps to run the SOEM stack on the Cortex-A core for the supported platforms.

3.9.1.3 Running SOEM Application on Cortex-M core

For detailed information, refer to Section 6.1.4 to run the SOEM stack on the Cortex-M core for the supported platforms.

4 Heterogeneous multi-SoC framework

This section describes the features of the heterogeneous multi-SoC framework and how to implement it using NXP hardware platforms.

4.1 Introduction

Heterogeneous Multi-SoC Framework enables the usage of a combination of MPU and i.MX RT1180 as an Industrial Switch. This extends the MPU hardware capability with the i.MX RT1180 hardware capability, thereby providing switch functionality, TSN functionality, and the capability of supporting different Industrial Protocols. The i.MX RT1180 can be used to run real-time tasks such as industrial protocols in the real-time domain. On the other hand, the MPU can process compute-heavy tasks in the non-real-time domain.

The external Ethernet ports of i.MX RT1180 can be exposed to the MPU side as standard Ethernet interfaces as data path. Different interfaces such as SPI or I2C can be used as the management interface between MPU and i.MX RT1180.

4.2 Software architecture

The Linux Distributed Switch Architecture (DSA) framework is used to expose the i.MX RT1180 NETC switch ports to MPU side. In this architecture, one of the NETC switch ports or ENETC port is used as the data interface. Different interfaces (for example LPSPI, I2C or message unit) can be used as management interfaces.

For more information regarding Linux DSA, refer to https://docs.kernel.org/networking/dsa/dsa.html.

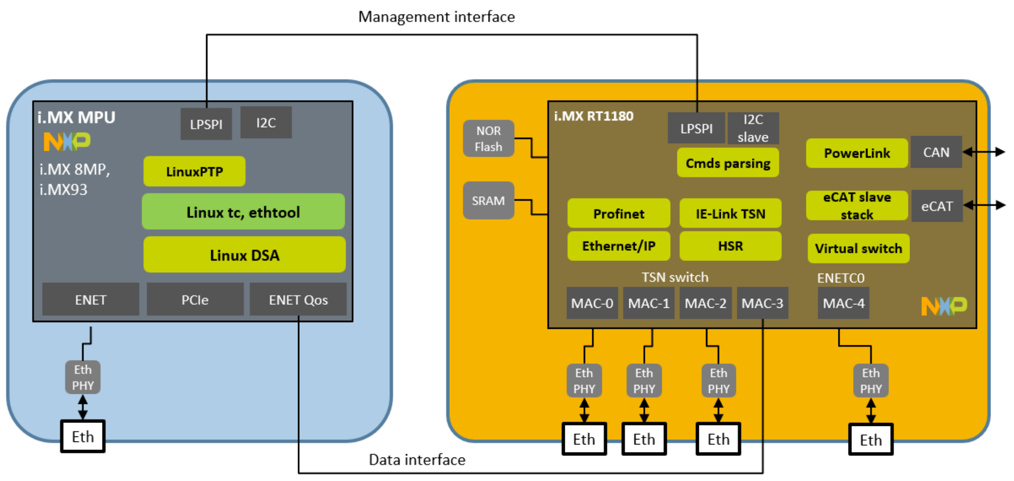


Figure 48. Using switch port as DSA CPU port which is connected to MPU

Note: *The industrial protocols listed in the box of i.MX RT1180 are not supported yet.*

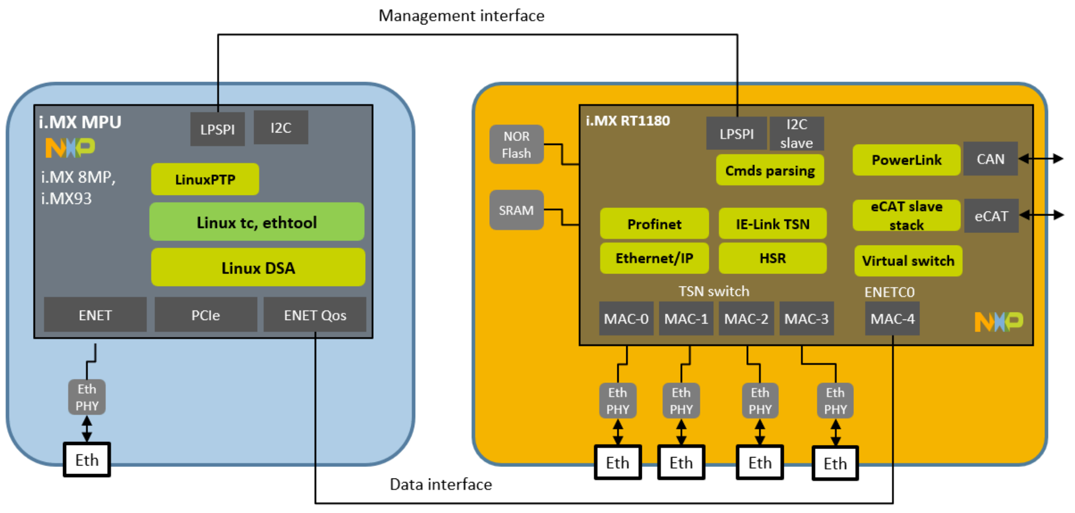


Figure 49. Using ENETC port as DSA CPU port which is connected to MPU

The software architecture implementation includes:

- NETC DSA switch driver on Linux
- Device driver of DSA control interface on Linux DSA
- Service driver of DSA control interface on i.MX RT1180
- NETC DSA switch configuration on i.MX RT1180
  - Using NETC switch port as DSA CPU port
  - Using ENETC port as DSA CPU port

  The external Ethernet ports of i.MX RT1180 are exposed to MPU with Linux DSA framework. The ports can be viewed as standard Ethernet ports which could support the below operations:
  - Binding Linux IP address to a specific port
  - Broadcast on a specific port and others

  To identify the source traffic of incoming traffic, the outer VLAN tag is used as the DSA tag to indicate which port the traffic is coming from. The VID field of the VLAN tag is encoded to include the source/destination port. Below is the description of the12-bit VID field:

Table 43. VID field description

`11`	`10`	`9`	`8`	`7`	`6`	`5`	`4`	`3`	`2`	`1`	`0`
`RSV`		`VBID`	`SWITCH_ID`			`VBID`		`PORT`			

***Note:** The above custom VID definition is coming from the Linux `tag_8021q` source code (`net/dsa/tag_8021q.c`) which is used by the Linux NETC DSA driver. As a result, VLAN ID 3072 to 3076 and 3088 are reserved by the Linux NETC DSA driver and will not be allowed to be used by the user.*

4.2.1 Using one of the i.MX RT1180 switch ports as DSA CPU port

Consider the case when one of the i.MX RT1180 switch ports is used as DSA CPU port that is connected to MPU. In such a situation, the packets received on the external ports of i.MX RT1180 are filtered using Ingress Port Filter Table. The incoming packets are filtered according to the ingress port ID, and forwarded to the DSA CPU port by either stream forwarding (by-passing the 802.1Q bridge forwarding) or 802.1Q bridge forwarding using Ingress Stream Table. The outer VLAN tag is added by using the Egress Treatment Table on egress DSA CPU port.

The packets received on the DSA CPU port are filtered according to the DSA tag, which is viewed as a VID from i.MX RT1180 perspective. This is done by using Ingress Port Filter Table and the packets are directed to the port mapped to the VID by using Ingress Stream Table. The outer VLAN tag is removed on ingress DSA CPU port by also using Ingress Stream Table.

4.2.2 Using i.MX RT1180 ENETC port as DSA CPU port

For the case using the ENETC port as the DSA CPU port that is connected to MPU, the packets received on the external ports of i.MX RT1180 are filtered using the Ingress Port Filter Table. The incoming packets are filtered according to the ingress port ID, and forwarded to the internal switch port (also called switch management port). The forwarding is done by either stream forwarding (by-passing the 802.1Q bridge forwarding) or 802.1Q bridge forwarding using Ingress Stream Table. The software virtual switch running on the CPU of i.MX RT1180 receives the packets and the software virtual switch adds the DSA tag and forwards the packets to the ENETC port.

The packets received on the ENETC port are forwarded by the software virtual switch to the internal switch port, where the packets are filtered according to the DSA tag, which is viewed as VID from i.MX RT1180 perspective. This is done by using Ingress Port Filter Table and the packets are directed to the port mapped to the VID by using Ingress Stream Table. The outer VLAN tag is removed by using Egress Treatment Table on egress switch port.

4.3 Hardware setup

This section describes how to setup hardware for Heterogeneous Multi-SoC framework on both the MPU side and i.MX RT1180 side.

The currently supported MPU platforms include i.MX 8M Plus EVK, i.MX 93 EVK, and i.MX 943 19x19 EVK.

Note: *For i.MX RT1180 EVK, the supported board revision is SCH-50577 REV C2 (700-50577 REVC), which has i.MX RT1180 Rev. B0 chip.*

4.3.1 Hardware preparation for i.MX RT1180 EVK

This section describes how to burn DSA image to i.MX RT1180 EVK board.

4.3.1.1 Hardware setup

To burn the DSA image to i.MX RT1180 EVK board, use the hardware setup described in this section.

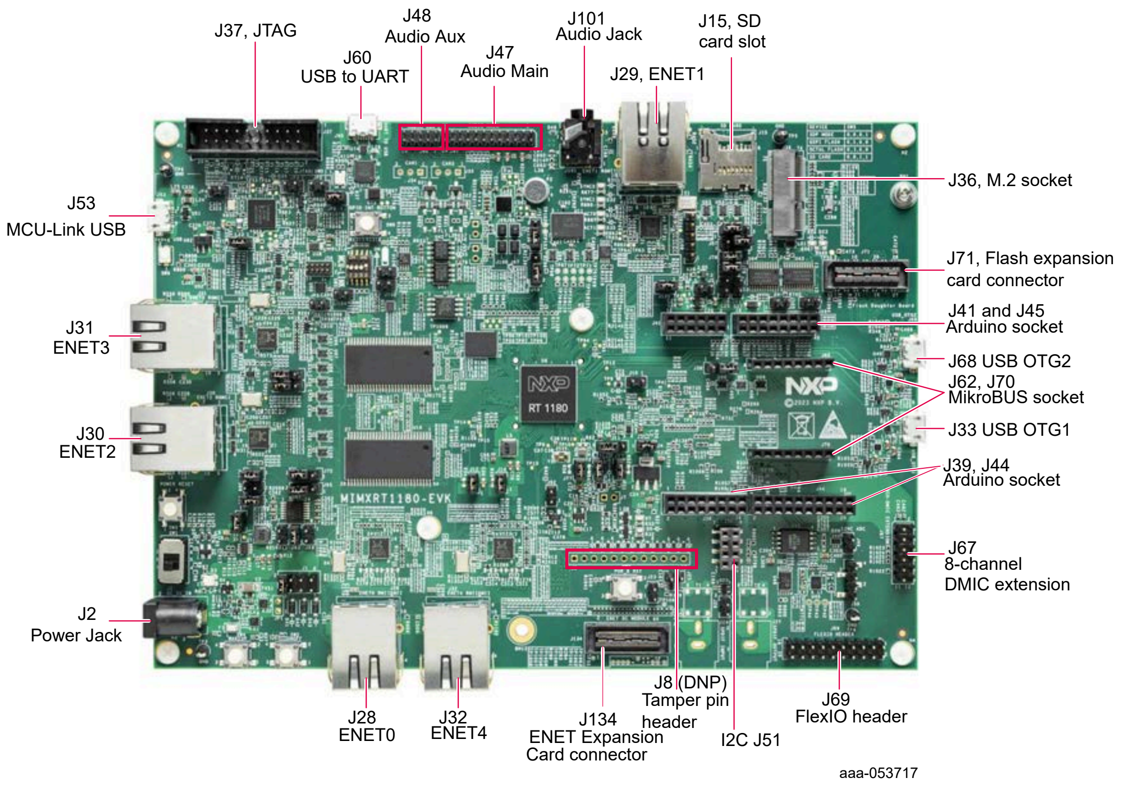


Figure 50. MIMXRT1180-EVK top view connectors

Figure 50 shows the connectors on the top-side view of the MIMXRT1180-EVK board.

To burn the image to i.MX RT1180, connect the PC to the MCU-Link USB J53 using a micro-USB cable.

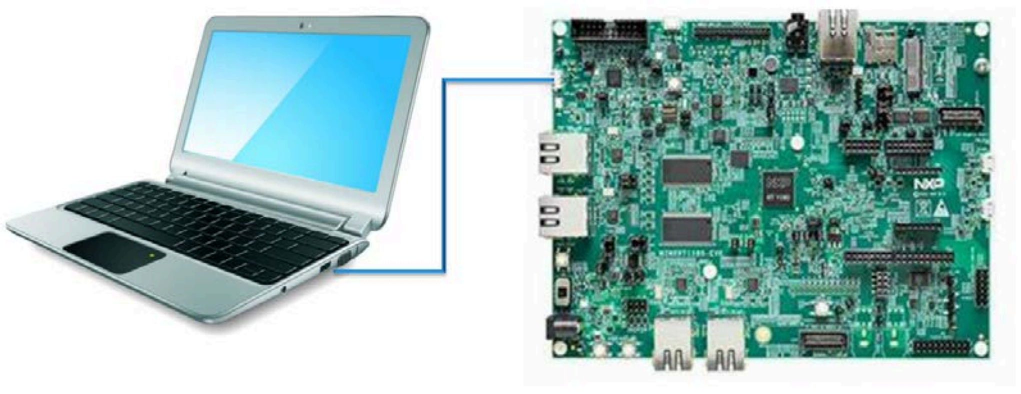


Figure 51. Connection between a PC and the MIMXRT1180-EVK board

4.3.1.2 Preparing the DSA images

There are two ways to get the DSA (distributed switch architecture) images, which are described in the following sections:

- Using pre-built images
- Building the DSA image

4.3.1.2.1 Using pre-built image

Real-time Edge release package includes DSA images in root file system. The folder is */example/heterogeneous-multi-soc*. The content of this package is as follows:

```
/examples/heterogeneous-multi-soc/
└── dsa-switch-evkmimxrt1180-cm33
    ├── ram_release
    │   ├── dsa_enetc.elf
    │   └── dsa_switch.elf
    └── release
        ├── dsa_enetc.bin
        ├── dsa_enetc.elf
        ├── dsa_switch.bin
        └── dsa_switch.elf
```

Images under the `ram_release` folder are for RAM boot. Images under `release` folder are for flash boot. The `dsa_enetc` image supports enetc as DSA CPU port. The `dsa_switch` image supports one of switch ports as DSA CPU port.

In this example we need to copy `release/dsa_switch.elf` to the PC.

4.3.1.2.2 Building the DSA image

The demo application images `dsa_switch.elf/dsa_switch.bin` and `dsa_enetc.elf/dsa_enetc.bin` for i.MX RT1180 EVK are installed into target rootfs with the MPU Yocto image build. These images are available in the `/examples/heterogeneous-multi-soc/dsa-switch-evkmimxrt1180-cm33/` directory.

Also, the demo application images can be deployed into the Yocto build directory by using the following Yocto commands:

```
# For i.MX 8M Plus EVK
$ DISTRO=nxp-real-time-edge MACHINE=imx8mp-lpddr4-evk source real-time-edge-
setup-env.sh -b <build_dir>
$ bitbake dsa-switch-evkmimxrt1180-cm33

# For i.MX 93 EVK
$ DISTRO=nxp-real-time-edge MACHINE=imx93evk source real-time-edge-setup-env.sh
 -b <build_dir>
$ bitbake dsa-switch-evkmimxrt1180-cm33
```

After the Yocto `bitbake` command is used, the demo application images can be found in the below directory on the build host:

```
<build_dir>/tmp/deploy/images/<MACHINE>/examples/heterogeneous-multi-soc/dsa-
switch-evkmimxrt1180-cm33
```

The demo applications are available in both `release` and `ram_release` mode. The `release` image boots from the external QuadSPI NOR flash and then relocates to internal memory (Code TCM, System TCM, and OCRAM) and must be flashed in the external NOR flash. The `ram_release` image boots from internal memory and must be loaded directly into internal memory by host debug tools.

4.3.1.3 Preparing the SPT tool

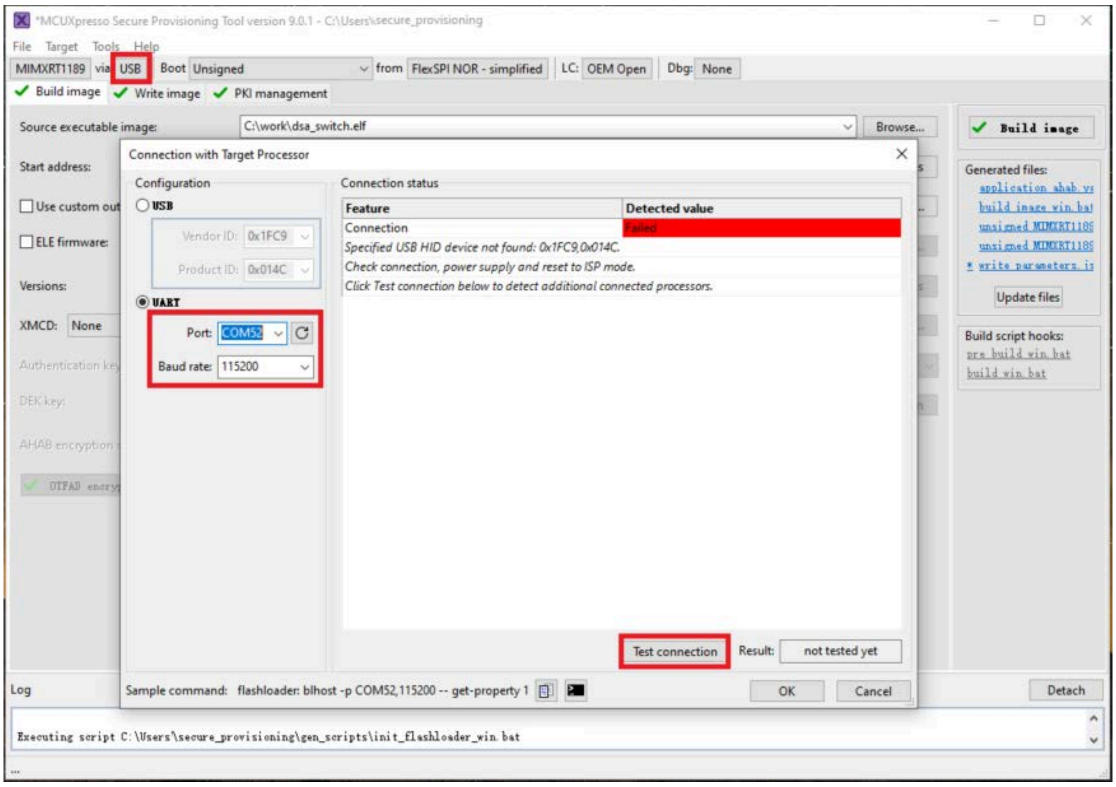
Download the MCUXpresso Secure Provisioning Tool (SPT) from the below link on the host PC: https://www.nxp.com/webapp/Download?colCode=MCUXPRESSO-SECURE-PROVISIONING-V9.0.1-WIND&appType=license.

Then, install the MCUXpresso Secure Provisioning tool to the PC.

4.3.1.4 Burning the DSA image

Follow the steps described below to burn the DSA image on i.MX RT1180.

1. Set “**SW5[1..4]**” of the board to “**0000**” to enable SDP mode.
2. Open SPT. Then, click “**USB**” to change to “**UART**”. Then, select the “**Port**” of the board and set the “**Baud rate**” to “**115200**”.



MIMXRT1189 via USB Boot Unsigned from FlexSPI NOR - simplified LC: OEM Open Dbg: None

Source executable image: C:\work\dsa_switch.elf

Connection with Target Processor

Configuration: UART — Port: COM52, Baud rate: 115200

Connection status

Feature	Detected value
Connection	Failed

Specified USB HID device not found: 0x1FC9,0x014C.
Check connection, power supply and reset to ISP mode.
Click Test connection below to detect additional connected processors.

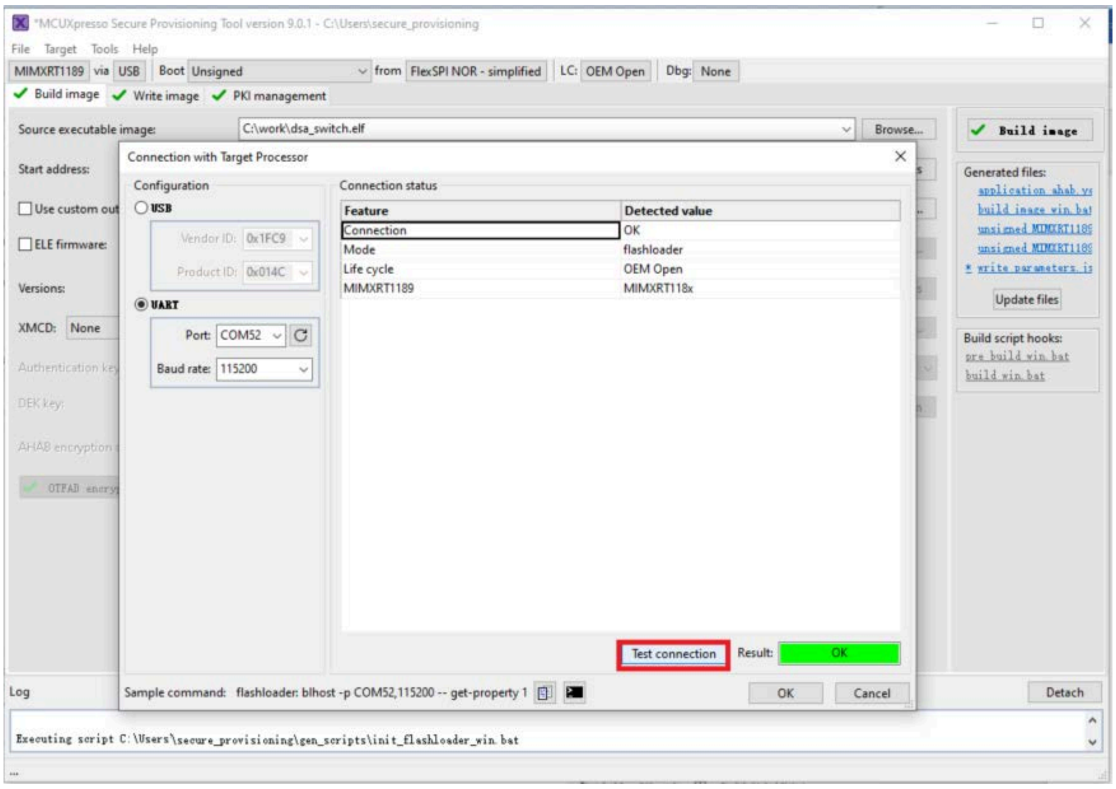
Result: not tested yet

Sample command: flashloader: blhost -p COM52,115200 -- get-property 1

```
Executing script C:\Users\secure_provisioning\gen_scripts\init_flashloader_win.bat
```

Figure 52. UART setting

3. Click the button “**Test connection**” to check the UART connection.



MIMXRT1189 via USB Boot Unsigned from FlexSPI NOR - simplified LC: OEM Open Dbg: None

Source executable image: C:\work\dsa_switch.elf

Connection with Target Processor

Configuration: USB (Vendor ID: 0x1FC9, Product ID: 0x014C); UART selected, Port: COM52, Baud rate: 115200

Connection status

Feature	Detected value
Connection	OK
Mode	flashloader
Life cycle	OEM Open
MIMXRT1189	MIMXRT118x

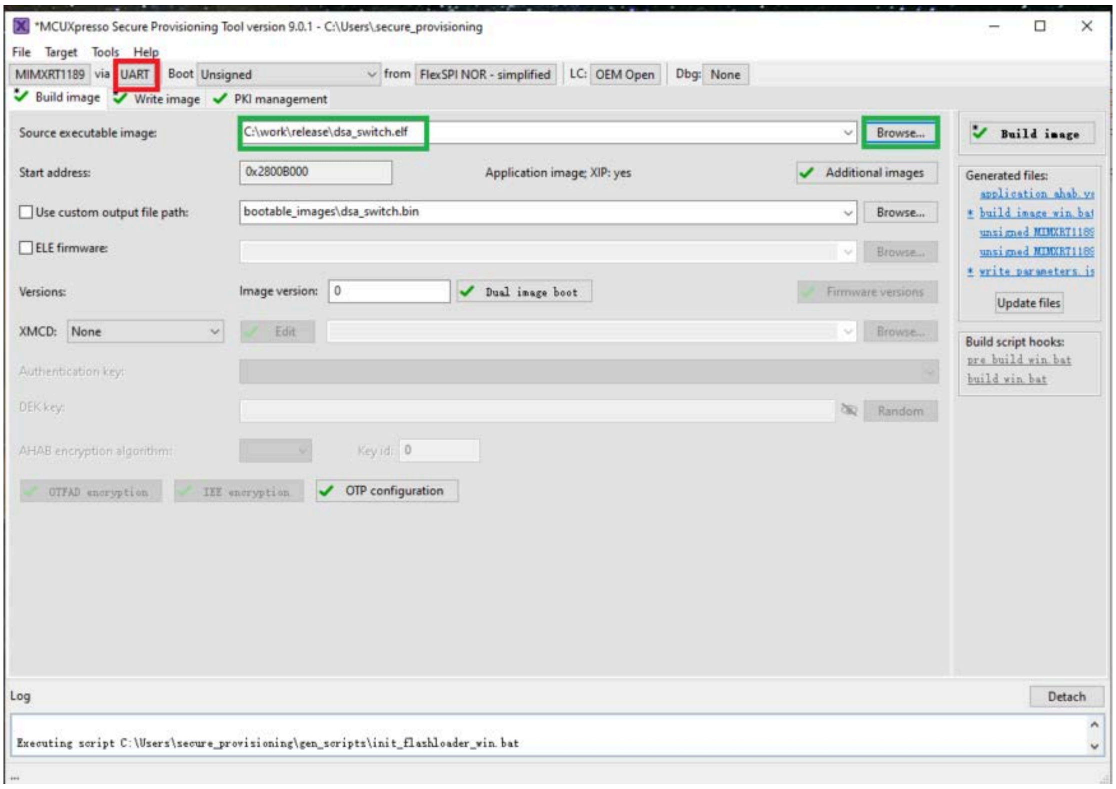
Result: OK

Sample command: flashloader: blhost -p COM52,115200 -- get-property 1

```
Executing script C:\Users\secure_provisioning\gen_scripts\init_flashloader_win.bat
```

Figure 53. UART connection

4. Click “**Browse**” to select “**release/dsa_switch.elf**”



MIMXRT1189 via UART Boot Unsigned from FlexSPI NOR - simplified LC: OEM Open Dbg: None

Source executable image: C:\work\release\dsa_switch.elf

Start address: 0x2800B000 Application image; XIP: yes Additional images

Use custom output file path: bootable_images\dsa_switch.bin

ELE firmware:

Versions: Image version: 0 Dual image boot

XMCD: None

Authentication key:

DEK key:

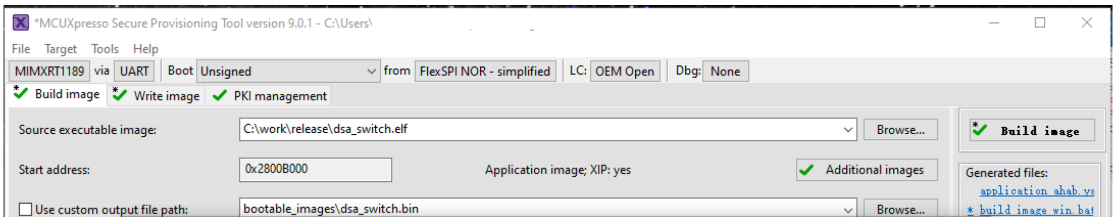
AHAB encryption algorithm: Key id: 0

OTFAD encryption IEE encryption OTP configuration

```
Executing script C:\Users\secure_provisioning\gen_scripts\init_flashloader_win.bat
```

Figure 54. Select “release/dsa_switch.elf”

5. Click the “**build image**” button to build the images.



*MCUXpresso Secure Provisioning Tool version 9.0.1 - C:\Users\

File Target Tools Help

MIMXRT1189 via UART Boot Unsigned from FlexSPI NOR - simplified LC: OEM Open Dbg: None

Build image | Write image | PKI management

Source executable image: C:\work\release\dsa_switch.elf Browse...

Start address: 0x2800B000 Application image; XIP: yes Additional images

Use custom output file path: bootable_images\dsa_switch.bin Browse...

Build image

Generated files: application_ahab.ya | build_image_win.bat

Figure 55. Building the image

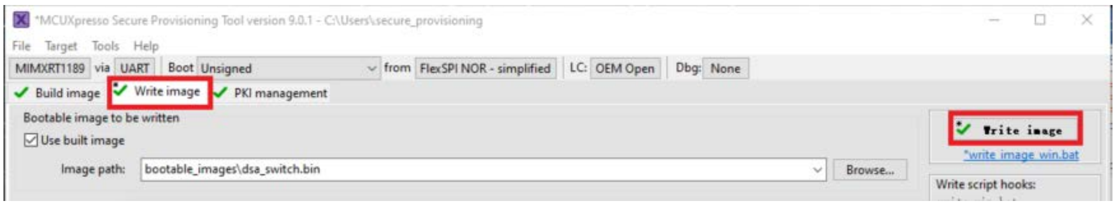
A success message is displayed in the Build image window after successful build. For example:

```
Success building image
```

The message is also displayed in the Secure Provisioning Tool window bottom.

```
Status of the last operation: Success building image
```

6. Click the “**write image**” button to burn the image to the flash.



*MCUXpresso Secure Provisioning Tool version 9.0.1 - C:\Users\secure_provisioning

File Target Tools Help

MIMXRT1189 via UART Boot Unsigned from FlexSPI NOR - simplified LC: OEM Open Dbg: None

Build image | Write image | PKI management

Bootable image to be written

Use built image

Image path: bootable_images\dsa_switch.bin Browse...

Write image

*write_image_win.bat

Write script hooks:

Figure 56. burn image to the flash

7. Change **SW5** to “**0100**”- flash boot mode.

4.3.1.5 Checking the DSA image

1. Start Putty or other serial terminal tools to access i.MX RT1180.
   The configuration of the terminal is as follows:
   - Speed (bit/s): 115200
   - Data bits: 8
   - Stop bits: 1
   - Parity: None
   - Flow control: None
2. Here is a partial log at startup

```
INFO           0 app main_task             : DSA SWITCH main_task started

Copyright  2024  NXP
INFO           0 app dsa_ctrl_task         : DSA control task started

BRIDGE>>
INIT           0.000000000 os     genavb_init                        : NXP's
 GenAVB/TSN stack version master-fe9e6c3e
INFO           0.000000000 os     __netc_1588_init                   : timerClk
 period (ns): 4 genClk period (ns): 100000
INFO           0.000000000 os     msgintr_msix_msgaddr               : msgintr
 addr return for the MSI-X Entry
INFO           0.000000000 os     __netc_1588_init                   :
 refClkHz: 240000000 defaultPpb: 0
INFO           0.000000000 os     __netc_1588_init                   : Alarm2
 is configured
```

```
INFO          0.000000000 os     __netc_1588_init                  : Alarm1
 is configured for Fiper
...
INIT          0.000000000 os     logical_port_init                 :
 logical_port(0) enabled, endpoint(0)
INIT          0.000000000 os     logical_port_init                 :
 logical_port(1) enabled, endpoint(1)
INIT          0.000000000 os     logical_port_init                 :
 logical_port(2) enabled, bridge(0, 0)
INIT          0.000000000 os     logical_port_init                 :
 logical_port(3) enabled, bridge(0, 1)
INIT          0.000000000 os     logical_port_init                 :
 logical_port(4) enabled, bridge(0, 2)
INIT          0.000000000 os     logical_port_init                 :
 logical_port(5) enabled, bridge(0, 3)
INIT          0.000000000 os     logical_port_init                 :
 logical_port(6) enabled, bridge(0, 4)
INIT          0.000000000 os     logical_port_init                 : logical
 ports configuration: 2 endpoints, 5 bridge ports
...
Heap used: 67752, free: 58200, min free: 58200
Malloc failed counter: 0

Total CPU load :     0
```

4.3.2 Hardware setup for i.MX 8M Plus EVK and i.MX RT1180 EVK

1. **SPI connection between i.MX 8M Plus EVK and i.MX RT1180 EVK**
   On i.MX 8M Plus EVK, ECSPI2 pins are available on **J21** connector, but due to signal incompatibility with the i.MX RT1180 EVK LPSPI3 pins on **J44** connector, board rework is needed on i.MX 8M Plus EVK to replace a translating transceiver NBT0104 **U55** which has limited capacitive loading for 70 pF with NTS0104GU12, which has internal pull-up of 10K and bigger capacitive loading. Refer to the schematic shown in the Figure 57 below:

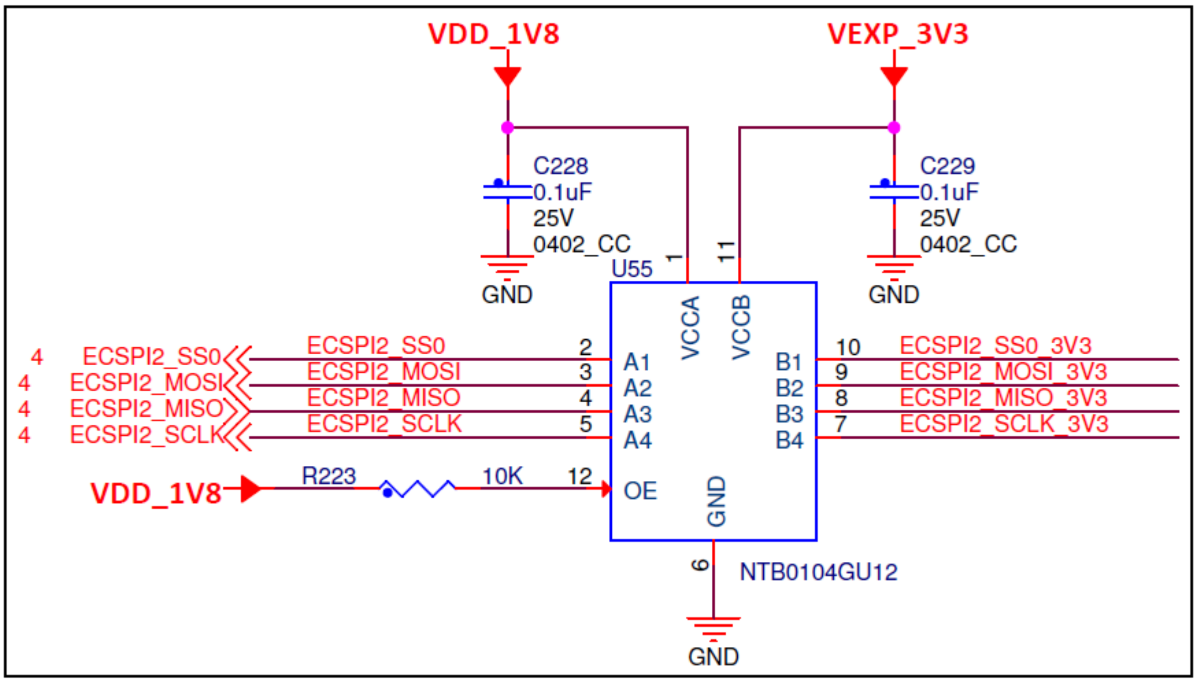


Figure 57. Hardware rework for SPI signals on i.MX 8M Plus EVK

Connect ECSPI2 pins on J21 connector to i.MX RT1180 EVK LPSPI3 pins by following the connection in Table 44.

Table 44. Pin connection between i.MX 8M Plus EVK and i.MX RT1180 EVK

i.MX 8M Plus EVK		Connection	i.MX RT1180 EVK	
Pin	Function		Pin	Function
19	ECSPI2_MOSI	<->	10	LPSPI3_SIN
21	ECSPI2_MISO	<->	8	LPSPI3_SOUT
23	ECSPI2_SCLK	<->	12	LPSPI3_CLK
24	ECSPI2_SS0	<->	6	LPSPI3_PCS0
25	GND	<->	14	GND

2. **Ethernet connection between i.MX 8M Plus EVK and i.MX RT1180 EVK**
For the case using one of the i.MX RT1180 external switch ports as DSA CPU port, use Ethernet cable to connect i.MX 8M Plus EVK ENET2 port (`eth1` in Linux) on **J8** RJ45 connector and i.MX RT1180 EVK NETC switch port 3 (ENET3) on **J31** RJ45 connector.
For the case using ENETC port as DSA CPU port, use Ethernet cable to connect i.MX 8M Plus EVK ENET2 port (`eth1` in Linux) on **J8** RJ45 connector and i.MX RT1180 EVK ENETC port (ENET4) on **J32** RJ45 connector.

4.3.3 Hardware setup for i.MX 93 EVK and i.MX RT1180 EVK

1. **LPSPI connection between i.MX 93 EVK and i.MX RT1180 EVK**
Using flying wire to connect i.MX 93 EVK LPSPI3 pins on **J1001** connector and i.MX RT1180 EVK LPSPI3 pins on **J44** connector by following the pin connection in below table:

Table 45. Pin connection between i.MX 93 EVK and i.MX RT1180 EVK

i.MX 93 EVK		Connection	i.MX RT1180 EVK	
Pin	Function		Pin	Function
19	LPSPI3_MOSI	<->	10	LPSPI3_SIN
21	LPSPI3_MISO	<->	8	LPSPI3_SOUT
23	LPSPI3_CLK	<->	12	LPSPI3_CLK
24	LPSPI3_PCS0	<->	6	LPSPI3_PCS0
25	GND	<->	14	GND

2. **Ethernet connection between i.MX 93 EVK and i.MX RT1180 EVK**
For the case using one of the i.MX RT1180 external switch ports as DSA CPU port, use Ethernet cable to connect i.MX 93 EVK ENET_QOS port (`eth1` in Linux) on **J501** RJ45 connector and i.MX RT1180 EVK NETC switch port 3 (ENET3) on **J31** RJ45 connector.
For the case using ENETC port as DSA CPU port, use Ethernet cable to connect i.MX 93 EVK ENET_QOS port (`eth1` in Linux) on **J501** RJ45 connector and i.MX RT1180 EVK ENETC port (ENET4) on **J32** RJ45 connector.

4.3.4 Hardware setup for i.MX 943 EVK and i.MX RT1180 EVK

1. **LPSPI connection between i.MX 943 EVK and i.MX RT1180 EVK**
Using flying wire to connect i.MX 943 EVK LPSPI3 pins on **J47** connector and i.MX RT1180 EVK LPSPI3 pins on **J44** connector by following the pin connection in below table:

Table 46. Pin connection between i.MX 943 EVK and i.MX RT1180 EVK

i.MX 943 EVK		Connection	i.MX RT1180 EVK	
Pin	Function		Pin	Function
8	LPSPI3_MOSI	<->	10	LPSPI3_SIN
10	LPSPI3_MISO	<->	8	LPSPI3_SOUT
12	LPSPI3_CLK	<->	12	LPSPI3_CLK
6	LPSPI3_PCS0	<->	6	LPSPI3_PCS0
14	GND	<->	14	GND

2. **Ethernet connection between i.MX 943 EVK and i.MX RT1180 EVK**
   For the case using one of the i.MX RT1180 external switch ports as DSA CPU port, use Ethernet cable to connect i.MX 943 EVK ENETC2 port (`eth2` in Linux) on **J27** RJ45 connector and i.MX RT1180 EVK NETC switch port 3 (ENET3) on **J31** RJ45 connector.

4.3.5 Bringing up MPU and i.MX RT1180 EVK

1. Make SPI and Ethernet connections between i.MX 8M Plus EVK, i.MX 93 EVK or i.MX 943 EVK and i.MX RT 1180 EVK by following the steps in either of the following:
   - Section 4.3.2
   - Section 4.3.3
   - Section 4.3.4
2. Make a serial connection between the board and the host PC using USB cable:
   - Use **J23** connector on i.MX 8M Plus EVK.
   - Use **J1401** connector on i.MX 93 EVK.
   - Use**J15** connector on i.MX 943 EVK.

   Open the terminal application on host PC, such as PuTTY or TeraTerm, and connect to the third debug console that is used by Linux running on the MPU.
3. Make a serial connection between **J53** connector on i.MX RT1180 EVK and USB connector on host PC using USB cable. Open another terminal application on the host PC, and connect to the debug console, which is used by FreeRTOS running on i.MX RT1180 EVK. Note that the terminal settings '**Implicit CR in every LF**' must be enabled for correct display of the logs.
4. Power on i.MX RT1180 EVK board, and flash the binary image `release/dsa_switch.bin` or `release/dsa_enetc.bin` on i.MX RT1180 EVK and power cycle the board. Alternatively, one can download the elf image `ram_release/dsa_switch.elf` or `ram_release/dsa_enetc.elf` by host debug tools (for example, using JTAG debugger connected to **J37** connector). After the demo application is running, there must be continuous logs displayed in the console.
5. Power on the MPU board and enter the U-Boot command line. Then execute the corresponding commands depending on which port on i.MX RT1180 is used as DSA CPU port.
   For the case using one of the i.MX RT1180 external switch ports as DSA CPU port:

```
# For i.MX 8M Plus EVK
u-boot=> setenv fdtfile imx8mp-evk-revb4-dsa.dtb
u-boot=> boot

# For i.MX 93 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-dsa.dtb
u-boot=> boot

# For i.MX 943 EVK
u-boot=> setenv fdtfile imx943-evk-hms-dsa.dtb
```

```
u-boot=> boot
```

For the case using ENETC port as DSA CPU port:

```
# For i.MX 8M Plus EVK
u-boot=> setenv fdtfile imx8mp-evk-revb4-dsa-enetc.dtb
u-boot=> boot

# For i.MX 93 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-dsa-enetc.dtb
u-boot=> boot
```

After Linux is up on MPU, refer to the Section 4.4 for the detailed usage on how to show and configure various NETC switch features on i.MX RT1180.

4.4 Runtime usage on MPU and i.MX RT1180 EVK

This section describes the major i.MX RT1180 NETC switch features that can be queried or configured on MPU + i.MX RT1180 EVK heterogeneous multi-SoC architecture using Linux DSA (Distributed Switch Architecture).

4.4.1 i.MX RT1180 NETC switch interface under Linux

On the MPU, after Linux is up, the Linux NETC DSA driver initializes successfully. At this stage, the i.MX RT1180 NETC front panel switch port 0 (ENET0), port 1 (ENET1), port 2 (ENET2), and port 3 (ENET3) must have a network device interface attached with the "hmsXpY" format.

The network port name of the format “hmsXpY” indicates a network port implemented through the Heterogeneous Multi-SoC Framework. Where ‘X’ represents the device number and ‘Y’ represents the port number.

Note: *The “swpN” port indicates i.MX943 EVK INTERNAL switch port instead of Heterogeneous Multi-SoC port.*

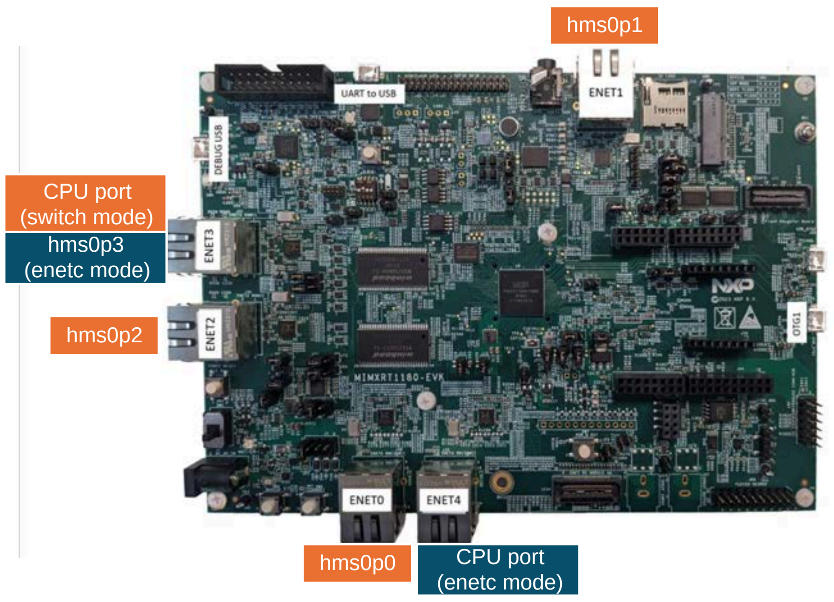


Figure 58. MIMXRT1180-EVK hmsXpY mapping

Table 47. i.MX RT1180 EVK NETC switch port mapping in Linux on MPU

i.MX RT1180 EVK NETC switch port	Label on i.MX RT1180 EVK	Linux network device name on MPU
Port 0	ENET0 (J28)	hms0p0
Port 1	ENET1 (J29)	hms0p1
Port 2	ENET2 (J30)	hms0p2
Port 3	ENTE3 (J31)	N/A or hms0p3 (see note)
Port 4 (internal)	N/A	N/A

Note: *Linux DSA does not currently create user network device for CPU port. When using one of the i.MX RT1180 external switch ports as the CPU port (that is, the i.MX RT1180 NETC switch port 3 is used as CPU port), it is not visible to the user as a normal network device in Linux. When using ENETC port as CPU port, the i.MX RT1180 NETC switch port 3 is shown as* `hms0p3` *in Linux.*

One can use `ifconfig -a` command to show these network device interfaces.

Example - NETC switch port interfaces `hmsXpY` shown by `ifconfig -a`

```
# ifconfig -a
[…]
hms0p0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet6 fe80::204:9fff:fe09:b564  prefixlen 64  scopeid 0x20<link>
        ether 00:04:9f:09:b5:64  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 30  bytes 6180 (6.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

hms0p1: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet6 fe80::204:9fff:fe09:b564  prefixlen 64  scopeid 0x20<link>
        ether 00:04:9f:09:b5:64  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17  bytes 2995 (2.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

hms0p2: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet6 fe80::204:9fff:fe09:b564  prefixlen 64  scopeid 0x20<link>
        ether 00:04:9f:09:b5:64  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 20  bytes 3330 (3.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

/* hms0p3 is only available when using ENETC port as a DSA CPU port */
hms0p3: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet6 fe80::204:9fff:fe09:b564  prefixlen 64  scopeid 0x20<link>
        ether 00:04:9f:09:b5:64  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
```

Also, the `ip link show` command uses the `hms0pN@eth1` name format to indicate the associated master Ethernet interface (`eth2`) on MPU for the DSA switch port.

Example - NETC switch port interfaces `hmsXpY` and DSA master interface `eth2` shown by `ip link show`.

```

# ip link show
[…]
10: hms0p0@eth2: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc
 noqueue state UP mode DEFAULT group defaul
t qlen 1000
    link/ether 00:04:9f:09:b5:64 brd ff:ff:ff:ff:ff:ff
11: hms0p1@eth2: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc
 noqueue state UP mode DEFAULT group defaul
t qlen 1000
    link/ether 00:04:9f:09:b5:64 brd ff:ff:ff:ff:ff:ff
12: hms0p2@eth2: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc
 noqueue state UP mode DEFAULT group defaul
t qlen 1000
    link/ether 00:04:9f:09:b5:64 brd ff:ff:ff:ff:ff:ff

/* hms0p3@eth2 is only available when using ENETC port as a DSA CPU port */
```

```
13: hms0p3@eth2: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc
 noqueue state UP mode DEFAULT group defaul
t qlen 1000
    link/ether 00:04:9f:09:b5:64 brd ff:ff:ff:ff:ff:ff
```

4.4.2 Maximum frame size configuration

The Linux NETC DSA driver correlates between Layer-2 maximum frame size and Layer-3 MTU. Typical settings of Layer-2 maximum frame size is 1518 bytes for standard untagged frames and the corresponding Layer-3 MTU is 1500 bytes.

Example 1: Setting Layer-3 MTU of NETC switch port 1 (`hms0p1`) to 1000 bytes

```
# ifconfig hms0p1 mtu 1000
or
# ip link set dev hms0p1 mtu 1000
```

Note:

Setting the Layer-3 MTU for an external NETC switch port on i.MX RT1180 EVK to a value larger than 1500 bytes also changes the Layer-3 MTU of both the CPU port on i.MX RT1180 EVK and the master Ethernet interface (`eth1`) on MPU accordingly.

As the maximum MTU of `eth1` is 1600, the maximum MTU value that can be configured on an external NETC switch port is 1596 bytes. However, the NETC Ethernet MAC supports configurable maximum frame size up to 2000 bytes. The 4 extra bytes in MTU for both the CPU port on i.MX RT1180 EVK and the master Ethernet interface (`eth1`) on MPU are used to store the 4-bytes 802.1Q DSA tag.

4.4.3 Single port mode for i.MX RT1180 NETC switch ports

After Linux is up on MPU, by default all the i.MX RT1180 NETC switch ports available in Linux works in single port mode. In this configuration mode the traffic received on other switch ports is forwarded to the CPU port (i.MX RT1180 NETC switch port 3). Each of the other external switch port interface can be used independently to send and receive packets.

Example 1: **Single port configuration of the Linux NETC DSA driver**

```
/* configure IP address on external switch interfaces */
# ip addr add 192.168.0.1/24 dev hms0p0
# ip addr add 192.168.1.1/24 dev hms0p1
# ip addr add 192.168.2.1/24 dev hms0p2
/* hms0p3 is only available when using ENETC port as a DSA CPU port */
# ip addr add 192.168.3.1/24 dev hms0p3
/* master interface(eth1 or eth2) to be brought up first - up by default */
# ip link set eth1 up
/* bring up the switch slave interfaces */
# ip link set hms0p0 up
# ip link set hms0p1 up
# ip link set hms0p2 up
/* hms0p3 is only available when using ENETC port as a DSA CPU port */
# ip link set hms0p3 up
```

Assuming there is one remote host connected to each of the external switch ports, one can ping the remote host.

```
/* Assuming the IP is 192.168.0.2 for the remote host connected to hms0p0 */
# ping 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=1.75 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=1.77 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=1.76 ms
[…]

/* Assuming the IP is 192.168.1.2 for the remote host connected to hms0p1 */
# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=2.80 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=1.76 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=1.77 ms
[…]

/* Assuming the IP is 192.168.2.2 for the remote host connected to hms0p2 */
# ping 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=3.23 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=1.75 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=1.75 ms
[…]

/* hms0p3 is only available when using ENETC port as a DSA CPU port */
/* Assuming the IP is 192.168.3.2 for the remote host connected to hms0p3 */
# ping 192.168.3.2
PING 192.168.3.2 (192.168.3.2) 56(84) bytes of data.
64 bytes from 192.168.3.2: icmp_seq=1 ttl=64 time=2.80 ms
64 bytes from 192.168.3.2: icmp_seq=2 ttl=64 time=1.76 ms
64 bytes from 192.168.3.2: icmp_seq=3 ttl=64 time=1.77 ms
[…]
```

4.4.4 Bridge mode for i.MX RT1180 NETC switch ports

In this configuration mode, the external switch ports (DSA slave interfaces) are added to a bridge for L2 forwarding. The `eth1` or `eth2` interface is brought up as the DSA master interface.

Example 1: Bridge configuration of the Linux NETC DSA driver

```
/* bring up master interface(eth1 or eth2) before the slave ports - up by
 default */
# ip link set eth1 up

/* bring up the switch slave interfaces */
# ip link set hms0p0 up
# ip link set hms0p1 up
# ip link set hms0p2 up
/* hms0p3 is only available when using ENETC port as a DSA CPU port */
# ip link set hms0p3 up

/* create bridge */
# ip link add name br0 type bridge

/* add the external switch ports to the bridge */
# ip link set dev hms0p0 master br0
# ip link set dev hms0p1 master br0
# ip link set dev hms0p2 master br0
/* hms0p3 is only available when using ENETC port as DSA CPU port */
# ip link set dev hms0p3 master br0
```

Starting from Real-time Edge 3.2 release, the MAC address of a newly created bridge is no longer the same as that of `hms0pN`; instead, it uses a randomly generated MAC address. This can cause communication issues between the bridge and the underlying devices. Please ensure that the bridge uses the same MAC address as `hms0pN`. Below are the configuration commands:

```
/* configure hms0pN MAC address as bridge's MAC address */
# MAC=$(cat /sys/class/net/hms0p2/address)
# ip link set dev br0 address $MAC
```

Set the Remote host's IP

```
/* Assuming the IP is 192.168.200.2 and net device is eth1 on the remote host
 * connected to one of the external switch ports (hms0p0, hms0p1 and hms0p2)
 * when using one of the i.MX RT1180 switch ports as a DSA CPU port or
 * (hms0p0, hms0p1, hms0p2 and hms0p3) when using ENETC port as DSA CPU port.
 */
/* Set the Remote host's IP and ping the br0 */
# ip addr add 192.168.200.2/24 dev eth1
```

Br0 pings the remote host machine

```
/* configure IP address and bring up the bridge */
# ip addr add 192.168.200.1/24 dev br0
# ip link set dev br0 up
# ping 192.168.200.2
PING 192.168.200.2 (192.168.200.2) 56(84) bytes of data.
64 bytes from 192.168.200.2: icmp_seq=1 ttl=64 time=1.76 ms
64 bytes from 192.168.200.2: icmp_seq=2 ttl=64 time=1.80 ms
64 bytes from 192.168.200.2: icmp_seq=3 ttl=64 time=1.77 ms
[…]
```

Execute the ping command on remote host machine

```
# ping 192.168.200.1
```

At the same time, the L2 forwarding works using the external switch ports (`hms0p0`, `hms0p1` and `hms0p2`) when using one of the i.MX RT1180 external switch ports as a DSA CPU port or (`hms0p0`, `hms0p1`, `hms0p2` and `hms0p3`) when using ENETC port as DSA CPU port. Assuming there is one remote host connected to each of the external switch ports, each remote host can ping other hosts.

4.4.5 i.MX RT1180 NETC switch port statistics counters

The NETC DSA switch driver supports `ethtool -S hmsXpY` statistics reporting for each external DSA slave switch port through the associated net devices. Note that the first 4 stats (`tx_packets, tx_bytes, rx_packets, rx_bytes`) are counted by Linux networking stack and the other stats are directly read from NETC switch hardware.

Example 1: Query the port statistics counters of NETC switch port 1 (`hms0p1`)

```

# ethtool -S hms0p1
```

```
NIC statistics:
     tx_packets: 232
     tx_bytes: 30985
     rx_packets: 1178
     rx_bytes: 76913
     in-bytes: 150749
     in-valid-bytes: 150749
     in-pause-frames: 0
     in-valid-frames: 1846
     in-vlan-frames: 0
     in-uc-frames: 814
     in-mc-frames: 85
     in-bc-frames: 947
     in-frames: 1846
[…]
     out-bytes: 381591
     out-valid-bytes: 381591
     out-pause-frames: 0
     out-valid-frames: 4539
     out-vlan-frames: 0
     out-uc-frames: 810
     out-mc-frames: 3673
     out-bc-frames: 56
     out-frames: 4539
[…]
     q0-rejected-bytes: 0
     q0-rejected-frames: 0
     q0-dequeue-bytes: 0
     q0-dequeue-frames: 0
     q0-dropped-bytes: 0
     q0-dropped-frames: 0
     q0-frames: 0
[…]
     q7-rejected-bytes: 0
     q7-rejected-frames: 0
     q7-dequeue-bytes: 0
     q7-dequeue-frames: 0
     q7-dropped-bytes: 0
     q7-dropped-frames: 0
     q7-frames: 0
```

4.4.6 VLAN configuration

By default, the DSA switch application running on the i.MX RT1180 EVK hardware platform configures the NETC switch as VLAN filtering enabled. The `bridge` tool from the `iproute2` package can be used to manipulate the VLAN filter table.

To make the bridge VLAN aware in Linux, run the below command to toggle the `vlan_filtering` property on the bridge that already exists.

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

By default, only the default `pvid` (1) of the bridge is installed on all the switch ports. So, all VLAN-tagged traffic, except that tagged with VID 1, is dropped.

Note: *Since the default `pvid` (port-based VLAN) is 1, all untagged traffic also gets internally processed by the switch as having VID 1. So*

- *Untagged traffic is treated the same as traffic tagged with VID 1, or any other value that the`pvid` has.*
- *Deleting VID 1 from the VLAN table effectively blocks untagged traffic too.*

Note: *VLAN IDs '3072' to '3076' and '3088' are reserved by the Linux NETC DSA driver and are not allowed to be used by the user.*

Example 1: To add a new VLAN filter entry by which both the NETC switch port 1 (`hms0p1`) and port 2 (`hms0p2`) can accept and transmit tagged traffic with VLAN ID 100:

```
# bridge vlan add dev hms0p1 vid 100
# bridge vlan add dev hms0p2 vid 100
```

Example 2: To delete the VLAN filter entry added in previous example.

```
# bridge vlan delete dev hms0p1 vid 100
# bridge vlan delete dev hms0p2 vid 100
```

Example 3: To display the current VLAN filter table:

```
# bridge vlan show
```

It is also possible for the switch to tag untagged traffic with a different VLAN ID on ingress using the `pvid` option.

Example 4: To add new VLAN filter entries by which both the NETC switch port 1 (`hms0p1`) and port 2 (`hms0p2`) can accept and transmit tagged traffic with VLAN ID 100 and 200, also configure the PVID of NETC switch port 1 (`hms0p1`) to 100 and configure the PVID of NETC switch port 2 (`hms0p2`) to 200.

```
# bridge vlan add dev hms0p1 vid 100 pvid
# bridge vlan add dev hms0p2 vid 100
# bridge vlan add dev hms0p1 vid 200
# bridge vlan add dev hms0p2 vid 200 pvid
```

Using the above configuration, when an untagged packet enters `hms0p1`, it gets internally processed by the switch as having VID 100. If it is forwarded to `hms0p2` as a result of FDB lookup and exits `hms0p2`, it is tagged with VLAN ID 100. Similarly, when an untagged packet enters `hms0p2`, it gets internally processed by the switch as having VID 200. If it is forwarded to `hms0p1` as a result of FDB lookup and exits `hms0p1`, it is tagged with VLAN ID 200.

In the above example, if the VLAN tag added by PVID on the original untagged packet is not desired, it can be stripped on egress using the `untagged` option.

Example 5: To strip the VLAN tag added by PVID on untagged source traffic:

```
# bridge vlan add dev hms0p1 vid 100 pvid untagged
# bridge vlan add dev hms0p2 vid 100 untagged
# bridge vlan add dev hms0p1 vid 200 untagged
# bridge vlan add dev hms0p2 vid 200 pvid untagged
```

Using the above configuration, when an untagged packet enters `hms0p1`, it gets internally processed by the switch as having VID 100. If it is forwarded to `hms0p2` as a result of FDB lookup and exits `hms0p2`, it is still untagged. Similarly, when an untagged packet enters `hms0p2`, it gets internally processed by the switch as having VID 200. If it is forwarded to `hms0p1` as a result of FDB lookup and exits `hms0p1`, it is still untagged.

4.4.7 FDB configuration

By default, hardware MAC learning is enabled. FDB table entries <MAC, VID, PORT> are added or updated in the FDB table when packets with new unique <MAC + VID> are received on the NETC switch port.

Example 1: To display the current FDB table entries related to NETC switch port `hms0p1`

```
# bridge fdb show | grep hms0p1
```

Example 2: To add a static FDB entry associated with NETC switch port `hms0p1`

```
# bridge fdb add dev hms0p1 11:22:33:44:55:66 vlan 1 master static
```

Note that the VLAN filter entry for the VLAN ID specified for `vlan` option must have been added before adding the FDB entry for that VLAN ID.

Example 3: To delete a static FDB entry

```
# bridge fdb del dev hms0p1 11:22:33:44:55:66 vlan 1 master static
```

4.4.8 PTP Stack

PTP driver based on DSA has been added into Real-time Edge Linux. Linuxptp stack is supported.

For the detailed information about IEEE 1588/802.1AS, refer to Section 5.3.

1. **Hardware setup**

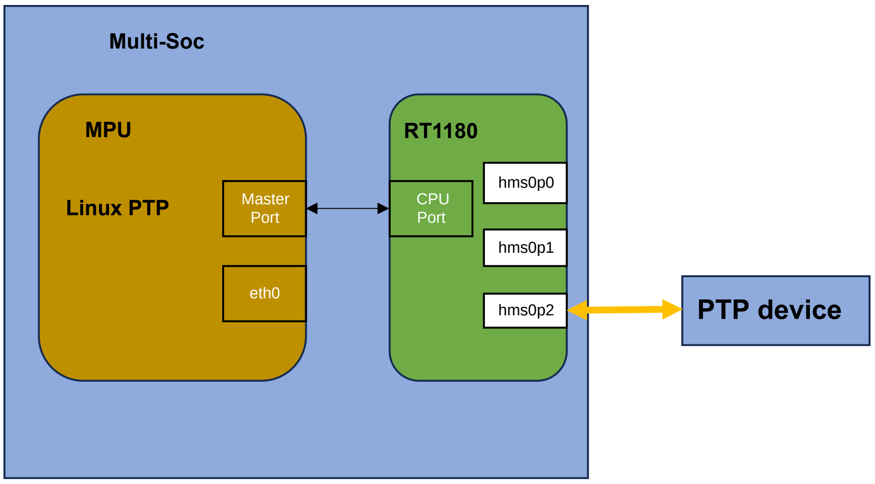


Figure 59. PTP Boards Connection

PTP stack only supports those i.MX RT1180 switch ports that are used as DSA CPU ports. To set up DSA, refer to Section 4.3.

For PTP peer, any board such as LS1028ARDB, i.MX 93, or i.MX 8M Plus can be used. Select an Ethernet port that supports a PTP connection with one of the i.MX RT1180 switch ports. Here, it is the hms0p2 port of i.MX RT1180.

2. **Running DSA switch elf on i.MX RT1180**

   Flash image and RAM images are provided by Real-time Edge. To burn the flash image, refer to the Section 4.3.1.1.

3. **Running ptp4l on boards**

   The below command can be used to start the ptp4l program on DSA switch (i.MX 8M Plus / i.MX 93 / i.MX 943) side.

   - The "`-i interface`" option specifies the PTP port.
   - The "`-p phc-device`" option specifies the PTP hardware clock device.

   In the above hardware setup, the hms0p2 port of i.MX RT1180 is used for clock synchronization, so "`-i hms0p2`" is used here.

```
# ptp4l -i hms0p2 -p /dev/ptpX -f /etc/ptp4l_cfg/gPTP.cfg -m
```

   The ethtool utility can be used to identify the PHC (PTP hardware clock) device number as shown below. The PTP Hardware Clock value displayed by *ethtool* is the index of the PTP hardware clock. It corresponds to the naming of the `/dev/ptp*` devices. Here, the PTP Hardware Clock is 1, which implies that "`-p /dev/ptp1`" must be used to specify the PHC device.

```
:~# ethtool -T hms0p2
Time stamping parameters for hms0p2:
Capabilities:
        hardware-transmit
        hardware-receive
        hardware-raw-clock
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
        off
        on
Hardware Receive Filter Modes:
        none
        ptpv2-l2-event
```

Scenario 1: MPU + i.MX RT1180 running as PTP master clock: MPU as PTP master

Starting ptp4l program on PTP peer to be running as PTP slave clock.

- Provide the network interface with hardware time stamping capability with the "`-i`" option and replace the "`<network interface>`" with the actual network interface name, such as `eth1`, in the "`-i <network interface>`" option.
- Use the ethtool utility as mentioned above on PTP peer side to identify the PHC device number and replace the "X" in "`-p /dev/ptpX`" with it.

```
# ptp4l -i <network interface> -p /dev/ptpX -f /etc/ptp4l_cfg/gPTP.cfg -m -s
ptp4l[25687.144]: ioctl SIOCETHTOOL failed: Operation not supported
ptp4l[25687.144]: selected /dev/ptp1 as PTP clock
ptp4l[25687.162]: port 1: get_ts_info not supported
ptp4l[25687.208]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[25687.209]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[25690.716]: port 1: new foreign master 2ecbe2.fffe.7c9e22-1
ptp4l[25690.717]: selected best master clock 2ecbe2.fffe.7c9e22
ptp4l[25690.717]: port 1: assuming the grand master role
ptp4l[25690.717]: port 1: LISTENING to GRAND_MASTER on RS_GRAND_MASTER
```

Starting ptp4l program on DSA switch side to be running as PTP master clock.

```
# ptp4l -i hms0p2 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m
ptp4l[290529.407]: selected /dev/ptp1 as PTP clock
ptp4l[290529.444]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[290529.444]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[290532.815]: port 1: LISTENING to MASTER on
 ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[290532.815]: selected local clock 2ecbe2.fffe.7c9e22 as best master
ptp4l[290532.815]: port 1: assuming the grand master role
ptp4l[290537.820]: port 1: new foreign master 00049f.fffe.06fe8b-1
ptp4l[290537.820]: selected best master clock 00049f.fffe.06fe8b
ptp4l[290537.821]: port 1: MASTER to SLAVE on RS_SLAVE
ptp4l[290539.448]: rms 133343990721727 max 266687981443494 freq -36907 +/- 3430
 delay   365 +/-   0
ptp4l[290540.451]: rms  903 max 1395 freq -34021 +/- 1209 delay   365 +/-   0
ptp4l[290541.453]: rms 1496 max 1570 freq -31334 +/- 385 delay   364 +/-   0
ptp4l[290542.454]: rms  939 max 1242 freq -30825 +/-  33 delay   365 +/-   0
ptp4l[290543.456]: rms  296 max  487 freq -31125 +/- 106 delay   364 +/-   0
ptp4l[290544.458]: rms   58 max   92 freq -31448 +/-  73 delay   364 +/-   0
ptp4l[290545.458]: rms   88 max   98 freq -31596 +/-  24 delay   364 +/-   0
ptp4l[290546.459]: rms   51 max   70 freq -31618 +/-   6 delay   364 +/-   0
ptp4l[290547.459]: rms   19 max   36 freq -31606 +/-  10 delay   363 +/-   0
ptp4l[290548.460]: rms    8 max   18 freq -31589 +/-  12 delay   361 +/-   0
ptp4l[290549.462]: rms    9 max   16 freq -31577 +/-  10 delay   361 +/-   0
ptp4l[290550.464]: rms    6 max    8 freq -31573 +/-   5 delay   363 +/-   0
ptp4l[290551.465]: rms    5 max   10 freq -31576 +/-   7 delay   365 +/-   0
ptp4l[290552.467]: rms    7 max   13 freq -31583 +/-   8 delay   365 +/-   0
ptp4l[290553.468]: rms    6 max   11 freq -31583 +/-   8 delay   365 +/-   0
ptp4l[290554.468]: rms    6 max   11 freq -31572 +/-   5 delay   364 +/-   0
ptp4l[290555.469]: rms    8 max   14 freq -31570 +/-  10 delay   364 +/-   0
ptp4l[290556.469]: rms    7 max   13 freq -31574 +/-   9 delay   364 +/-   0
ptp4l[290557.470]: rms    6 max   10 freq -31573 +/-   8 delay   364 +/-   0
ptp4l[290558.471]: rms    6 max   12 freq -31570 +/-   8 delay   364 +/-   0
ptp4l[290559.472]: rms    5 max    9 freq -31569 +/-   7 delay   364 +/-   0
ptp4l[290560.472]: rms    4 max    7 freq -31576 +/-   3 delay   364 +/-   0
ptp4l[290561.473]: rms    5 max   11 freq -31577 +/-   6 delay   363 +/-   0
ptp4l[290562.473]: rms    6 max   12 freq -31569 +/-   7 delay   363 +/-   0
```

Scenario 2: MPU + i.MX RT1180 running as PTP slave clock

Start ptp4l program on PTP peer side to be running as PTP master clock.

- Providing the network interface with hardware time stamping capability with the "-i" option and replace the "<network interface>" with actual network interface name, such as eth1, in the "-i <network interface>" option.
- Use the ethtool utility as mentioned above on PTP peer side to identify the PHC device number and replace the "X" in "-p /dev/ptpX" with it.

```
# ptp4l -i <network interface> -p /dev/ptpX -f /etc/ptp4l_cfg/gPTP.cfg -m
ptp4l[290699.108]: selected /dev/ptp1 as PTP clock
ptp4l[290699.144]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[290699.144]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[290702.668]: port 1: LISTENING to MASTER on
 ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[290702.668]: selected local clock 2ecbe2.fffe.7c9e22 as best master
ptp4l[290702.668]: port 1: assuming the grand master role
```

Starting ptp4l program on MPU side to be running as PTP slave clock. DSA switch

```
# ptp4l -i hms0p2 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m -s
ptp4l[25866.504]: ioctl SIOCETHTOOL failed: Operation not supported
ptp4l[25866.504]: selected /dev/ptp1 as PTP clock
ptp4l[25866.536]: port 1: get_ts_info not supported
ptp4l[25866.580]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[25866.580]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[25870.400]: selected local clock 00049f.fffe.06fe8b as best master
ptp4l[25870.566]: port 1: new foreign master 2ecbe2.fffe.7c9e22-1
ptp4l[25870.566]: selected best master clock 2ecbe2.fffe.7c9e22
ptp4l[25870.566]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[25870.835]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[25871.587]: rms 1685 max 2235 freq  -1906 +/- 1164 delay   362 +/-   0
ptp4l[25872.588]: rms  311 max  477 freq   -893 +/- 421 delay   363 +/-   0
ptp4l[25873.589]: rms  525 max  552 freq    +60 +/- 139 delay   362 +/-   0
ptp4l[25874.585]: rms  326 max  437 freq   +233 +/-  11 delay   363 +/-   0
ptp4l[25875.591]: rms  107 max  179 freq   +135 +/-  42 delay   362 +/-   0
ptp4l[25876.592]: rms   19 max   28 freq    +20 +/-  23 delay   362 +/-   0
ptp4l[25877.592]: rms   31 max   35 freq    -31 +/-  12 delay   362 +/-   0
ptp4l[25878.592]: rms   18 max   31 freq    -39 +/-   7 delay   362 +/-   0
ptp4l[25879.592]: rms    6 max   10 freq    -30 +/-   6 delay   363 +/-   0
ptp4l[25880.593]: rms    4 max    6 freq    -25 +/-   6 delay   364 +/-   0
ptp4l[25881.591]: rms    3 max    6 freq    -27 +/-   4 delay   365 +/-   0
ptp4l[25882.593]: rms    4 max    6 freq    -23 +/-   4 delay   364 +/-   0
ptp4l[25883.594]: rms    4 max    5 freq    -29 +/-   4 delay   365 +/-   0
ptp4l[25884.599]: rms    4 max    7 freq    -22 +/-   4 delay   365 +/-   0
ptp4l[25885.592]: rms    4 max    6 freq    -27 +/-   5 delay   365 +/-   0
ptp4l[25886.597]: rms    3 max    7 freq    -25 +/-   5 delay   366 +/-   0
ptp4l[25887.597]: rms    5 max    7 freq    -27 +/-   7 delay   366 +/-   0
ptp4l[25888.598]: rms    3 max    5 freq    -26 +/-   4 delay   366 +/-   0
ptp4l[25889.602]: rms    5 max    9 freq    -22 +/-   6 delay   365 +/-   0
ptp4l[25890.601]: rms    3 max    6 freq    -26 +/-   3 delay   364 +/-   0
ptp4l[25891.605]: rms    5 max    9 freq    -22 +/-   7 delay   364 +/-   0
ptp4l[25892.604]: rms    5 max    7 freq    -31 +/-   3 delay   365 +/-   0
ptp4l[25893.606]: rms    7 max   13 freq    -18 +/-   6 delay   365 +/-   0
ptp4l[25894.608]: rms    4 max    7 freq    -26 +/-   5 delay   365 +/-   0
ptp4l[25895.604]: rms    5 max    9 freq    -24 +/-   7 delay   365 +/-   0
ptp4l[25896.602]: rms    5 max    9 freq    -27 +/-   7 delay   365 +/-   0
ptp4l[25897.609]: rms    5 max    7 freq    -28 +/-   6 delay   365 +/-   0
ptp4l[25898.611]: rms    5 max    9 freq    -25 +/-   7 delay   365 +/-   0
ptp4l[25899.602]: rms    3 max    6 freq    -22 +/-   4 delay   365 +/-   0
```

4.4.9 PTP Virtual clock and multi-domain usage

The **PTP Virtual Clock (`ptp_vclock`)** is a software-based clock mechanism in the Linux kernel that emulates hardware Precision Time Protocol (PTP) clocks. It enables high-precision time synchronization without physical hardware, using existing NICs to create isolated virtual clocks for multi-domain timing scenarios.

The `ptp_vclock` creates virtual `/dev/ptpN` devices (for example, `/dev/ptp3`) that mimic hardware PHCs (Physical Hardware Clocks). This method allows time-sensitive applications to operate in environments lacking dedicated timing hardware. Each virtual clock must bind to a physical PHC device. The physical PHC provides hardware timestamping capabilities.

Each virtual clock operates in a distinct **PTP domain** (for example, Domain 0 for control traffic, Domain 1 for data). This method prevents timing interference between domains.

The following is an example of using a virtual clock. The port hms0p2 uses a physical clock for clock synchronization, and hms0p1 and hms0p0 respectively use virtual clock for clock synchronization.

Hardware setup

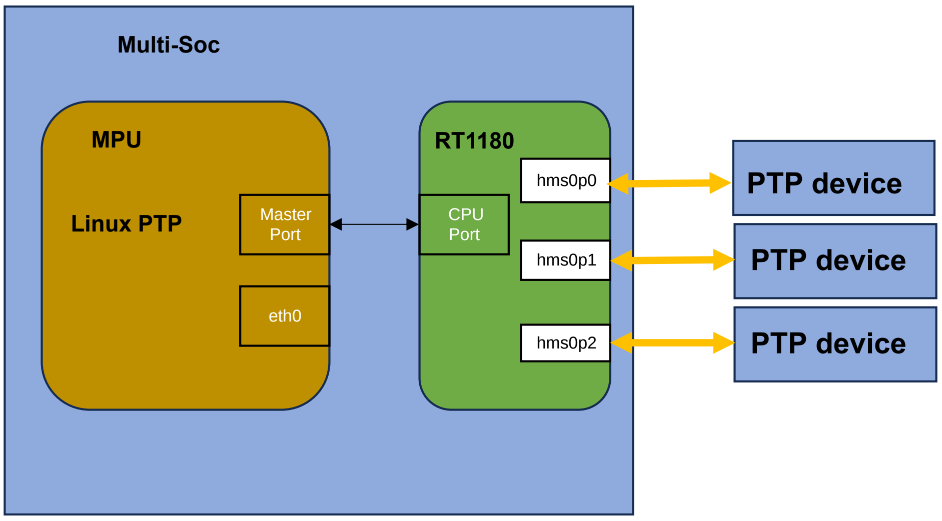


Figure 60. PTP Boards Connection

PTP stack only supports those i.MX RT1180 switch ports that are used as DSA CPU ports. To set up DSA, refer to Section 4.3

For PTP devices, any board such as LS1028ARDB, i.MX 93, or i.MX 8M Plus can be used. Select an Ethernet port that supports PTP to connect with one of i.MX RT1180 switch ports.

Get a PHC device

The ethtool utility can be used to identify the PHC (PTP hardware clock) device number used by Heterogeneous Multi-SoC DSA switch as shown below.

The PTP Hardware Clock value displayed by *ethtool* is the index of the PTP hardware clock. It corresponds to the naming of the `/dev/ptp*` devices. Here, the PTP Hardware Clock is 1.

```
~# ethtool -T hms0p2
Time stamping parameters for hms0p2:
Capabilities:
        hardware-transmit
        hardware-receive
        hardware-raw-clock
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
        off
        on
Hardware Receive Filter Modes:
        none
        ptpv2-l2-even
```

Create virtual clock

The Sysfs interface is used to create a virtual clock.

`/sys/class/ptp/ptp1/n_vclocks` : Controls the number of active virtual clocks tied to a physical PHC (e.g., `ptp1`)

The below command creates 2 virtual clocks `/dev/ptp2` and `/dev/ptp3` .

```
echo 2 > /sys/class/ptp/ptp1/n_vclocks
```

Running ptp4l on boards

1. Running ptp4l on hms0p2 with physical clock

   ```
   # ptp4l -i hms0p2 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m
   ```

2. Running ptp4l on hms0p1 with virtual clock

   ```
   # ptp4l -i hms0p1 -p /dev/ptp2 -f /etc/ptp4l_cfg/gPTP.cfg -m
   ```

3. Running ptp4l on hms0p0 with virtual clock

   ```
   # ptp4l -i hms0p0 -p /dev/ptp3 -f /etc/ptp4l_cfg/gPTP.cfg -m
   ```

Changing domain number

Each PTP device binds to a unique PTP domain(Domain 0-127). Domain number can be set to `/etc/ptp4l_cfg/gptp.cfg`. Add/modify domainNumber in the global section:

```
[global]
domainNumber 1    # Valid range: 0-127
```

Launch `ptp4l` with the Config File on hms0p1:

```
# ptp4l -i hms0p1 -p /dev/ptp2 -f /etc/ptp4l_cfg/gPTP.cfg -m
```

4.4.10 802.1CB usage

4.4.10.1 Network configuration

Use the below steps to configure the 802.1CB usage.

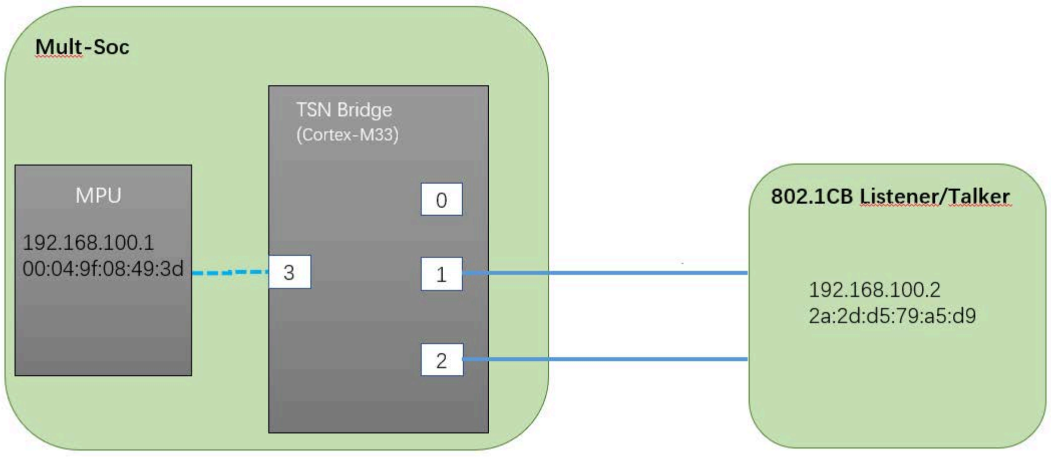


Figure 61. Network configuration for 802.1CB usage

1. Add switch ports into the bridge:

```
ip link add name br0 type bridge vlan_filtering 1
ip link set br0 up
ip link set hms0p0 master br0 && ip link set hms0p0 up
ip link set hms0p1 master br0 && ip link set hms0p1 up
ip link set hms0p2 master br0 && ip link set hms0p2 up
```

2. Connect `hms0p1` to the network, add IP on the `br0` and test the connection:

```
ifconfig br0 192.168.100.1
ping 192.168.100.2
```

3. Add destination MAC in FDB if the MAC is not learned:

```
bridge fdb add dev hms0p1 2a:2d:d5:79:a5:d9 vlan 1 master static
```

4. Drop streams to prevent loop back

```
tc qdisc add dev hms0p1 clsact
tc qdisc add dev hms0p2 clsact
tc filter add dev hms0p1 ingress protocol 802.1Q flower skip_sw \
src_mac 00:04:9f:08:49:3d vlan_id 0 action police mtu 1 \
conform-exceed drop/ok
tc filter add dev hms0p2 ingress protocol 802.1Q flower skip_sw \
src_mac 00:04:9f:08:49:3d vlan_id 0 action police mtu 1 \
conform-exceed drop/ok
tc filter add dev hms0p1 ingress protocol 802.1Q flower skip_sw \
src_mac 00:aa:bb:cc:dd:11 vlan_id 0 action police mtu 1 \
conform-exceed drop/ok
tc filter add dev hms0p2 ingress protocol 802.1Q flower skip_sw \
src_mac 00:aa:bb:cc:dd:11 vlan_id 0 action police mtu 1 \
conform-exceed drop/ok
```

   Note: *"`00:04:9f:08:49:3d`" is the local bridge MAC on Linux. "`00:aa:bb:cc:dd:11`" is the local pseudo MAC on i.MX RT1180.*

5. Connect hms0p2 to the network to obtain redundancy paths.

4.4.10.2 802.1CB sequence generator

1. Add 802.1CB sequence generator:

```
tc qdisc add dev hms0p1 clsact
tc filter add dev hms0p1 egress protocol 802.1Q flower skip_sw \
dst_mac 2a:2d:d5:79:a5:d9 vlan_id 1 action frer rtag tag-action tag-push
```

2. Split the stream to hms0p2

```
tc filter add dev hms0p1 egress protocol 802.1Q flower skip_sw \
dst_mac 2a:2d:d5:79:a5:d9 vlan_id 1 action mirred egress mirror dev hms0p2
```

3. Check the R-tag in listener side # tcpdump -i eno0

```
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eno0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
00:31:27.536183 00:04:9f:08:49:3d (oui Freescale) > 2a:2d:d5:79:a5:d9 (oui
 Unknown), ethertype Unknown (0xf1c1), length 104:
    0x0000:  0000 0000 0800 4500 0054 4f2a 4000 4001  ......E..TO*@.@.
    0x0010:  a229 c0a8 6401 c0a8 6403 0800 7a3c 0003  .)..d...d...z<..
    0x0020:  0001 bf55 a866 0000 0000 4a30 0d00 0000  ...U.f....J0....
    0x0030:  0000 1011 1213 1415 1617 1819 1a1b 1c1d  ................
    0x0040: 1e1f 2021 2223 2425 2627 2829 2a2b 2c2d ...!"#$%&'()*+,-
    0x0050:  2e2f 3031 3233 3435 3637                 ./01234567
00:31:28.556234 00:04:9f:08:49:3d (oui Freescale) > 2a:2d:d5:79:a5:d9 (oui
 Unknown), ethertype Unknown (0xf1c1), length 104:
    0x0000:  0000 0001 0800 4500 0054 4fe6 4000 4001  ......E..TO.@.@.
    0x0010:  a16d c0a8 6401 c0a8 6403 0800 8cec 0003  .m..d...d.......
    0x0020:  0002 c055 a866 0000 0000 367f 0d00 0000  ...U.f....6.....
    0x0030:  0000 1011 1213 1415 1617 1819 1a1b 1c1d  ................
    0x0040: 1e1f 2021 2223 2425 2627 2829 2a2b 2c2d   ...!"#$%&'()*+,-
    0x0050:  2e2f 3031 3233 3435 3637                 ./01234567
```

4. Drop streams to prevent loop back

```
tc qdisc add dev hms0p1 clsact
tc qdisc add dev hms0p2 clsact
tc filter add dev hms0p1 ingress protocol 802.1Q flower skip_sw \
src_mac 00:04:9f:08:49:3d vlan_id 0 action police mtu 1 \
conform-exceed drop/ok
tc filter add dev hms0p2 ingress protocol 802.1Q flower skip_sw \
src_mac 00:04:9f:08:49:3d vlan_id 0 action police mtu 1 \
conform-exceed drop/ok
tc filter add dev hms0p1 ingress protocol 802.1Q flower skip_sw \
src_mac 00:aa:bb:cc:dd:11 vlan_id 0 action police mtu 1 \
conform-exceed drop/ok
tc filter add dev hms0p2 ingress protocol 802.1Q flower skip_sw \
src_mac 00:aa:bb:cc:dd:11 vlan_id 0 action police mtu 1 \
conform-exceed drop/ok
```

5. Connect hms0p2 to the network to obtain redundancy paths.

4.4.10.3 802.1CB sequence recovery

1. Add 802.1CB sequence recovery:

```
tc filter add dev hms0p1 ingress protocol 802.1Q flower skip_sw \
dst_mac 00:04:9f:08:49:3d vlan_id 0 action frer rtag recover alg \
vector history-length 32 reset-time 10000 tag-action tag-pop
```

2. Ping on the remote terminal and check the packets received on MPU side:

```
# tcpdump -i br0
04:04:01.030771 IP aontst11.nl-nym01.nxp.com > imx93evk: ICMP echo request,
 id 51345, seq 2, length 64
04:04:01.030873 IP imx93evk > aontst11.nl-nym01.nxp.com: ICMP echo reply, id
 51345, seq 2, length 64
04:04:02.054815 IP aontst11.nl-nym01.nxp.com > imx93evk: ICMP echo request,
 id 51345, seq 3, length 64
04:04:02.054914 IP imx93evk > aontst11.nl-nym01.nxp.com: ICMP echo reply, id
 51345, seq 3, length 64
```

4.4.11 802.1 Qci usage

4.4.11.1 Network configuration

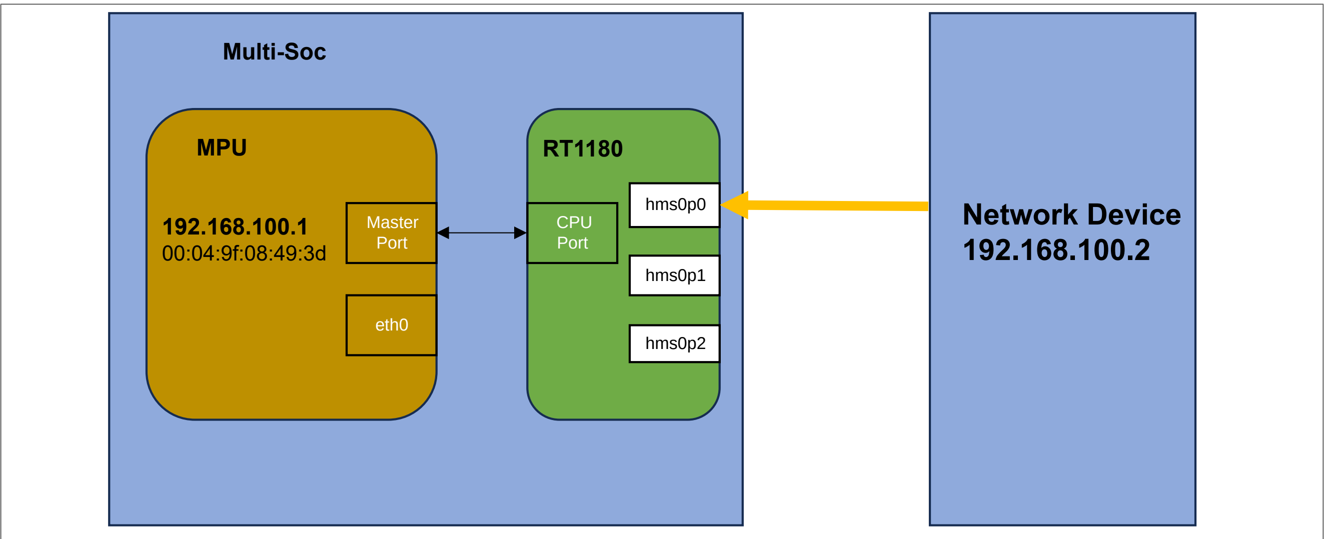


Figure 62. 802.1 Qci usage

Bridge mode

```
ip link add name br0 type bridge vlan_filtering 1
ip link set br0 up
ip link set hms0p0 master br0 && ip link set hms0p0 up
ip link set hms0p1 master br0 && ip link set hms0p1 up
ip link set hms0p2 master br0 && ip link set hms0p2 up
```

Standalone mode

```
ifconfig hms0p0 up
```

4.4.11.2 802.1 Qci configuration

1. **Set Qci gate**

```
tc qdisc add dev hms0p0 clsact
tc filter add dev hms0p0 ingress protocol 802.1Q flower skip_sw \
dst_mac 00:04:9f:08:49:3d vlan_id 0 action gate index 1 \
base-time 0 priority 3 sched-entry OPEN 6000 -1 -1 \
sched-entry CLOSE 6000 -1 -1
```

2. **Set Qci flow meter**

```
tc filter add dev hms0p0 ingress protocol 802.1Q flower skip_sw \
dst_mac 00:04:9f:08:49:3d vlan_id 0 action police index 1 \
rate 10Mbit burst 10000 conform-exceed drop/ok
```

3. **Set Qci SFI priority**

```
tc filter add dev hms0p0 ingress protocol 802.1Q flower skip_sw \
dst_mac 00:04:9f:08:49:3d vlan_id 0 vlan_prio 1 action gate \
index 1 base-time 0 priority 3 sched-entry CLOSE 6000 -1 -1
```

4. **Set Qci SFI maxsdu**

```
tc filter add dev hms0p0 ingress protocol 802.1Q flower skip_sw \
dst_mac 00:04:9f:08:49:3d vlan_id 0 vlan_prio 1 action police \
mtu 1000 conform-exceed drop/ok
```

5. **Set both gate and flow meter**

```
tc filter add dev hms0p0 ingress protocol 802.1Q flower skip_sw \
dst_mac 00:04:9f:08:49:3d vlan_id 0 action gate index 1 \
base-time 0 priority 3 sched-entry OPEN 6000 -1 -1 action police \
index 1 rate 10Mbit burst 10000 conform-exceed drop/ok
```

6. Show the Qci rules configured on hms0p0:

```
tc -s filter show dev hms0p0 ingress
```

7. Delete one of the Qci rule:

```
tc filter del dev hms0p0 ingress pref 49152
```

4.4.12 802.1 Qbv usage

4.4.12.1 Network configuration

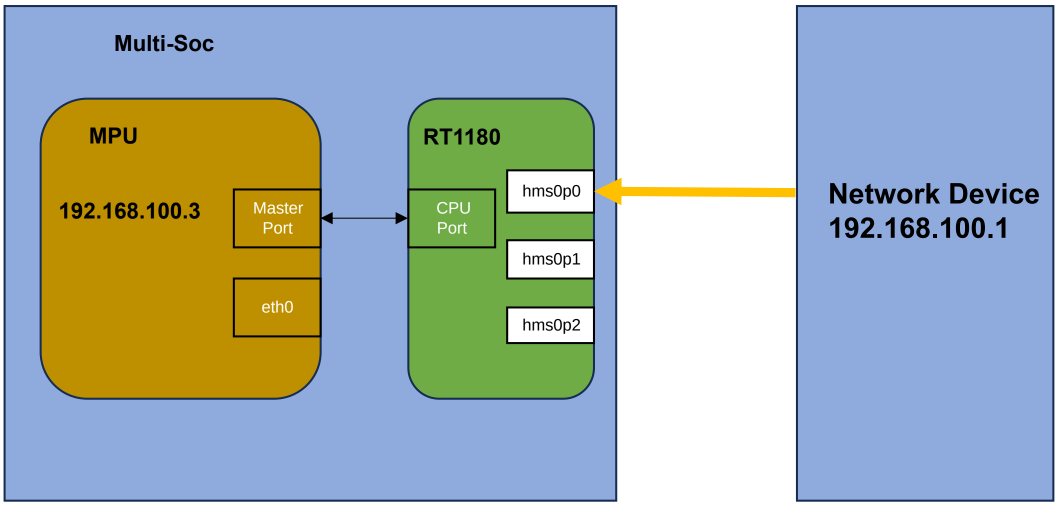


Figure 63. Network configuration for 802.1 Qbv usage

TSN 802.1 Qbv is an IEEE specification that allows to define transmission time slots for networking traffic queues. It adds enhancements to traffic scheduling with the introduction of time-aware shaper. For DSA switch, both bridge mode and single port mode support TSN 802.1 Qbv setting. The following Qbv configuration example takes the network card hms0p0 as an example. Refer to the following command to select a work mode for hms0p0 - bridge mode or single port mode.

Bridge mode

```
ip link add name br0 type bridge vlan_filtering 1
ip link set br0 up
ip link set hms0p0 master br0 && ip link set hms0p0 up
ip link set hms0p1 master br0 && ip link set hms0p1 up
ip link set hms0p2 master br0 && ip link set hms0p2 up
ifconfig br0 192.168.100.3 up
```

Single port mode

```
ifconfig hms0p0 192.168.100.3 up
```

4.4.12.2 Configuring 802.1 Qbv via tc

1. **Open the gate**

   ```
   tc qdisc replace dev hms0p0 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 sched-entry S 01 300000 flags 0x2
   ```

2. **Ping remote IP successfully**

   ```
   ping 192.168.100.1
   ```

3. **Close the gate**

   ```
   tc qdisc replace dev hms0p0 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 fe 300000 flags 0x2
   ```

4. **ping remote IP failed**

   ```
   ping 192.168.100.1
   ```

   Disable Qbv configuration

   ```
   tc qdisc del dev hms0p0 parent root handle 100
   ```

4.4.12.3 Configuring 802.1 Qbv via netconf

1. For installing Netopeer2-cli on Ubuntu22.04, refer to "Section 6.4.2.2".
2. Create Qbv configuration file `qbv.xml`

   ```
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
       xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
       xmlns:dot1q="urn:ieee:std:802.1Q:yang:ieee802-dot1q-bridge"
       xmlns:sched="urn:ieee:std:802.1Q:yang:ieee802-dot1q-sched"
       xmlns:sched-bridge="urn:ieee:std:802.1Q:yang:ieee802-dot1q-sched-bridge">
   <interface>
   ```

```
    <name>hms0p0</name>
    <enabled>true</enabled>
    <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
type">ianaift:ethernetCsmacd</type>
    <dot1q:bridge-port>
        <sched-bridge:gate-parameter-table>
            <sched-bridge:queue-max-sdu-table>
                <sched-bridge:traffic-class>0</sched-bridge:traffic-class>
                <sched-bridge:queue-max-sdu>1024</sched-bridge:queue-max-sdu>
            </sched-bridge:queue-max-sdu-table>
            <sched-bridge:gate-enabled>true</sched-bridge:gate-enabled>
            <sched-bridge:admin-gate-states>127</sched-bridge:admin-gate-
states>
            <sched-bridge:admin-control-list>
                <sched-bridge:gate-control-entry>
                    <sched-bridge:index>0</sched-bridge:index>
                    <sched-bridge:operation-name>sched:set-gate-states</
sched-bridge:operation-name>
                    <sched-bridge:gate-states-value>5</sched-bridge:gate-
states-value>
                    <sched-bridge:time-interval-value>400000</sched-
bridge:time-interval-value>
                </sched-bridge:gate-control-entry>
                <sched-bridge:gate-control-entry>
                    <sched-bridge:index>1</sched-bridge:index>
                    <sched-bridge:operation-name>sched:set-gate-states</
sched-bridge:operation-name>
                    <sched-bridge:gate-states-value>7</sched-bridge:gate-
states-value>
                    <sched-bridge:time-interval-value>300000</sched-
bridge:time-interval-value>
                </sched-bridge:gate-control-entry>
                <sched-bridge:gate-control-entry>
                    <sched-bridge:index>2</sched-bridge:index>
                    <sched-bridge:operation-name>sched:set-gate-states</
sched-bridge:operation-name>
                    <sched-bridge:gate-states-value>7</sched-bridge:gate-
states-value>
                    <sched-bridge:time-interval-value>300000</sched-
bridge:time-interval-value>
                </sched-bridge:gate-control-entry>
            </sched-bridge:admin-control-list>
            <sched-bridge:config-change>true</sched-bridge:config-change>
            <sched-bridge:admin-cycle-time>
                <sched-bridge:numerator>1</sched-bridge:numerator>
                <sched-bridge:denominator>10000</sched-bridge:denominator>
            </sched-bridge:admin-cycle-time>
            <sched-bridge:admin-base-time>
                <sched-bridge:seconds>0</sched-bridge:seconds>
                <sched-bridge:nanoseconds>5000</sched-bridge:nanoseconds>
            </sched-bridge:admin-base-time>
            <sched-bridge:supported-list-max>50</sched-bridge:supported-list-
max>
            <sched-bridge:supported-cycle-max>
                <sched-bridge:numerator>1</sched-bridge:numerator>
                <sched-bridge:denominator>1</sched-bridge:denominator>
            </sched-bridge:supported-cycle-max>
            <sched-bridge:supported-interval-max>1000000000</sched-
bridge:supported-interval-max>
        </sched-bridge:gate-parameter-table>
    </dot1q:bridge-port>
            </interface>
            </interfaces>
```

3. **Configure Qbv by netopeer2-cli**

```
$ netopeer2-cli
> connect --login root --host 192.168.1.1
> edit-config --target running --config=qbv.xml
```

4.4.13 802.1 Qbu usage

4.4.13.1 Network configuration

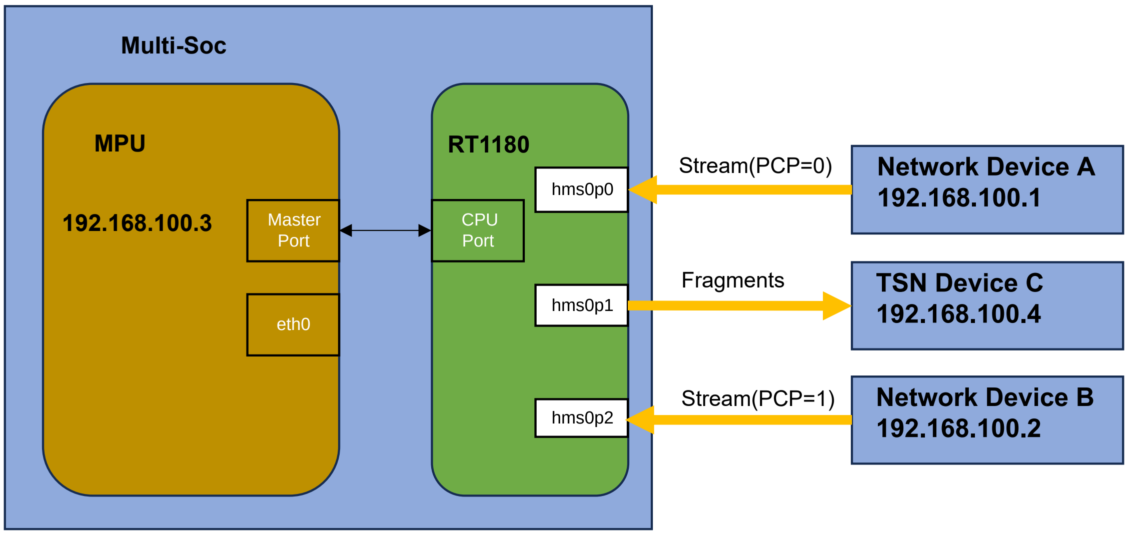


Figure 64. Qbu usage

1. **Bridge mode**

```
ip link add name br0 type bridge vlan_filtering 1
ip link set br0 up
ip link set hms0p0 master br0 && ip link set hms0p0 up
ip link set hms0p1 master br0 && ip link set hms0p1 up
ip link set hms0p2 master br0 && ip link set hms0p2 up
ifconfig br0 192.168.100.3 up
```

4.4.13.2 Qbu configuration

1. **Enable pmac**

```
ethtool --set-mm hms0p0 tx-enabled on pmac-enabled on verify-enabled off \
tx-min-frag-size 60
ethtool --set-mm hms0p1 tx-enabled on pmac-enabled on verify-enabled off \
tx-min-frag-size 60
ethtool --set-mm hms0p2 tx-enabled on pmac-enabled on verify-enabled off \
tx-min-frag-size 60
```

2. **Configure preemption queues**

```
tc qdisc add dev hms0p1 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 fp E P E E E E P E hw 1
```

3. **Send packets from TSN network**
   Send priority 0 and priority 1 traffic on hms0p0 and hms0p2. The preemption queue is then preempted, and fragment packets are obtained on hms0p1.

4.4.14 HSR usage

This section describes High-availability Seamless Redundancy (HSR) implementation, network configuration, and testing.

4.4.14.1 Network configuration

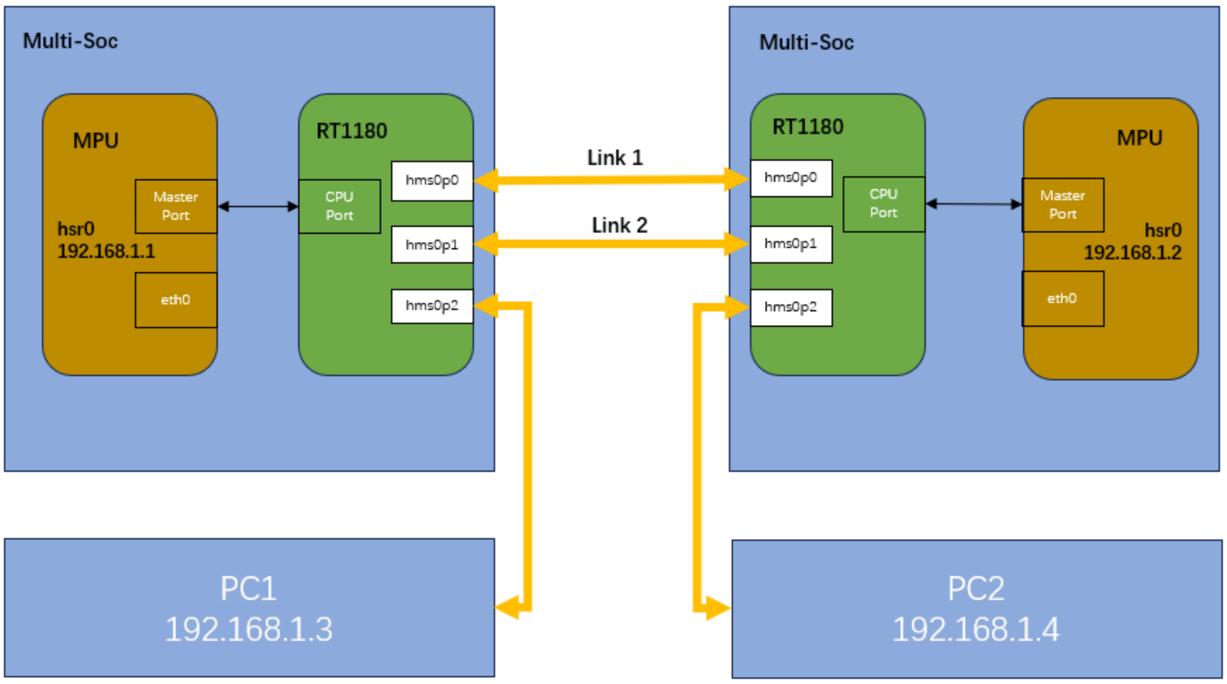


Figure 65. Network configuration for HSR

1. Enable the switch ports using the commands below:

```
ip link set hms0p0 up
ip link set hms0p1 up
ip link set hms0p2 up
```

2. Enable HSR offload for both interfaces:

```
ethtool -K hms0p0/hms0p1 hsr-fwd-offload on ethtool -K hms0p0/hms0p1 hsr-dup-
offload on ethtool -K hms0p0/hms0p1 hsr-tag-ins-offload on ethtool -K hms0p0/
hms0p1 hsr-tag-rm-offload on
```

3. Create the HSR interface and add slave interfaces to it.

```
ip link add name hsr0 type hsr slave1 hms0p0 slave2 hms0p1 supervision 45
 version 1
```

4. Add an IP address to the HSR interface:

```
On board A:
ifconfig hsr0 192.168.1.1
```

```
ip link set hsr0 up
On board B:
ifconfig hsr0 192.168.1.2
ip link set hsr0 up
```

4.4.14.2 HSR network test

1. Use the ‘`ping`’ command to test the connection:

   ```
   On board A:
   ping 192.168.1.2
   ```

2. Ping command runs successfully when link 1 and link 2 are connected. After the connection, `ping` command runs successfully even after disconnecting link 1 or link 2.
3. Use `iperf` to test the network

   ```
   On board B:
   iperf3 -s
   On board A:
   iperf3 -c 192.168.1.2
   ```

   When one of the links is down or up, there is no significant change in bandwidth.
4. Use `ping` to test the `RedBox` of connection as follows:

   ```
   On PC1:
   ping 192.168.1.4
   ```

   The `ping` command runs successfully when link 1 and link 2 are connected. After the connection, `ping` command runs successfully even after disconnecting link 1 or link 2.

5 Real-time networking

5.1 Time Sensitive Networking (TSN) on NXP platforms

Time Sensitive Networking (TSN) is an extension to traditional Ethernet networks, providing a set of standards compatible with IEEE 802.1 and 802.3. These extensions intend to address the limitations of standard Ethernet in sectors ranging from industrial and automotive applications to live audio and video systems. Applications running over traditional Ethernet must be designed to be very robust in order to withstand corner cases such as packet loss, delay, or even reordering. TSN aims to provide guarantees for deterministic latency and packet loss under congestion. Therefore, it allows critical and non-critical traffic to be converged in the same network.

This chapter describes the process and use cases for implementing TSN features on the i.MX 8M LPDDR4 Plus, i.MX 8DXL LPDDR4 EVK, i.MX 93 EVK, i.MX 93 9x9 QSB, i.MX 93 14x14 EVK, and LS1028ARDB boards.

5.1.1 TSN hardware capability

Table 48. 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 (LS1028 A)	Y	Y	Y	Y	N	Y
Felix switch (LS1028A)	Y	Y	Y	Y	Y	Y
Stmac (i.MX 8DXL, i.MX 8M Plus, i.MX 93)	Y	Y	Y	Y	N	N

5.1.2 TSN configuration

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

Table 49. TSN configuration tool support on different hardware 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
Stmac (i.MX 8DXL, i.MX 8M Plus, i.MX 93)	tc-taprio	ethtool	tc-cbs	ptp4l, Gen AVB/TSN stack	N/A	N/A

5.1.2.1 Using Linux traffic control (tc) to configure TSN

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 LS1028A platform, ENETC QoS driver must 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

5.1.2.2 Using Tsntool to configure TSN

Tsntool is a tool to set the TSN capability of the Ethernet ports of TSN Endpoint and TSN switch. It is used on the LS1028A platform. 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 the kernel to use it for testing using the commands below:

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

Refer to the Section 5.1.2.2.1 for the details.

5.1.2.2.1 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.*

5.1.2.2.1.1 Getting the source code

Github of the tsntool code is mentioned below.

https://github.com/nxp-qoriq/tsntool

5.1.2.2.1.2 Tsn tool commands

The Table 50 lists the TSN tool commands and their description.

Table 50. 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

Table 50. TSN tool commands and their description*...continued*

Command	Description
qcisficounterget	Gets stream filter counters
qcifmiset	Sets flow metering instance
qcifmiget	Gets flow metering instance
cbsset	Sets TCs credit-based shaper configuration
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

5.1.2.2.1.3 Tsntool commands and parameters

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

Table 51 lists the qbvset parameter and its arguments.

Table 51. 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.

Table 51. qbvset*...continued*

Parameter <argument>	Description
	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 52. qbvget

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

Table 53. 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.
`--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`

Table 53. cbstreamidset *...continued*

Parameter <argument>	Description
`--nulltagged <value>`	`tsnCpeNullDownTagged`
`--nullvid <value>`	`tsnCpeNullDownVlan`
`--sourcemac <value>`	`tsnCpeSmacVlanDownSrcMac`
`--sourcetagged <value>`	`tsnCpeSmacVlanDownTagged`
`--sourcevid <value>`	`tsnCpeSmacVlanDownVlan`

Table 54. cbstreamidget

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

Table 55. 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.
`--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 56. qcisfiget

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

Table 57. 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.
`--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 58. qcisgiget

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

Table 59. 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.

Table 59. qcifmiset *...continued*

Parameter <argument>	Description
`--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 60. 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 61. 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 62. 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 63. cbsget

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

Table 64. 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 65. 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 66. 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.
`--iport_mask <value>`	INPUT_PORT_MASK: If the packet is from input port belonging to this port mask, then it is 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 67. 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 68. 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 69. qcicapget

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

Table 70. tsncapget

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

5.1.2.2.1.4 Input tips

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

- When a user inputs a command, use the **TAB** key to help list the related commands.
  For example:
  ```
  tsntool> qbv
  ```
  Then press **TAB** key, to get all related `qbv*` start commands.
  If there is only one choice, it is filled as the whole command automatically.
- When you want to input parameters and do not remember the parameter name, you can just input “--”. Then pressing **TAB** key displays all the parameters.
- If you input only half the parameter’s name, pressing the **TAB** key lists all the related names.
- **History**: Press the up arrow “↑” . User gets the command history and can re-use the command.

5.1.2.2.1.5 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
```

5.1.2.3 Using NETCONF to configure TSN remotely

1. **Overview**

The NETCONF protocol defines a mechanism for device management and configuration retrieval and modification. It enables a client to adjust to the specific features of any network equipment by using a remote procedure call (RPC) paradigm and a system to expose device (server) capabilities.

YANG is a standards-based, extensible, hierarchical data modeling language. YANG is used to model the configuration and state data used by NETCONF operations, RPCs, and server event notifications.

2. **Support for different platforms in Real-time Edge**

Table 72. Real-time Edge platform support

TSN offload	Real-time Edge		
	LS1028A		i.MX 8DXL / i.MX 8M Plus / i.MX 93
	`libtsn`	`tc`	`tc`
802.1Qbv (Time Aware Shaper)	Y	Y	Y
802.1Qbu/802.3br (Frame Preemption)	Y	Y	Y
802.1Qav (Credit Based Shaper)	-	-	-
802.1CB (Frame Replication and Elimination for Reliability)	-	-	N/A
802.1Qci (Per-Stream Filtering and Policing)	Y	Y	N/A
IP config	Y	Y	Y
MAC config	Y	Y	Y
VLAN config	Y	Y	Y

3. **NETCONF tools Installation and configuration**

The command `netopeer2-cli` can be used as a NETCONF tool, which is present in the Netopeer2 software package. Refer to Section 6.4.2 for more information. The example XML configuration files can be found in the `Instances` directory of the git repo (`https://github.com/nxp-real-time-edge-sw/real-time-edge-sysrepo.git`), including Qbv, Qbu, Qci, and stream identification via network.

- For Netopeer2 installation, refer to Section 6.4.2.2.
- For the usage of netopeer2-cli, refer to Section 6.4.3.3.
- For a configuration example, refer to Section 6.4.3.7.

5.1.2.4 TSN configuration demo

In this demo, a network administrator can configure and monitor the TSN network on a web page. It can manage multiple TSN devices. Before starting the demo, set up the device network as shown in the Figure 66.

This section describes the steps for remote configuration.

- **Overview**

The **Web UI** allows the remote control of the YANG model. The user can connect to the http server and input TSN parameters on the web UI.

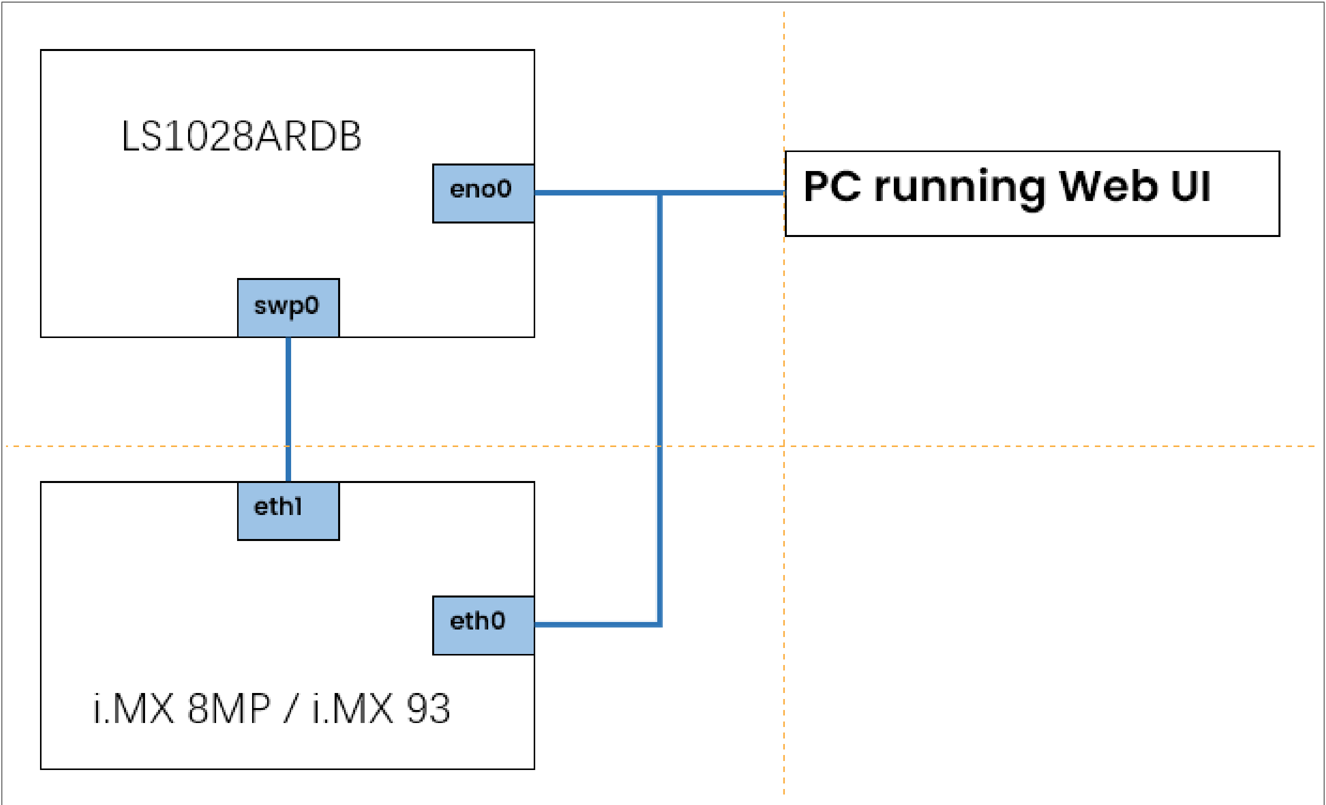


Figure 66. A sample setup for remote configuration

5.1.2.4.1 Software setup

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/`.

Set up a web server on the Ubuntu PC that is connected to the TSN network. The following steps show how to set up the software and configure devices:

1. Install dependent packages on Ubuntu 22.04.
   For Ubuntu:

```
$ sudo apt update
$ sudo apt install -y libtool libtool-bin libffi-dev \
libxslt1-dev libcurl4-openssl-dev xsltproc zlib1g-dev \
libssl-dev libaugeas-dev libreadline-dev \
pkg-config libxml2-dev cmake openssh-server \
libnss-mdns avahi-utils sphinx-doc sphinx-common

$ sudo apt install -y python3-sphinx python3-setuptools python3-libxml2
$ sudo apt install -y python3-pip python3-dev python3-flask python3-pexpect

$ sudo pip3 install flask-restful websockets argparse
```

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. However, libnetconf requires a version of 0.7.3 or later. Remove the default version and reinstall it by downloading the source code and installing it manually.*

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

```
git clone https://github.com/nxp-qoriq/tsntool
cd tsntool/demos/cnc/
```

5. Install libnetconf:
   In the below command segments,
   - PATH-to-tsntool is the path to the tsntool source code.

```
$ git clone https://github.com/CESNET/libnetconf.git
$ cd libnetconf
$ git checkout 8e934324e4b1e0ba6077b537e55636e1d7c85aed -b fornetconf1
$ cp PATH-to-tsntool/demos/cnc/0003-update-the-hostkeys-of-ssh-session-
connection.patch .
$ git am 0003-update-the-hostkeys-of-ssh-session-connection.patch
$ autoreconf --force --install
$ ./configure
$ make
$ sudo make install
```

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
$ cd PATH-to-libnetconf/python
$ python3 setup.py build; sudo python3 setup.py install
```

   Note:
   If rebuilding the python library, user must remove the `build` folder by using the command `rm build -rf` before rebuilding. On the boards that Real-time Edge software 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 the topoagent server on the boards supported
   - Ensure that the netopeer2-server run at boards (this step is not necessary for topology discovery).
   - Ensure that the lldpd daemon is running on the boards.
   - Ensure that the avahi-daemon is running on the boards.
     – Start the topology server on the LS1028ardb board (execute the following commands on the device console):

     ```
     ifconfig eno2 up
     ip link add name switch type bridge vlan_filtering 1
     ip link set switch up
     ip link set swp0 master switch && ip link set swp0 up
     ip link set swp1 master switch && ip link set swp1 up
     ip link set swp2 master switch && ip link set swp2 up
     ip link set swp3 master switch && ip link set swp3 up

     #Stop lldpd service:
     pkill lldpd

     #Start lldpd and limit interfaces to use. Use all ports except the control
      port.
     lldpd -I swp0,swp1,swp2,swp3

     # Set the host name as below:

     avahi-set-host-name real-time-edge-ls1028ardb
     cd /usr/share/samples/cncdemo/python3 topoagent.py
     ```

     – Start the topology server on the i.MX 8M Plus or the i.MX 93 board:

     ```
     #Stop lldpd service.
     pkill lldpd
     #Start lldpd and limit interfaces to use.. Use all ports except the
      control port.
     lldpd -I eth1
     #If the hostname is not real-time-edge-$boardname, change to real-time-
     edge-$boardname.
     avahi-set-host-name real-time-edge-imx93evk
     cd /usr/share/samples/cncdemo/python3 topoagent.py
     ```

9. Use the web browser to track the topology and configuration of the devices. Input the IP of the web server with the port 8180 at the browser. For example:

   ```
   http://10.193.20.147:8180
   ```

   Note:
   TSN configuration debug:
   - *Users can 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 on the board or not.*

   Limitations of Web UI are:
   - *The server setup on a Ubuntu PC could be more compatible.*
   - *Supports Qbv, Qbu, and Qci in current version.*
   - *For Qci setting, Stream-gate entry must 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.*

5.1.2.4.2 Remote configuration

This section describes the steps for remote configuration.

- **Overview**
  The **Web UI** allows the remote control of the YANG model. The user can connect to the http server and input TSN parameters on the web UI. Click "**Yes, confirm**" button to send the parameters to the board as shown in Figure 67.

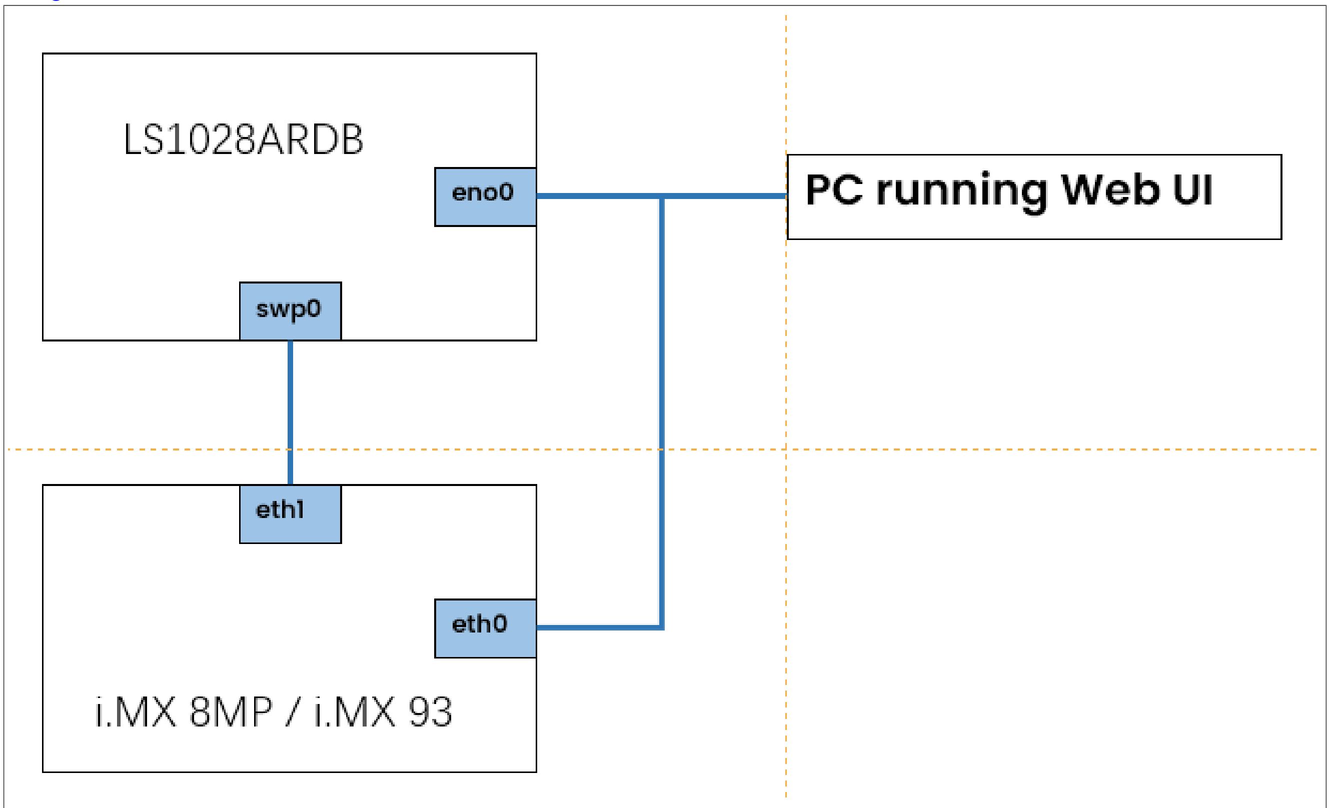
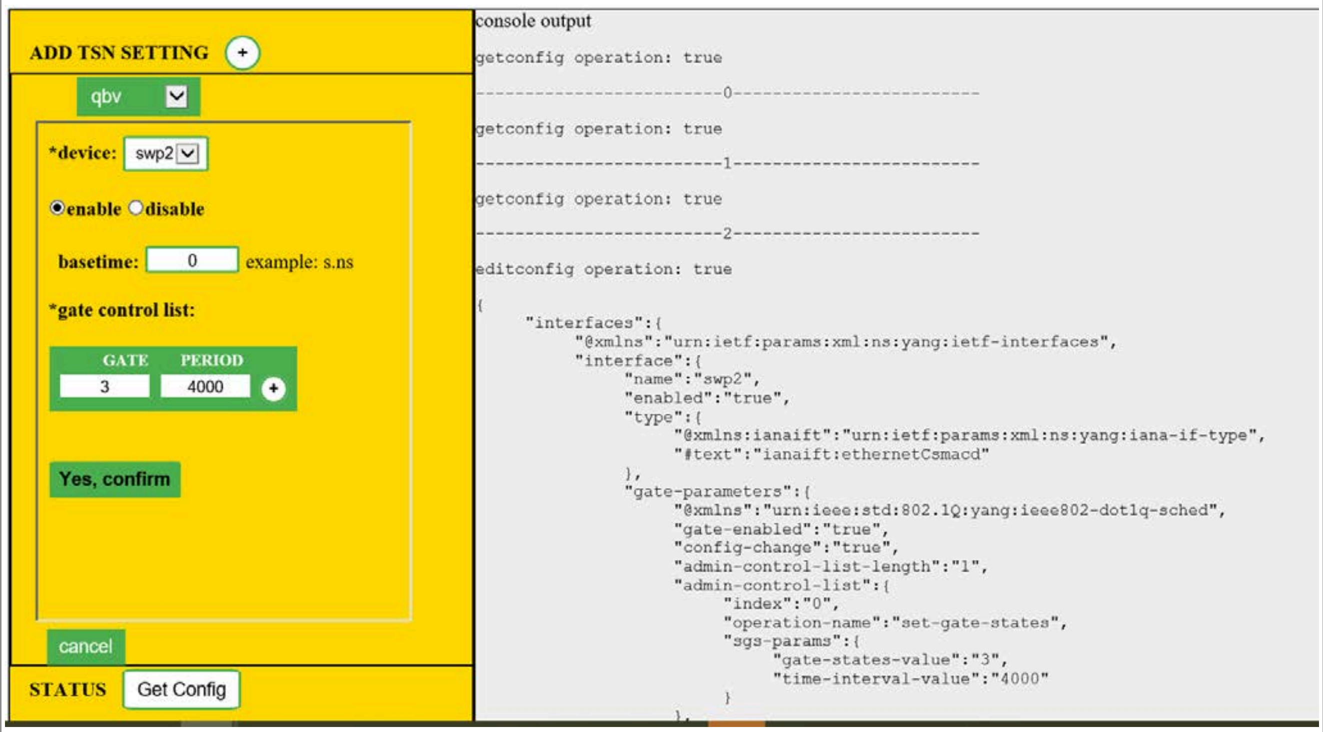


Figure 67. A sample setup for remote configuration

- **User Interface**
  Click the device displayed on the home page, and an interface description table appears. Click the interface to jump to the configuration page.
- **Qbv Configuration**: Selecting '`qvb`' option setting displays the options as shown in Figure 68.



qbv

*device: swp2

[✓ enable] disable

basetime: 0 example: s.ns

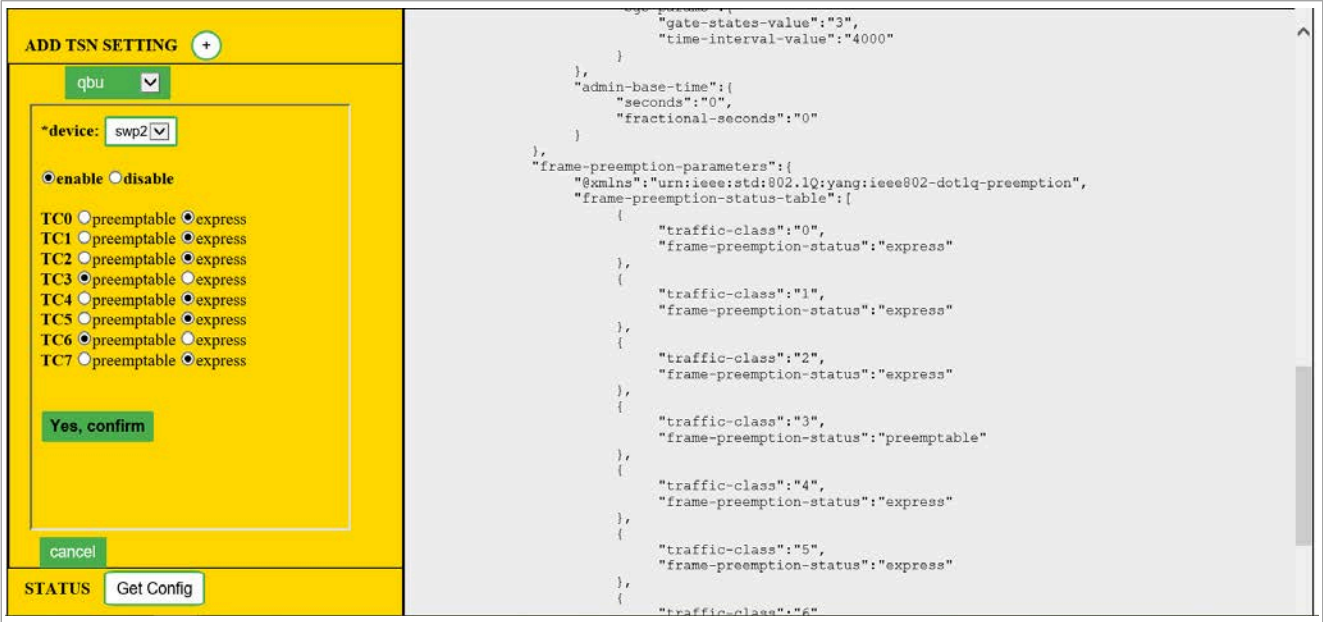
*gate control list:

GATE	PERIOD
3	4000

```
console output
getconfig operation: true
-------------------------0-------------------------
getconfig operation: true
-------------------------1-------------------------
getconfig operation: true
-------------------------2-------------------------
editconfig operation: true
{
     "interfaces":{
          "@xmlns":"urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "interface":{
               "name":"swp2",
               "enabled":"true",
               "type":{
                    "@xmlns:ianaift":"urn:ietf:params:xml:ns:yang:iana-if-type",
                    "#text":"ianaift:ethernetCsmacd"
               },
               "gate-parameters":{
                    "@xmlns":"urn:ieee:std:802.1Q:yang:ieee802-dot1q-sched",
                    "gate-enabled":"true",
                    "config-change":"true",
                    "admin-control-list-length":"1",
                    "admin-control-list":{
                         "index":"0",
                         "operation-name":"set-gate-states",
                         "sgs-params":{
                              "gate-states-value":"3",
                              "time-interval-value":"4000"
                         }
```

Figure 68. Qbv Configuration

- **Qbu Configuration**: Selecting '**`qbu`**' option setting displays the options as shown in Figure 69.



qbu

*device: swp2

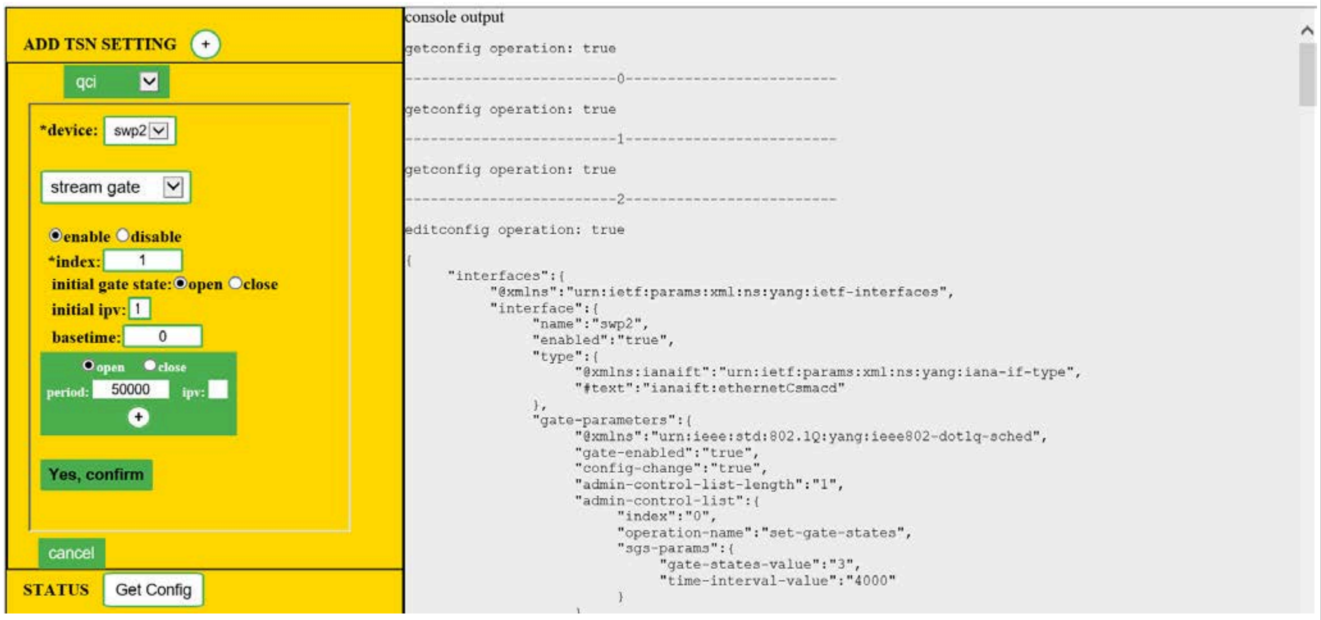
[✓ enable] disable

TC0 preemptable [✓ express]
TC1 preemptable [✓ express]
TC2 preemptable [✓ express]
TC3 [✓ preemptable] express
TC4 preemptable [✓ express]
TC5 preemptable [✓ express]
TC6 [✓ preemptable] express
TC7 preemptable [✓ express]

```
                              "gate-states-value":"3",
                              "time-interval-value":"4000"
                         }
                    },
                    "admin-base-time":{
                         "seconds":"0",
                         "fractional-seconds":"0"
                    }
               },
               "frame-preemption-parameters":{
                    "@xmlns":"urn:ieee:std:802.1Q:yang:ieee802-dot1q-preemption",
                    "frame-preemption-status-table":[
                         {
                              "traffic-class":"0",
                              "frame-preemption-status":"express"
                         },
                         {
                              "traffic-class":"1",
                              "frame-preemption-status":"express"
                         },
                         {
                              "traffic-class":"2",
                              "frame-preemption-status":"express"
                         },
                         {
                              "traffic-class":"3",
                              "frame-preemption-status":"preemptable"
                         },
                         {
                              "traffic-class":"4",
                              "frame-preemption-status":"express"
                         },
                         {
                              "traffic-class":"5",
                              "frame-preemption-status":"express"
                         },
                         {
                              "traffic-class":"6"
```

Figure 69. Qbu configuration

- **Qci Configuration**: Selecting '**`qci`**' option setting displays the options as shown in Figure 70.



*device: swp2
stream gate
[✓ enable] disable
*index: 1
initial gate state: [✓ open] close
initial ipv: 1
basetime: 0
[✓ open] close
period: 50000 ipv:

```
console output

getconfig operation: true

-------------------------0-------------------------

getconfig operation: true

-------------------------1-------------------------

getconfig operation: true

-------------------------2-------------------------

editconfig operation: true

{
     "interfaces":{
          "@xmlns":"urn:ietf:params:xml:ns:yang:ietf-interfaces",
          "interface":{
               "name":"swp2",
               "enabled":"true",
               "type":{
                    "@xmlns:ianaift":"urn:ietf:params:xml:ns:yang:iana-if-type",
                    "#text":"ianaift:ethernetCsmacd"
               },
               "gate-parameters":{
                    "@xmlns":"urn:ieee:std:802.1Q:yang:ieee802-dot1q-sched",
                    "gate-enabled":"true",
                    "config-change":"true",
                    "admin-control-list-length":"1",
                    "admin-control-list":{
                         "index":"0",
                         "operation-name":"set-gate-states",
                         "sgs-params":{
                              "gate-states-value":"3",
                              "time-interval-value":"4000"
                         }
```

Figure 70. Qci configuration

The **qci** interface allows user to select the configuration for "stream identify", "stream filter", "stream gate", and "flow metering".

Note:

1. Configure the "`stream identify`" first, then configure the "`stream gate`" and "f`low metering`", configure the "`stream filter`" at last.

2. "`index`" in "`stream filter`" configuration and "`streamhandle`" in "`stream identify`" should be the same value.

3. "`flow meter index`" in "`stream filter`" and "`index`" in "`flow metering`" should be the same value (63-246).

5.1.2.4.3 Dynamic remote configuration

The dynamic TSN configuration is used for configuring the TSN parameters dynamically. Users do not need to log into each TSN node to specify the TSN parameters for TSN configuration. They must only 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 YANG models deliver the configuration parameters are delivered to each node.

5.1.2.4.3.1 TSN working flow

This section provides an example of the TSN configuration working flow, which is described below:

After topology discovery and device registration, the network topology can be displayed over the web-browser. The user must select the nodes, specify the stream, input the timing requirement through the stream reservation component, and schedule configuration component. The results are passed down to the schedule mapping component to calculate the mapping from customer input to the TSN configuration. The configuration is instantiated using the YANG model and is delivered to different nodes for actual configuration.

The major components include:

- TSN network topology discovery
- Schedule mapping
- NETCONF/YANG configuration

- TSN Protocol Driver and TSN configuration
- Dashboard for stream management and customer input parse

Figure 71 shows the architecture diagram.

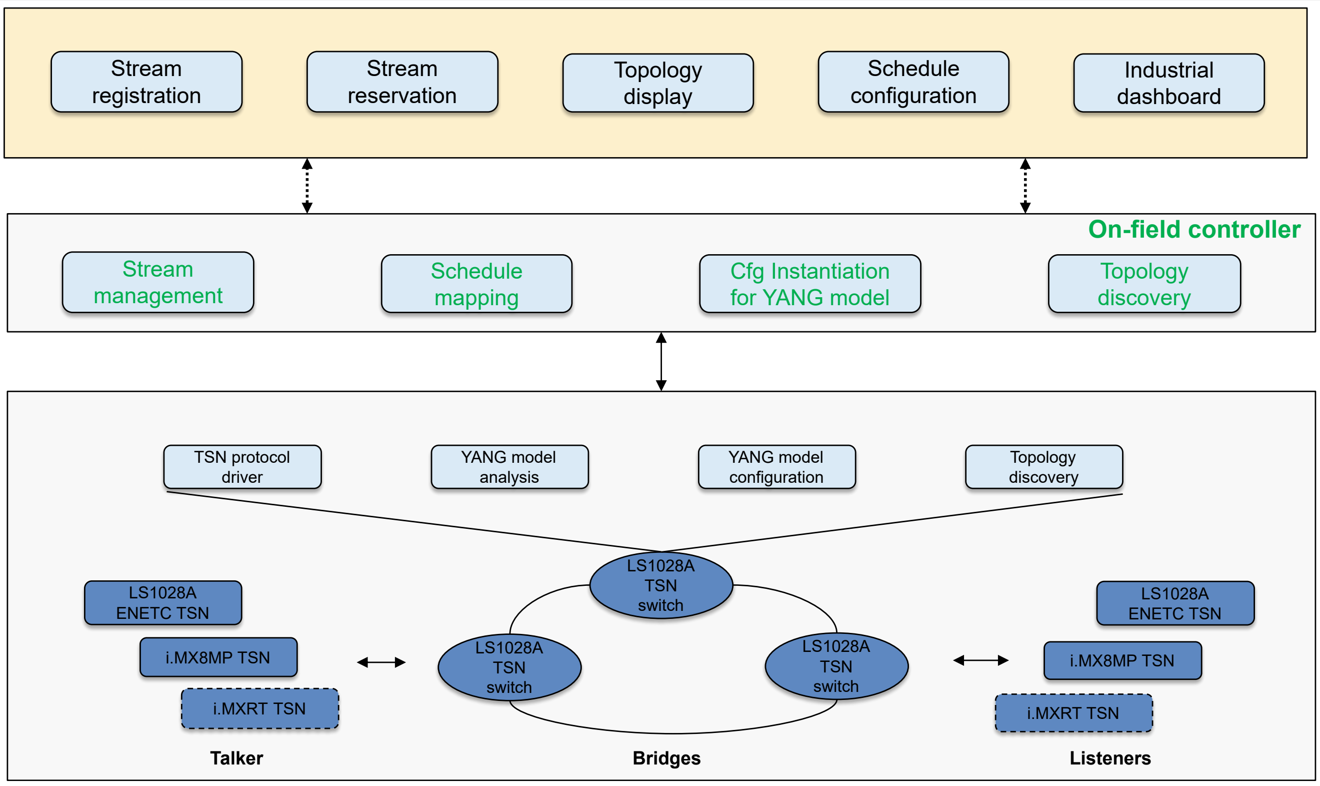


Figure 71. TSN architecture diagram

The Figure 71 illustrates the three layers of the TSN architecture. The first layer is the TSN network layer. The second layer is the service layer, which runs on the on-field controller/server. The third layer is an optional service that runs in the cloud or on-field server.

The TSN network layer includes TSN switches and endpoints that form the TSN network. For example, it includes the LS1028A TSN switch and TSN endpoints such as LS1028 ENETC TSN and i.MX 8M Plus TSN endpoint. The different components run on each of the nodes. For example, the topology discovery component collects the network topology, YANG model performs the TSN register configuration, and NETCONF server parses the YANG model for TSN configuration.

The second layer is the on-field controller layer. It is the server running on-field to host the services of the industrial board, topology discovery and schedule mapping.

The third layer runs on the cloud, which could host the services running on the on-field controller. This layer is an optional layer.

5.1.2.4.3.2 Topology discovery

The topology discovery component is used to discover network connections by running LLDP on each TSN network node. The connection information is delivered to topology discovery service running on the on-field server.

5.1.2.4.3.3 Path selection

Path selection implements an algorithm to select the path between the selected talker and listener. If there are multiple paths, the dashboard displays all paths and the user can select one of the paths for the stream. Set a different VLAN ID for the selected path, and the stream with this VID can flow in the path.

5.1.2.4.3.4 Path delay

Clock synchronization and path delay calculation are two prerequisites for schedule mapping. Clock synchronization uses gPTP to synchronize the clock of the system. The example described in this document uses linuxptp PMC tool to get the path delay.

Figure 72 shows a sample configuration to show the PMC running environment on LS1028ARDB boards.

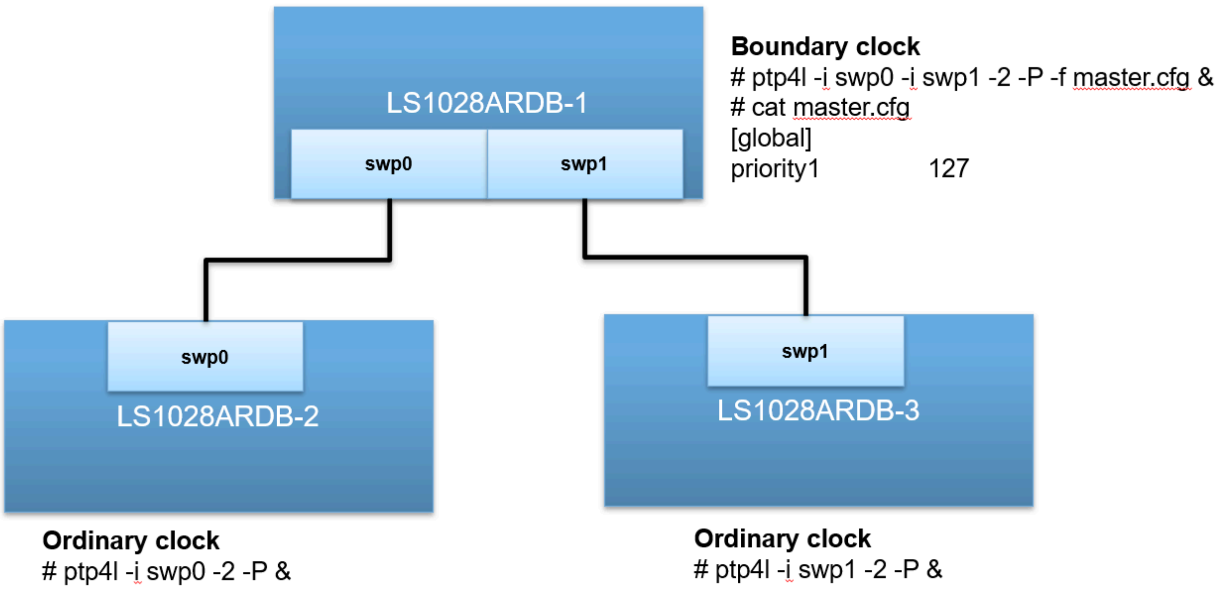


Figure 72. PMC running environment on LS1028ARDB boards

5.1.2.4.3.5 Schedule mapping

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

- Gets the user input and converting the input into TSN parameters.
- Gets the path and path delay from the link object of the NetworkGraph file.
- Gets the old TSN configuration for each node and calculates a new configuration to meet the user's requirements.

5.1.2.4.3.6 Dashboard configuration demo

The figure below shows the dashboard for the configuration demo.

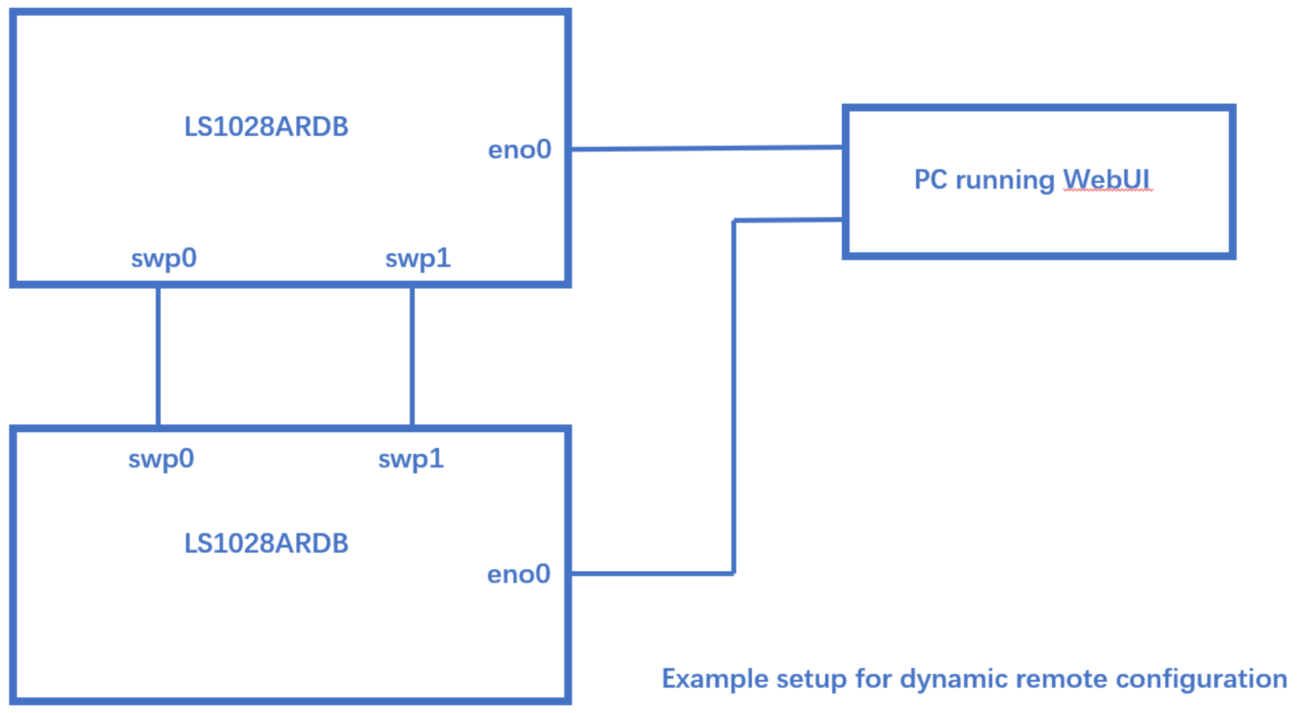


Figure 73. Dashboard Configuration Demo

Registering a stream

Click "**Check Path**" button, input the start device in "**first device**" input box, and end device in "**Second device**" input box. Then click the "**submit**" button, path is described in the Figure 74.

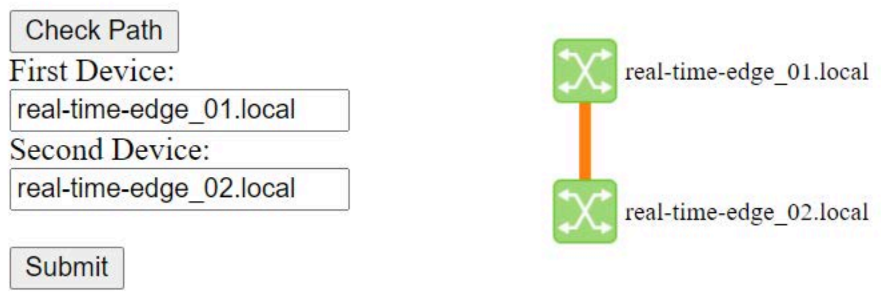
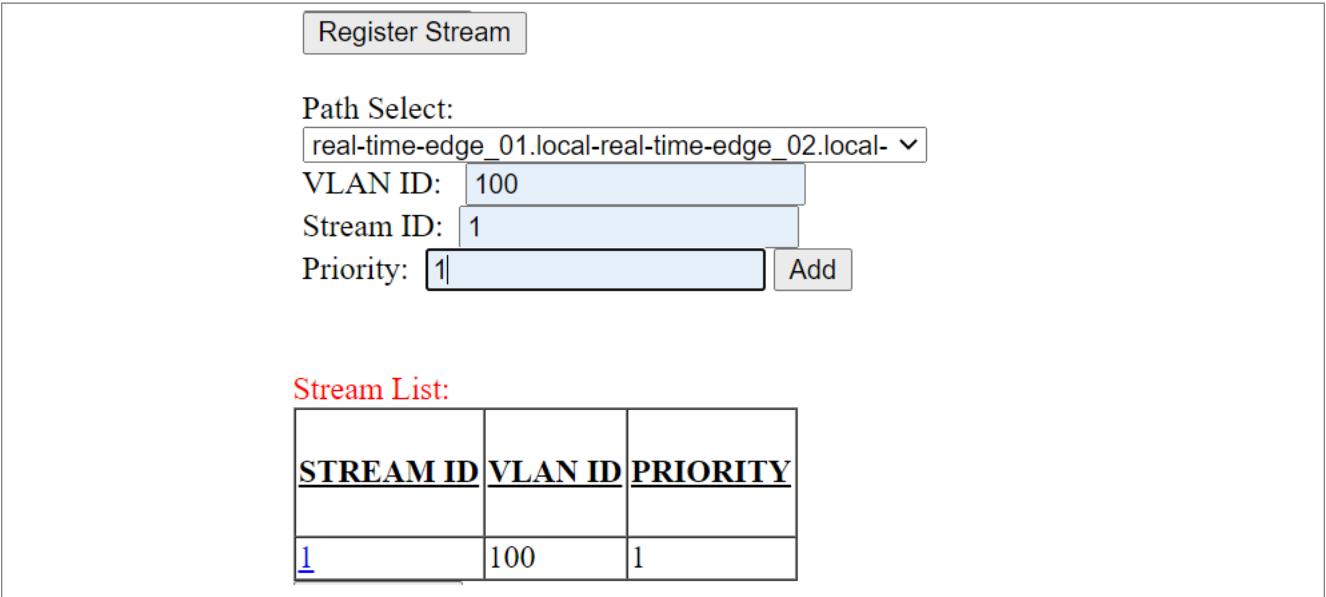


Figure 74. Stream Registration

Click "**Register Stream**" button, then select the path in path select. Fill **VLAN ID**, **Stream ID**, **Priority**, and then click "**Add**" button. The Figure 75 shows the output a stream table.



Path Select: real-time-edge_01.local-real-time-edge_02.local-

VLAN ID: 100

Stream ID: 1

Priority: 1

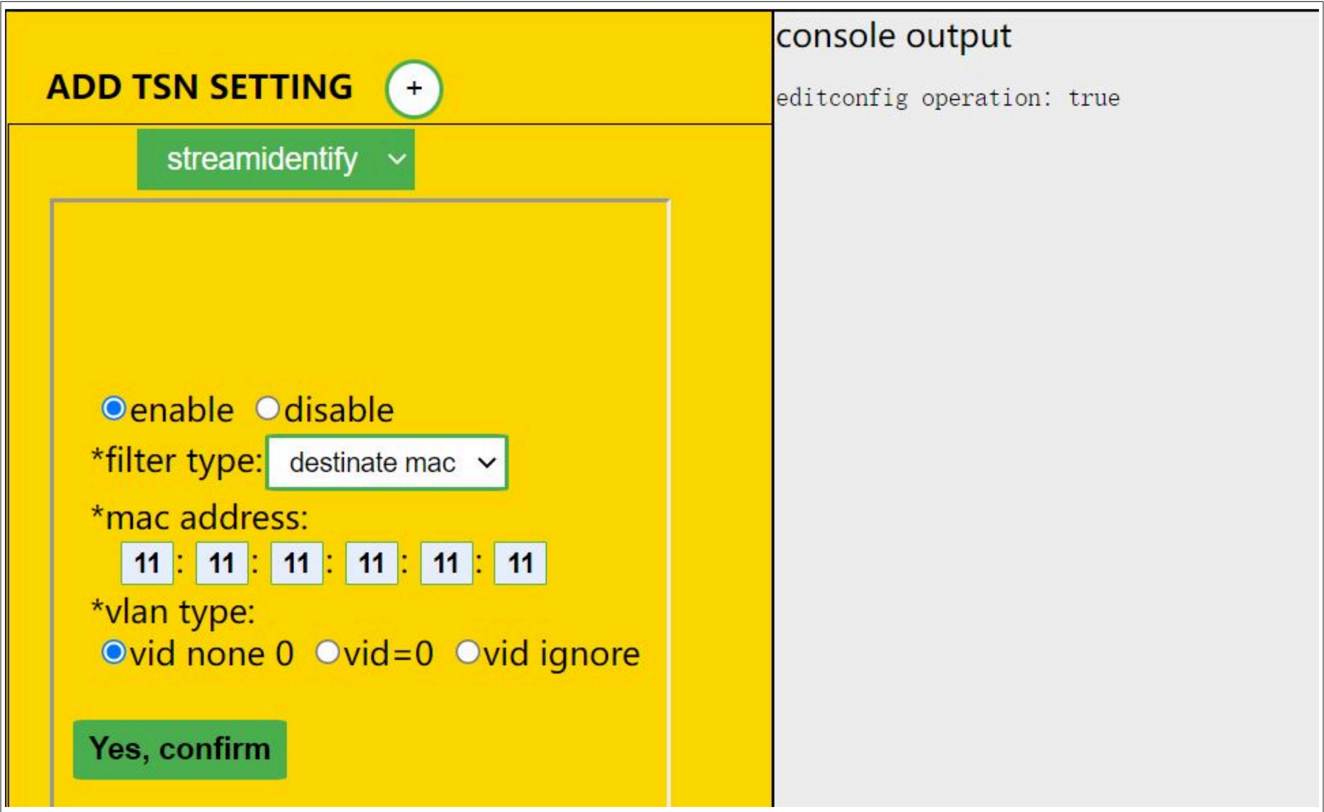
Stream List:

STREAM ID	VLAN ID	PRIORITY
1	100	1

Figure 75. Registering a Stream

Configure stream identification

Click the **stream ID** in the **stream table** and then jump to the stream configuration page. Select **streamidentify** and fill the information in the input boxes. The stream MAC information and VLAN ID identify a stream according to the 802.1CB definition. This information is used by the PSFP configuration. Therefore, the **streamidentify** page must be configured before configuring Qci and CFQ. Refer Figure 76.



(●) enable ( ) disable

*filter type: destinate mac

*mac address:
11 : 11 : 11 : 11 : 11 : 11

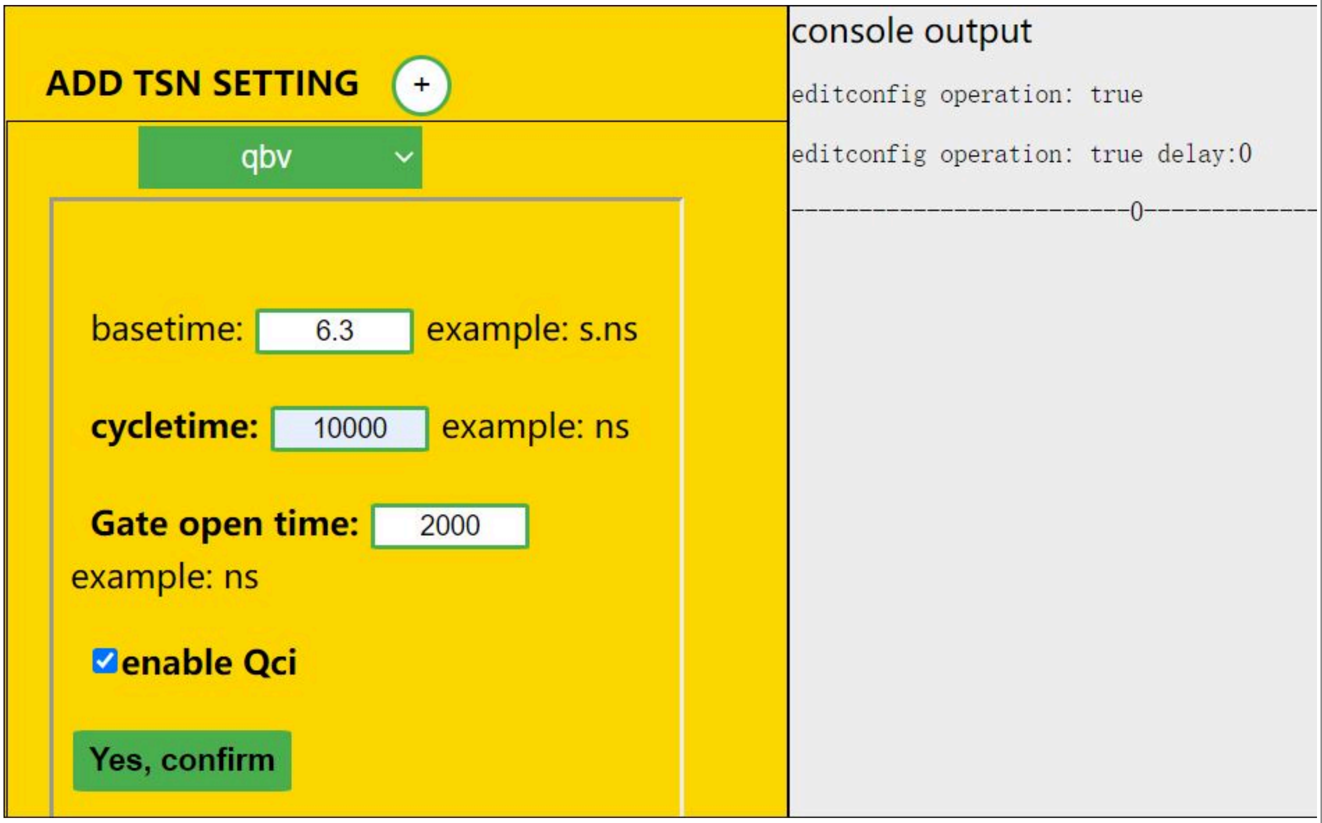
*vlan type:
(●) vid none 0 ( ) vid=0 ( ) vid ignore

```
editconfig operation: true
```

Figure 76. Configuring stream identification

Configure Qbv and Qci On Stream

Select Qbv, and then fill basetime, cycletime, and gate open time in the respective input textboxes. Select **enable Qci** button to configure both Qci gate control on input port and Qbv control list on the output port. The CNC server calculates the gate open time slot on each board and get a minimum time delay. Each path node tries to open gate with a minimum time delay. Refer to Figure 77.



basetime: 6.3 example: s.ns

cycletime: 10000 example: ns

Gate open time: 2000 example: ns

[✓] enable Qci

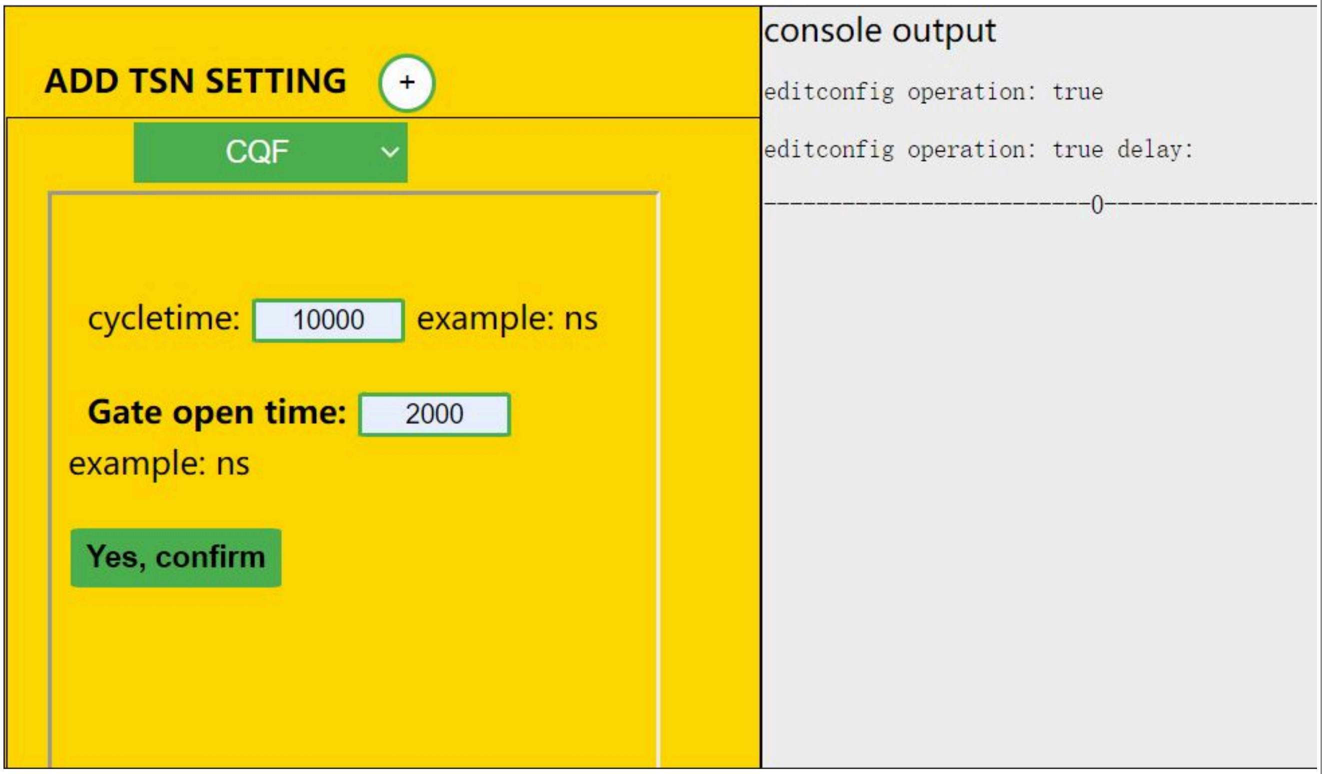
console output

```
editconfig operation: true

editconfig operation: true delay:0

-----------------------0------------
```

Figure 77. Configuring Qbv and Qci On Stream

Configuring CQF

The CQF configuration is based on the 802.1Qch definition to configure Qbv and Qci. The CQF configuration cannot be mixed with the previous Qbv configuration. In CQF configuration, the cycle time and gate open time for all streams must be the same, and cycle time must be an integer multiple of gate open time. Packets are delayed for a gate open time on each path node. Refer Figure 78.



cycletime: 10000 example: ns

Gate open time: 2000 example: ns

```
editconfig operation: true

editconfig operation: true delay:

--------------------------0-----------------
```

Figure 78. Configuring CQF

5.1.3 TSN on i.MX 943/i.MX 95

Real Time Edge software supports TSN implementation on i.MX 943 and i.MX 95 hardware platforms. The i.MX 943 processor has a NETC complex that has TSN ability. There are three endpoint ports (one of them is a MUX connected to a switch port) and two switch ports.

The following sections describe TSN configuration on i.MX 943 hardware platforms as an example.

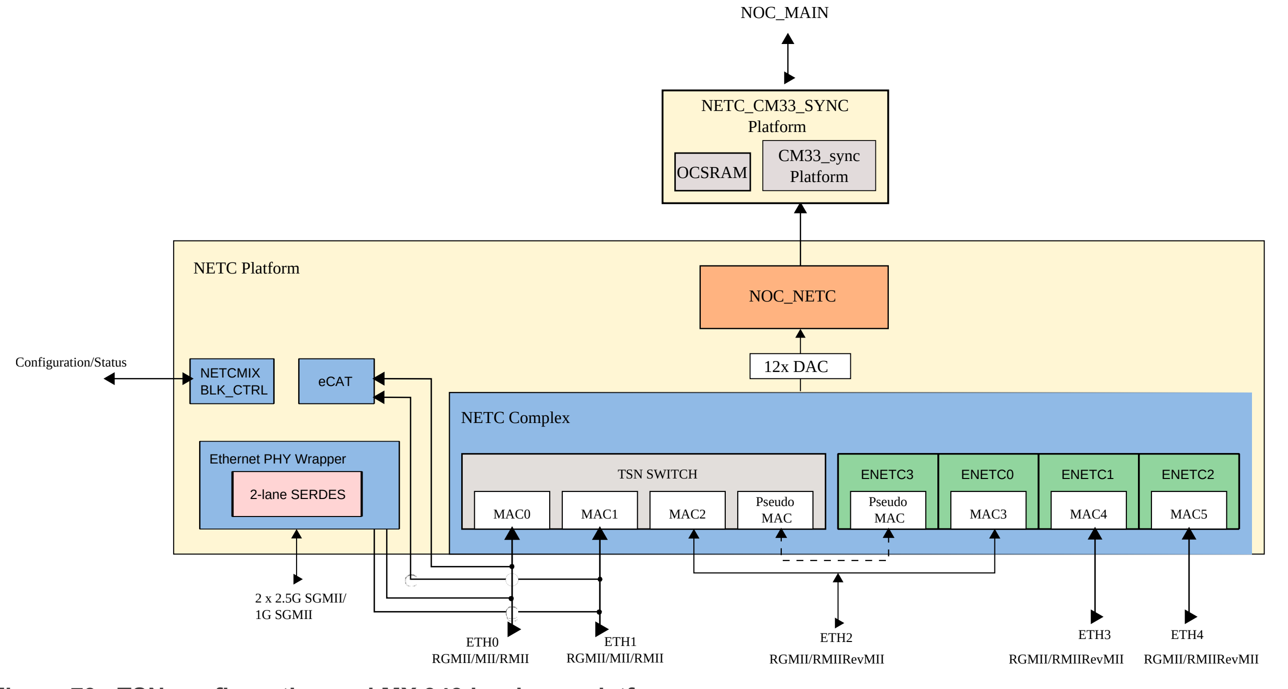


Figure 79. TSN configuration on i.MX 943 hardware platforms

5.1.3.1 TSN configuration on ENETC

Use eth1 connected to another device to test the TSN configurations.

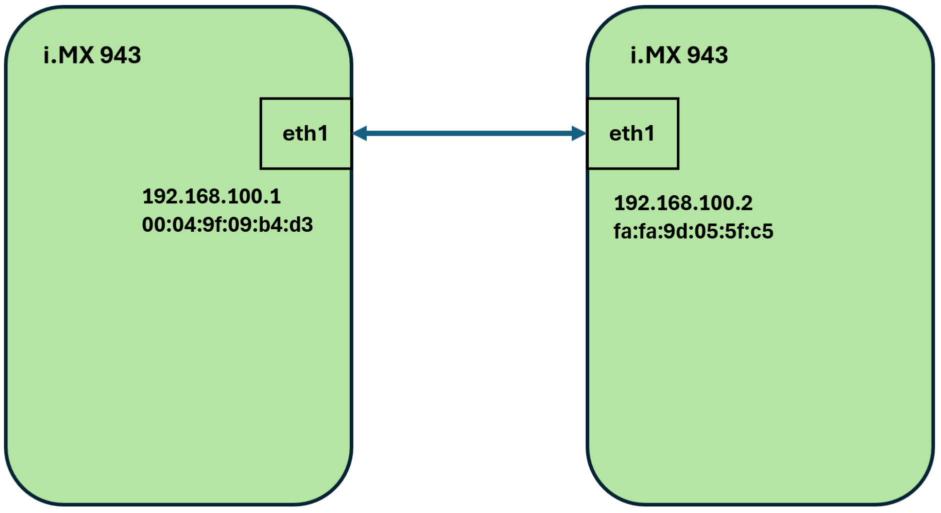


Figure 80. Testing the TSN configuration

5.1.3.1.1 Clock synchronization

To test 1588 synchronization on ENETC interfaces, use the following procedure:

1. Check PTP clock and timestamping capability:

```
# ethtool -T eth1
Time stamping parameters for eth1:
Capabilities:
        hardware-transmit
        software-transmit
        hardware-receive
        software-receive
        software-system-clock
        hardware-raw-clock
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
        off
        on
        onestep-sync
Hardware Receive Filter Modes:
        none
        all
```

2. Configure the IP address and run ptp4l on two boards:

   ```
   # ifconfig eth1 <ip_addr>
   # ptp4l -i eth1 -p /dev/ptp1 -m
   ```

3. After running, one board is automatically selected as the master, and the slave board prints synchronization messages.
4. For 802.1AS testing, just use the configuration file `gPTP.cfg` in the linuxptp source. Run the below command on the boards, instead:

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

5.1.3.1.1.1 Qbv

Basic gates closing

1. Close the queue 0 gate.

   ```
   tc qdisc replace dev eth1 parent root handle 100 taprio num_tc 4 map 0 1 2 3
    0 1 2 3 queues 1@0 1@1 1@2 1@3 base-time 0 sched-entry S fe 100000 flags 2
   ```

2. Pinging to the peer device fails.

   ```
   ping 192.168.100.2
   ```

3. Open the queue 0 gate.

   ```
   tc qdisc replace dev eth1 parent root handle 100 taprio num_tc 4 map 0 1 2 3
    0 1 2 3 queues 1@0 1@1 1@2 1@3 base-time 0 sched-entry S ff 100000 flags 2
   ```

4. Ping to the peer device, and check that the ping is successful.

   ```
   ping 192.168.100.2
   ```

Note: *i.MX 95 has 6 hardware queues. Therefore, set the* `num_tc` *in mqprio to 6.*

Performance test

1. Use ptp4l to synchronize ptp clock to the peer and synchronize ptp clock to Linux system clock on two connected boards.

```
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 &
```

2. Configure Qbv on the sender board

```
tc qdisc replace dev eth1 parent root handle 100 taprio num_tc 4 map 0 1 2 3 0 1
 2 3 queues 1@0 1@1 1@2 1@3 base-time 0 sched-entry S fd 100000 sched-entry S 2
 5000 sched-entry S fd 895000 flags 2
```

3. Use isochron to send packets periodically on the sender board as shown below.

```
isochron send -i eth0 --base-time 10.0 --cycle-time 0.001 -s 256 -d
 ff:ff:ff:ff:ff:ff -p 1 -n 10
```

The test result show that packets send on each 1 ms, the Qbv gate opens at 100 µs offset. The packets are sent after the gate opens.

5.1.3.1.1.2 Qbu

1. Connect the eth1 port of the two i.mx 943 boards. Enable preemption MAC merge layer on the two boards.

   ```
   ethtool --set-mm eth1 tx-enabled on pmac-enabled on verify-enabled off tx-
   min-frag-size 60
   ```

2. Set the queue 2 to be preemptable.

   ```
   tc qdisc add dev eth1 root handle 1: mqprio num_tc 4 map 0 1 2 3 queues 1@0
   1@1 1@2 1@3 fp E E P E hw 1
   ```

3. Send two streams into queue 1 and queue 2 on the sender board. Check the below counter for the number of fragments transmitted

   ```
   /usr/share/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 1 -s 150 -n 0 -m
   90:e2:ba:ff:ff:ff -n 0
   ```

4. Check the below counter for the number of fragments transmitted.

   ```
   ethtool -I --show-mm eth1 | grep MACMergeFragCountTx
   ```

5.1.3.1.1.3 Qav

1. Configure different traffic classes.

```
tc qdisc add dev eth1 root handle 1: mqprio num_tc 4 map 0 0 1 1 2 2 3 3 queues
 1@0
1@1 1@2 1@3 hw 1
```

2. Configure the bandwidth on eth1 as 100 Mbit/s.

```
tc qdisc replace dev eth1 parent 1:4 cbs locredit -1350 hicredit 150 sendslope
-900000 idleslope 100000 offload 1
```

3. Send a stream with full bandwidth on eth1, it must be about 100 Mbit/s rate.

```
/usr/share/samples/pktgen/pktgen_sample01_simple.sh -i eth1 -q 3 -s 500 -n
30000
```

5.1.3.1.1.4 Qci

Qci gate test

1. Close the Qci gate on eth1.

```
tc qdisc add dev eth1 clsact
tc filter add dev eth1 ingress flower skip_sw \
dst_mac 00:04:9f:09:b4:d3 action gate index 1 base-time 0 \
sched-entry CLOSE 6000 -1 -1
```

2. A ping sent from the board B to board A fails.

Qci flow meter test

1. Use the command to limit the flow meter rate for the stream sent to board A.

```
tc filter add dev eth1 ingress flower skip_sw \
  dst_mac 00:04:9f:09:b4:d3 action gate index 1 base-time 0 \
  sched-entry OPEN  6000 -1 -1 action police index 1 rate 10Mbit \
  burst 10000 conform-exceed drop/ok
```

2. Test the stream bandwidth, which is limited to 10 Mbit/s.

```
//On board B
Iperf -s
//On board A
Iperf -c 192.168.100.2
```

5.1.3.2 TSN configuration on NETC switch (i.MX 943)

Use two switch ports swp0 and swp1 connected to another two devices to test the TSN configurations.

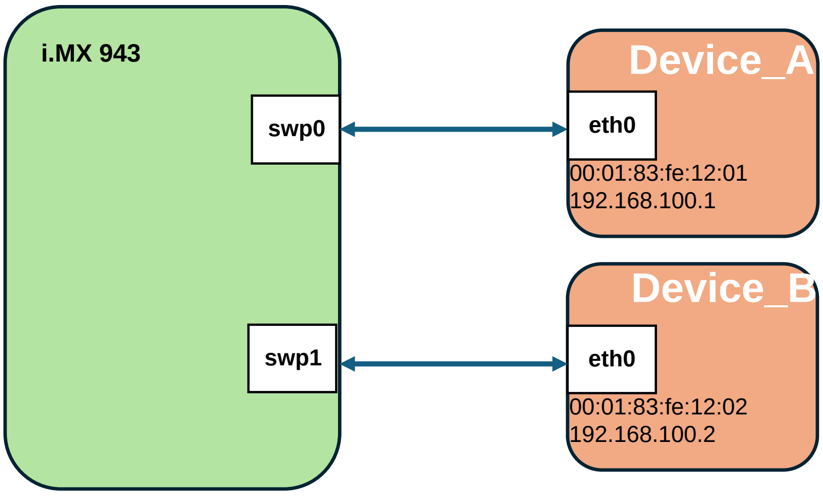


Figure 81. Testing the TSN configuration

Add the switch ports in bridge.

```
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
ifconfig switch 192.168.100.3
```

5.1.3.2.1 Clock synchronization

To test 1588 synchronization on ENETC interfaces, use the following procedure:

1. Check the PTP clock and timestamping capability as shown below:

   ```

   # ethtool -T swp0
   Time stamping parameters for swp0:
   Capabilities:
           hardware-transmit
           software-transmit
           hardware-receive
           software-receive
           software-system-clock
           hardware-raw-clock
   PTP Hardware Clock: 0
   Hardware Transmit Timestamp Modes:
           off
           on
           onestep-sync
   Hardware Receive Filter Modes:
           none
           ptpv2-l4-event
           ptpv2-l2-event
           ptpv2-event
   ```

2. For 802.1AS testing, use the configuration file `gPTP.cfg` in the Linuxptp source. Run the below command on the boards, instead:

   ```
   # ptp4l -i swp0 -p /dev/ptp0 -f /etc/ptp4l_cfg/gPTP.cfg -m
   ```

5.1.3.2.1.1 Qbv (i.MX943)

1. Close the queue 0 gate.

```

            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 fe 100000 flags 2
```

2. Initiate a ping from device A to device B. Observe that it fails.

```

            ping 192.168.100.2
```

3. Open the queue 0 gate.

```

            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 ff 100000 flags 2A ping from the device A to device B is successful.
```

```

```

```
            ping 192.168.100.2
```

Performance test

1. Use ptp4l to synchronize PTP clock to switch and synchronize PTP clock to Linux system clock on device A and device B.

```
            ptp4l -i eth0 -p /dev/ptp0 -f /etc/ptp4l_cfg/gPTP.cfg -m > /var/log/
ptp4l.log 2>&1 &
            phc2sys -s eth0 -O 0 -S 0.00002 -m > /var/log/phc2sys.log 2>&1 &
```

2. Use ptp4l to synchronize the ptp clock on the i.mx943 switch.

```
            ptp4l -i swp0 -i swp1 -p /dev/ptp0 -f /etc/ptp4l_cfg/gPTP.cfg -m > /
var/log/ptp4l.log 2>&1 &
```

3. Configure Qbv on i.MX943 switch board.

```
            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 fd 100000 sched-entry S 2 5000 sched-entry S fd 895000 flags 2
```

3. Use isochron to send packets periodically on the board A.

```
            isochron send -i eth0 --base-time 10.0 --cycle-time 0.001 -s 256 -d
 ff:ff:ff:ff:ff:ff -p 1 -n 10
```

4. Use isochron to receive packets on board B:

```
            Isochron recv -i eth0
```

5. The test result shows that packets are sent on each 1 ms, the qbv gate opens at 100 µs offset. The packets are sent after the gate opens.

5.1.3.2.1.2 Qbu (i.MX 943)

Follow the steps below to enable Qbu:

1. Connect the swp0 port to device A, the swp1 port to device B, and the swp2 port to the device C. The Ethernet port of Device B must support the MAC merge layer of 802.3br.
2. Enable preemption MAC merge layer on swp1 and device B.

   ```
   ethtool --set-mm swp1 tx-enabled on pmac-enabled on verify-enabled off tx-
   min-frag-size 60
   ```

3. Set the queue 2 to be preemptable.

   ```
   tc qdisc add dev swp1 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 fp E E P E E E E E hw 1
   ```

4. Send two streams on device A and device C.

```
//On device A
VLAN_ID=1 VLAN_P=1 VLAN_CFI=0 \   /usr/share/samples/pktgen/
pktgen_sample01_simple.sh -i eth0 -q 0 -s 500 \
 -m “00:01:83:fe:12:02” -n 0
//On device C
VLAN_ID=1 VLAN_P=2 VLAN_CFI=0 \ /usr/share/samples/pktgen/
pktgen_sample01_simple.sh -i eth0 -q 0 -s 500 \
  -m “00:01:83:fe:12:02” -n 0
```

5. Check the below counter for the number of fragments transmitted.

```
   ethtool -I --show-mm swp1 | grep MACMergeFragCountTx
```

5.1.3.2.1.3 Qav (i.MX 943)

1. Configure different traffic classes.

```
tc qdisc add dev swp1 parent root handle 100: 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 1
```

2. Configure the bandwidth on swp1 as 10.8 Mbit/s.

```
tc qdisc replace dev swp1 parent 100:8 cbs idleslope 10800 sendslope -89200
 hicredit 12 locredit -113 offload 1
```

3. Send a stream on device A and test the bandwidth on the receiver of device B. It must be about 10.8 Mbit/s rate.

```
//On device A
VLAN_ID=1 VLAN_P=7 VLAN_CFI=0 \ /usr/share/samples/pktgen/
pktgen_sample01_simple.sh -i eth0 -q 0 -s 500 \
  -m “00:01:83:fe:12:02” -n 0
```

5.1.3.2.1.4 Qci (i.MX 943)

Qci flow meter test

- Limit the flow meter rate for the stream sent to board A by using the command below:

```
tc filter add dev swp0 ingress flower skip_sw \
   dst_mac 00:01:83:fe:12:02 action gate index 1 base-time 0 \
   sched-entry OPEN  6000 -1 -1 action police index 1 rate 10Mbit \
   burst 10000 conform-exceed drop/ok
```

- Test the stream bandwidth. Observe that it is limited to 10 Mbit/s.

```
//On device B
  Iperf -s
//On device A
  Iperf -c 192.168.100.2
```

Qci gate test (i.MX 943)

1. Close the Qci gate on swp0 using the command below:

```
tc qdisc add dev swp0 clsact
tc filter add dev swp0 ingress flower skip_sw \
dst_mac 00:01:83:fe:12:02 action gate index 1 base-time 0 \
sched-entry CLOSE 6000 -1 -1
```

2. Observe that a ping from device B to device A fails.

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

The following sections describe TSN configuration on i.MX 8DXL, i.MX 8M Plus, or i.MX 93 hardware platforms

5.1.4.1 Test environment

On i.MX 8M Plus EVK / i.MX 93 EVK / i.MX 93 14x14 EVK platform, the interface name of `ENET_QOS` port which supports TSN is eth1. On i.MX 8DXL EVK / i.MX 93 9x9 LPDDR4 QSB, the interface name of `ENET_QOS` port which supports TSN is eth0.

Note: *For i.MX 93 14x14 EVK, in order to use ENET_QOS interface, TJA1103SDB ENET PHY daughter card is needed to be connected on **J9** connector. To verify the TSN features, two i.MX 93 14x14 EVK boards are connected back-to-back on ENET_QOS interface via TJA1103ADB ENET PHY daughter card. Also on one of the TJA1103SDB, connect jumper to short pin 2-3 of **J14** (CONFIG 6).*

Connect ENET_QOS port to the TestCenter to test TSN features. The commands in this section use the i.MX 8M Plus EVK platform as example:

Use the following command to check the TSN Ethernet device name:

```
#ls /sys/devices/platform/soc@0/30800000.bus/30bf0000.ethernet/net/ eth1
```

The Figure 82 shows the TSN test environment setup.

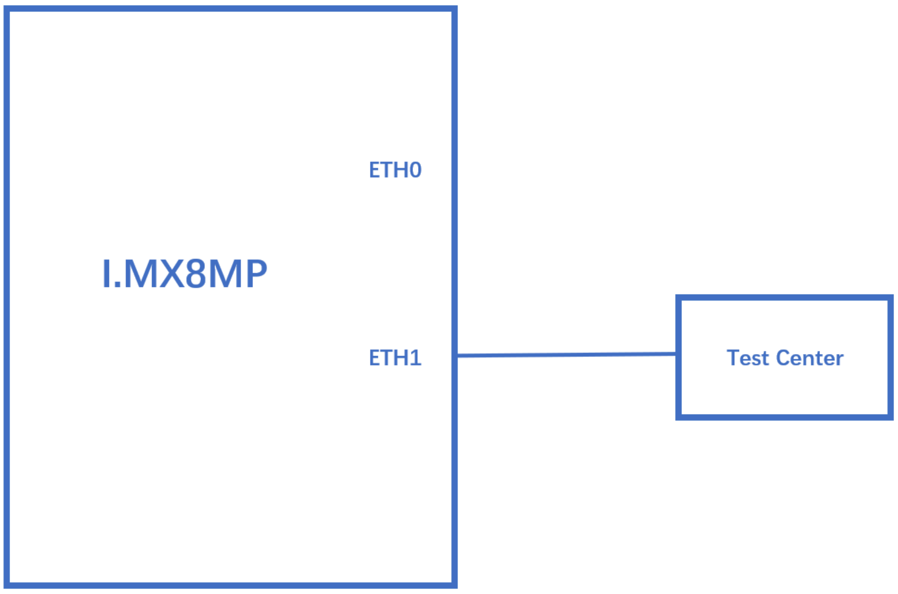


Figure 82. TSN test environment setup

Note: *TestCenter is a device used to capture streams from eth1 of i.MX 8M Plus board. For this example, the Spirent TestCenter is used to capture preemptable frames in the Qbu test case.*

5.1.4.2 Clock synchronization

To test 1588 synchronization on dwcMAC interfaces, use the following procedure:

1. Connect eth1 interfaces on two boards in a back-to-back manner. The Linux booting log is as follows:

```
…
pps pps0: new PPS source ptp0
…
```

2. Configure the IP address using the command below:

```
ifconfig eth1 192.168.3.1
```

3. Check PTP clock and time stamping capability:

```
# ethtool -T eth1
Time stamping parameters for eth1:
Capabilities:
       hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
       software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
       hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
       software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
       software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
       hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
       off                   (HWTSTAMP_TX_OFF)
       on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
       none                  (HWTSTAMP_FILTER_NONE)
       all                   (HWTSTAMP_FILTER_ALL)
       ptpv1-l4-event        (HWTSTAMP_FILTER_PTP_V1_L4_EVENT)
       ptpv1-l4-sync         (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
       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, run clock synchronization using the ptp4l command along with the parameter `--hwts_filter=full`. For example:*

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

5.1.4.3 Qbv

Note: *The Qbu frame preemption capability is enabled automatically if the link partner also supports frame preemption. This is achieved by LLDP protocol by which the link partner announces its support for the preemption capability via an Additional Ethernet Capabilities TLV in an LLDPDU addressed to the Nearest Bridge group address (see IEEE Std 802.1Q).. So firstly user needs to check whether frame preemption is active by command `ethtool --show-frame-preemption eth1`. If so, user needs to disable frame preemption using command `ethtool --set-frame-preemption eth1 disabled` before configuring Qbv using `S` (SetGateStates) command in `sched-entry`.*

1. Enable the `ptp device`, and get the current `ptp time`.

```
ptp4l -i eth1 -p /dev/ptp1 -m
#Get current time(seconds)
devmem2 0x30bf0b08
0x5E01F9B2
```

2. Get the basetime to be 2 minutes later.

```
#Basetime = (currentime + 120) * 1000000000 = 1577187882000000000
```

3. Set time schedule, open queue 1 in 100 µs and open queue 2 in 100 µs.

```
tc qdisc replace dev eth1 parent root handle 100 taprio \
    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.

```
/usr/share/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) is obtained. Or if the Ethernet port is connected to another board, the frames can be captured on that board by using the 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 must be set on each `tc taprio` command.*
- *Use "`devmem2 0x30bf0c58`" to get `Qbv status` and check if `qbv status` is active. Refer to the `MTL_EST_Status` register.*

5.1.4.4 Qbu

1. Using ethtool to enable Qbu on eth1, set queue 2 to be preemptable.

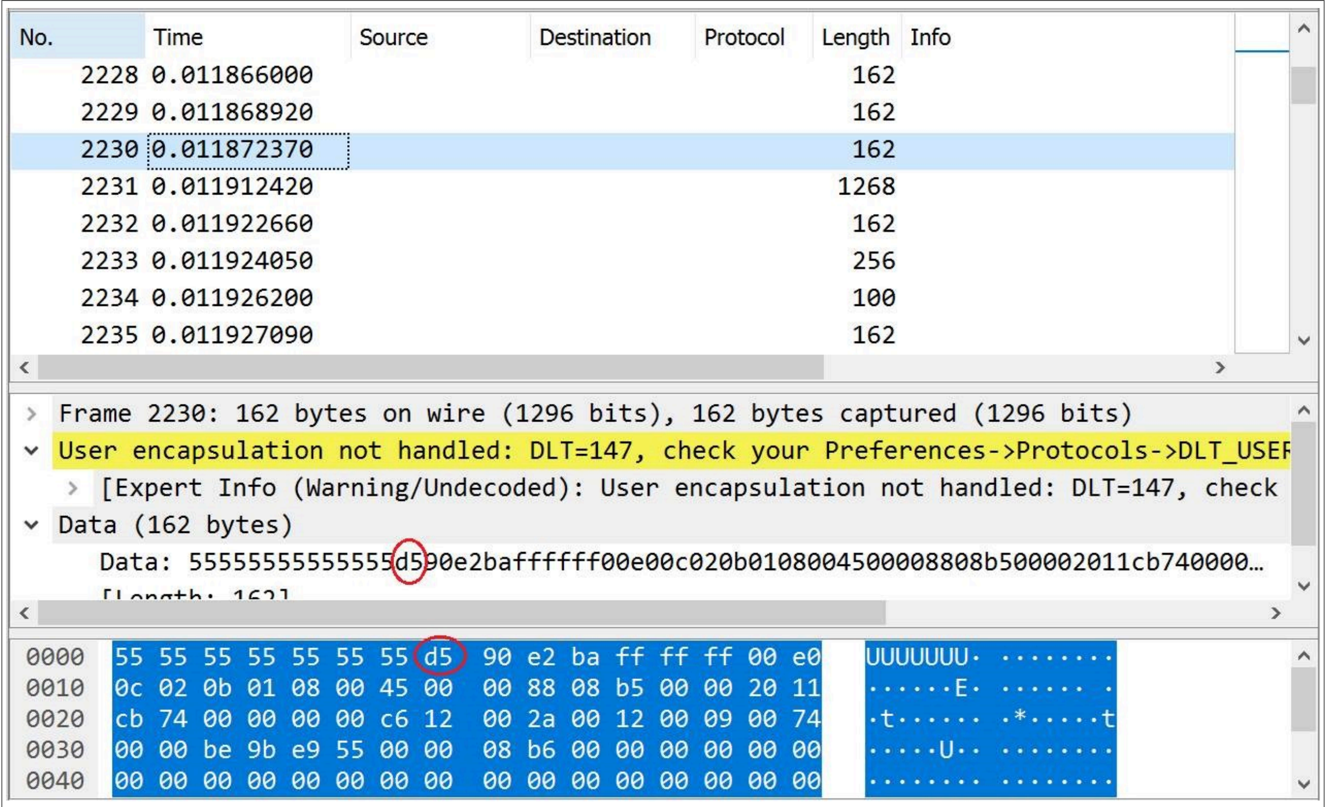
```
ethtool --set-mm eth1 tx-enabled on pmac-enabled on verify-enabled off  tx-
min-frag-size 60
tc qdisc add dev eth1 root handle 1: mqprio num_tc 5 map 0 1 2 3 4 queues 1@0
 1@1 1@2 1@3 1@4 fp P E P E E hw 1
```

   Note: *Once Qbu is enabled, queue 0 is always preemptable queue. To support preemption, the MAC should have at least 1 queue designated as an express queue.*
   Note: *On a back-to-back setup using two i.MX 8M Plus EVK boards connected via eth1, ensure that Qbu is enabled on eth1 of both boards.*

2. Send two streams into queue 1 and queue 2.

```
/usr/share/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 1 -s 150 -n 0 -m
 90:e2:ba:ff:ff:ff
```

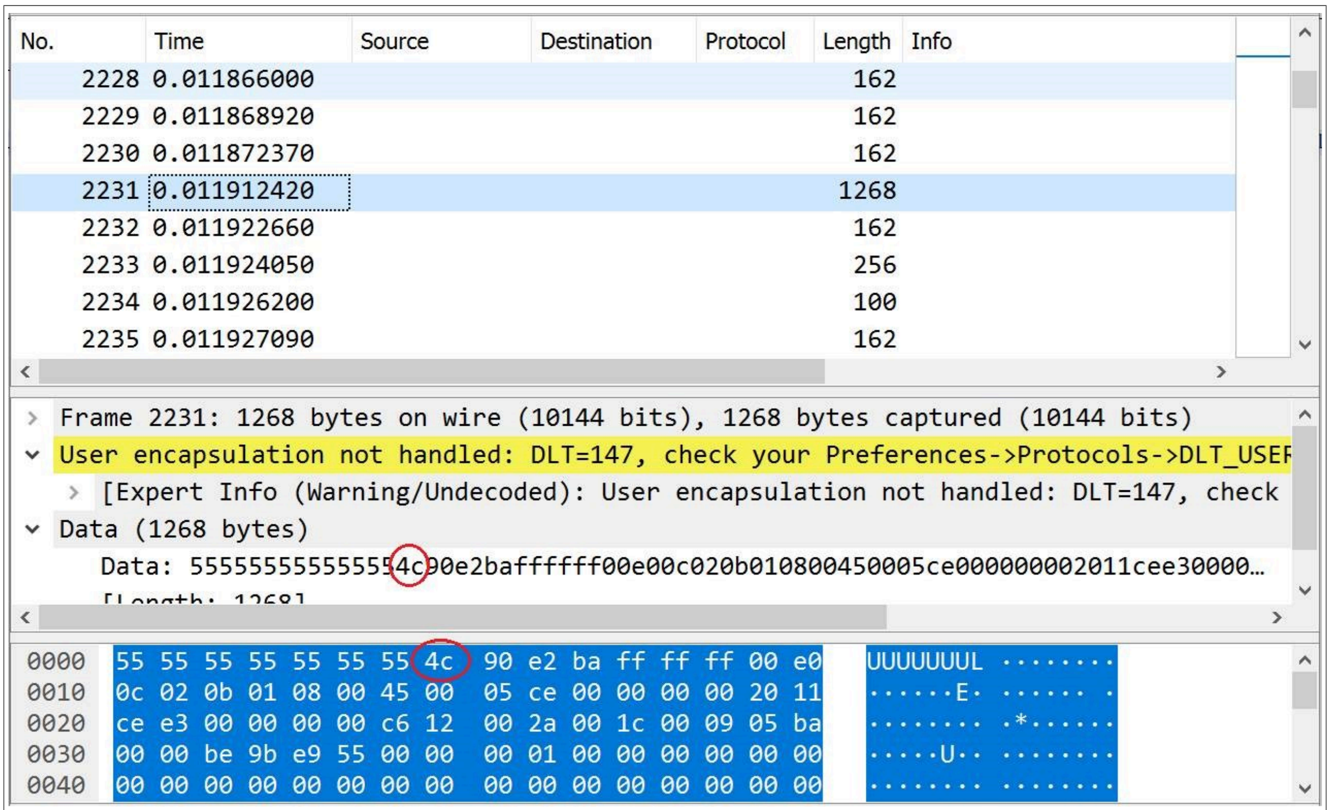
3. Capture the mPacket on Spirent TestCenter. Users can observe that Q2 frames are preempted into fragments.
   Note: *Spirent TestCenter can capture the preamble of mPacket. Refer to Section 99.3, "MAC Merge Packet (mPacket)" of IEEE standard for Ethernet 802.3-2018 for the mPacket format.*
   a. Below is an example mPacket that contains an express packet, which has SMD value of 0xD5.



```
No.   Time         Source  Destination  Protocol  Length  Info
2228 0.011866000                                  162
2229 0.011868920                                  162
2230 0.011872370                                  162
2231 0.011912420                                  1268
2232 0.011922660                                  162
2233 0.011924050                                  256
2234 0.011926200                                  100
2235 0.011927090                                  162

> Frame 2230: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits)
v User encapsulation not handled: DLT=147, check your Preferences->Protocols->DLT_USEF
  > [Expert Info (Warning/Undecoded): User encapsulation not handled: DLT=147, check
v Data (162 bytes)
    Data: 55555555555555d590e2baffffff00e00c020b0108004500008808b500002011cb740000…
    [Length: 162]

0000  55 55 55 55 55 55 55 d5  90 e2 ba ff ff ff 00 e0   UUUUUUU· ········
0010  0c 02 0b 01 08 00 45 00  00 88 08 b5 00 00 20 11   ······E· ······ ·
0020  cb 74 00 00 00 00 c6 12  00 2a 00 12 00 09 00 74   ·t······ ·*·····t
0030  00 00 be 9b e9 55 00 00  08 b6 00 00 00 00 00 00   ·····U·· ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

Figure 83. Sample mPAcket that contains an express packet

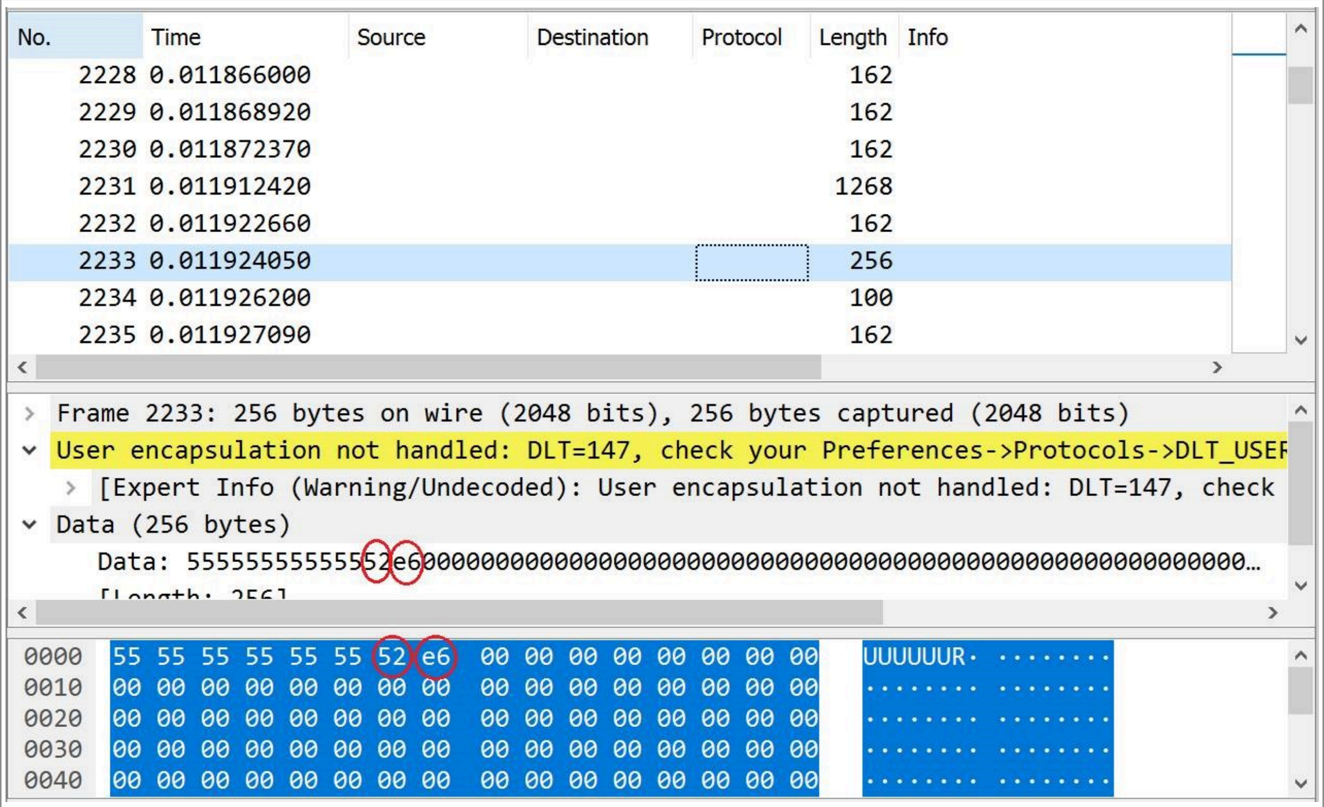
   b. Below is an example mPacket containing an initial fragment of a preemptable packet, which has SMD-S1 value of 0x4C.



```
No.   Time         Source   Destination   Protocol   Length  Info
2228 0.011866000                                      162
2229 0.011868920                                      162
2230 0.011872370                                      162
2231 0.011912420                                     1268
2232 0.011922660                                      162
2233 0.011924050                                      256
2234 0.011926200                                      100
2235 0.011927090                                      162

> Frame 2231: 1268 bytes on wire (10144 bits), 1268 bytes captured (10144 bits)
v User encapsulation not handled: DLT=147, check your Preferences->Protocols->DLT_USEF
   > [Expert Info (Warning/Undecoded): User encapsulation not handled: DLT=147, check
v Data (1268 bytes)
     Data: 55555555555555 4c 90e2baffffff00e00c020b010800450005ce000000002011cee30000…

0000  55 55 55 55 55 55 55 4c  90 e2 ba ff ff ff 00 e0   UUUUUUUL ········
0010  0c 02 0b 01 08 00 45 00  05 ce 00 00 00 00 20 11   ······E· ······ ·
0020  ce e3 00 00 00 00 c6 12  00 2a 00 1c 00 09 05 ba   ········ ·*······
0030  00 00 be 9b e9 55 00 00  00 01 00 00 00 00 00 00   ·····U·· ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

Figure 84. 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.



```
No.   Time          Source   Destination   Protocol   Length   Info
2228 0.011866000                                      162
2229 0.011868920                                      162
2230 0.011872370                                      162
2231 0.011912420                                      1268
2232 0.011922660                                      162
2233 0.011924050                                      256
2234 0.011926200                                      100
2235 0.011927090                                      162

> Frame 2233: 256 bytes on wire (2048 bits), 256 bytes captured (2048 bits)
v User encapsulation not handled: DLT=147, check your Preferences->Protocols->DLT_USEF
  > [Expert Info (Warning/Undecoded): User encapsulation not handled: DLT=147, check
v Data (256 bytes)
    Data: 55555555555552e6000000000000000000000000000000000000000000000000…

0000  55 55 55 55 55 55 52 e6  00 00 00 00 00 00 00 00   UUUUUUR· ········
0010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

Figure 85. 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
```

5.1.4.4.1 Preemption verification

The preemption capability is enabled only if the link partner announces its support for the preemption capability via an Additional Ethernet Capabilities TLV in an LLDPDU addressed to the Nearest Bridge group address (see IEEE Std 802.1Q). The preemption capability is disabled if the MAC Merge sublayer receives indication of link failure.

Connect `eth1` of two boards back to back. Then, run below command to enable the hardware verification:

```
ethtool --set-mm eth1 tx-enabled on pmac-enabled on verify-enabled on  tx-min-
frag-size 60
```

After that, make the link down and then up to enable the verify mPacket and response mPacket to be exchanged between two boards. Run below command to show the preemption status.

```
ethtool --show-mm eth1
```

Capture the LLDP frames on `eth1` port to check the Additional Ethernet Capabilities TLV.

5.1.4.5 Qav

To perform a Qav test, perform the steps listed below:

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 the bandwidth of queue 3 to be 20 Mbit/s.
   ```
   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:
   ```
   /usr/share/samples/pktgen/pktgen_sample01_simple.sh -i eth1 -q 3 -s 500 -n
    3000
   ```
4. Get the result. The bandwidth is 19 Mbit/s. For a workaround to the deviation, refer to the Note 1 after point 8 of this section.
   ```
   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 the bandwidth of queue 4 to be 40 Mbit/s.
   ```
   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.

```
/usr/share/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:

```
/usr/share/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 3 -s 1500 -n 0
```

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

Note: *According to errata TKT0649448, the following formula holds true:*

Additional/Extra bandwidth = ((# of packets x 12 bytes) / (Total number of bytes that is transmitted in that window, including the preamble bytes of each packet)) x Fractional bandwidth.

actual bandwidth = goal_bandwidth * (1 – (12 / (8 + average_packet_length))

5.1.5 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.

5.1.5.1 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 queues
 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 hw 1
```

5.1.5.1.1 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
   ```

5.1.5.1.2 Qbv

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

5.1.5.1.2.1 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.
```

5.1.5.1.2.2 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
```

5.1.5.1.2.3 Qbv performance test

Use the setup described in the Figure 86 for testing ENETC port0 (MAC0).

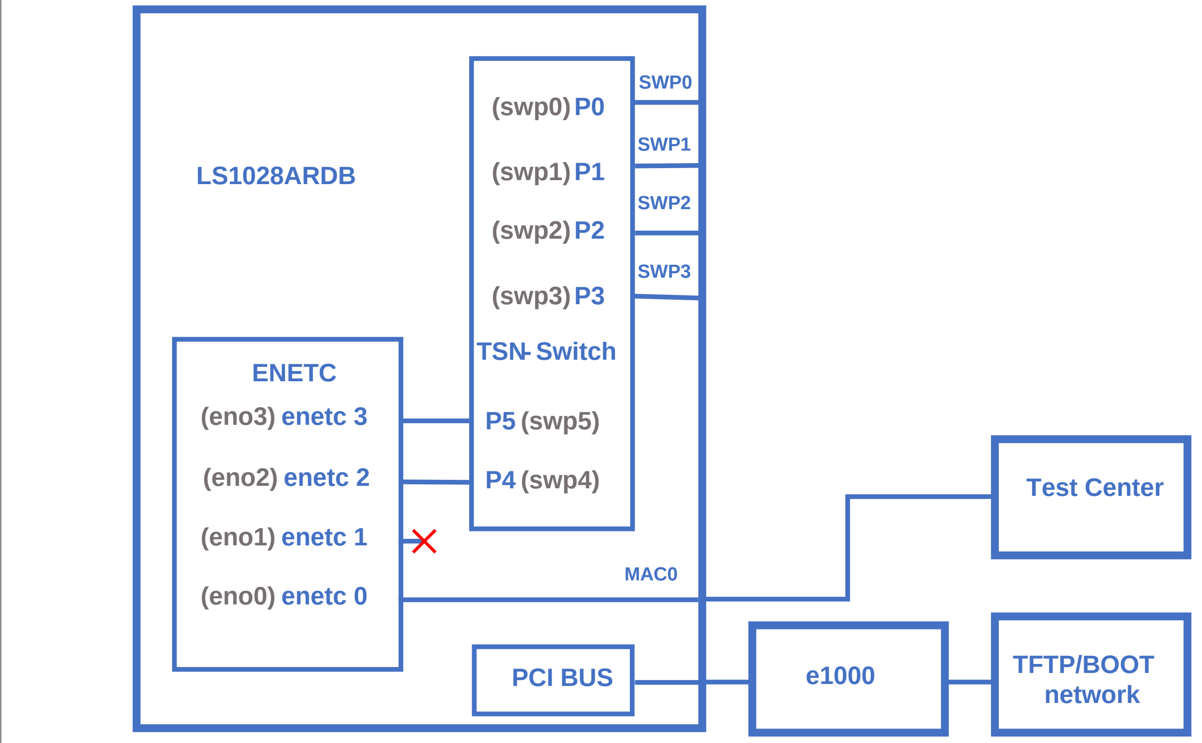


Figure 86. Setup for testing ENETC port0

Note: *TestCenter is a device used to capture streams from* `enetc0` *of LS1028ARDB board. Users can use another board to capture streams by using* `tcpdump` *command and then use Wireshark to analyze it.*

```
cat > qbv5.txt << EOF
t0 11111111b 1000000 t1 00000000b 1000000
EOF
qbvset --device eno0 --entryfile qbv5.txt
/usr/share/samples/pktgen/pktgen_twoqueue.sh -i eno0 -q 3 -n 0
#The stream would get about half line rate
```

5.1.5.1.2.4 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
```

5.1.5.1.3 Qbu

- If using 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 the using only one board, 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 figure below for the Qbu test.

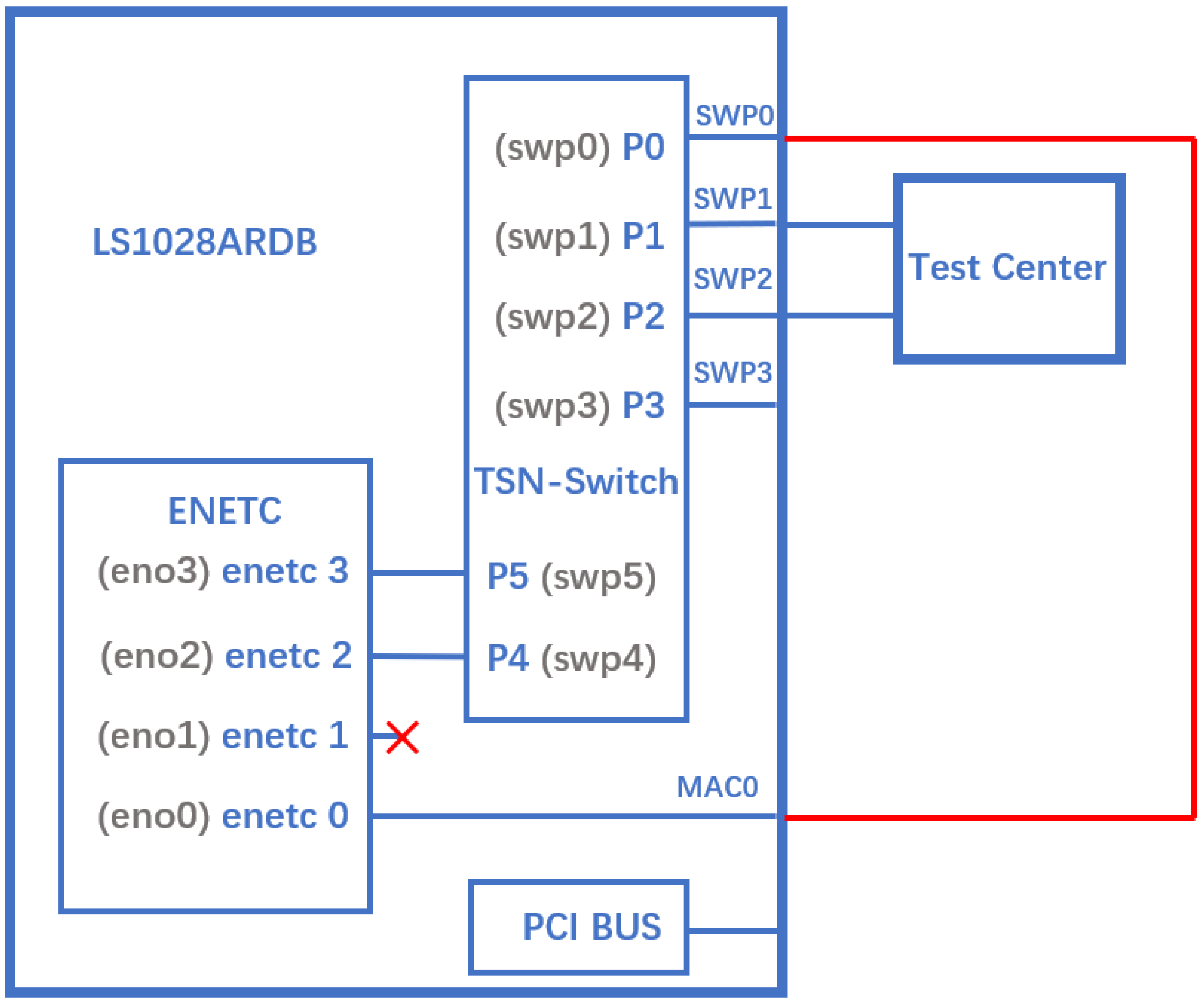


Figure 87. Qbu test

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. 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
 queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 hw 1
```

2. Make sure that the link speed is 1 Gbit/s by using the command:

```
ethtool eno0
```

3. If it is not 1 Gbit/s, set it to 1 Gbit/s 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):

```
ethtool --set-mm swp0 tx-enabled on pmac-enabled on
```

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
/usr/share/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-mm eno0 tx-enabled on pmac-enabled on verify-enabled off  tx-min-
frag-size 60
tc qdisc del dev eno0 root handle 1:
tc qdisc add dev eno0 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 fp E P P P P P P P hw 1
```

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 5.1.4.4.1.

5.1.5.1.4 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.

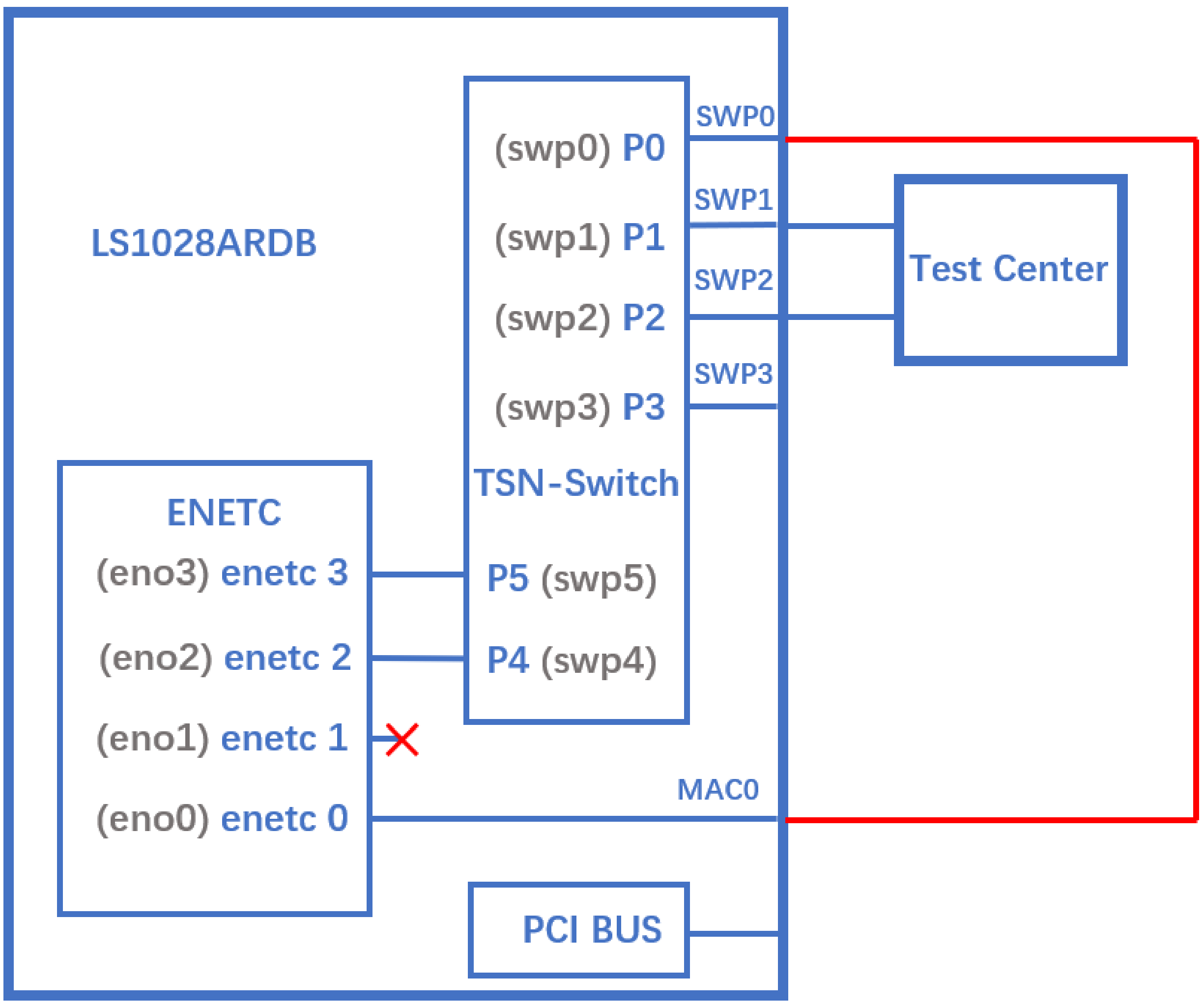


Figure 88. Qci test case setup

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

5.1.5.1.4.1 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
```

5.1.5.1.4.2 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
```

5.1.5.1.4.3 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
   ```

5.1.5.1.4.4 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.

5.1.5.1.4.5 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
```

5.1.5.1.5 Qav

5.1.5.1.5.1 Using tsntool

Figure 89 illustrates the hardware setup diagram for the Qav test.

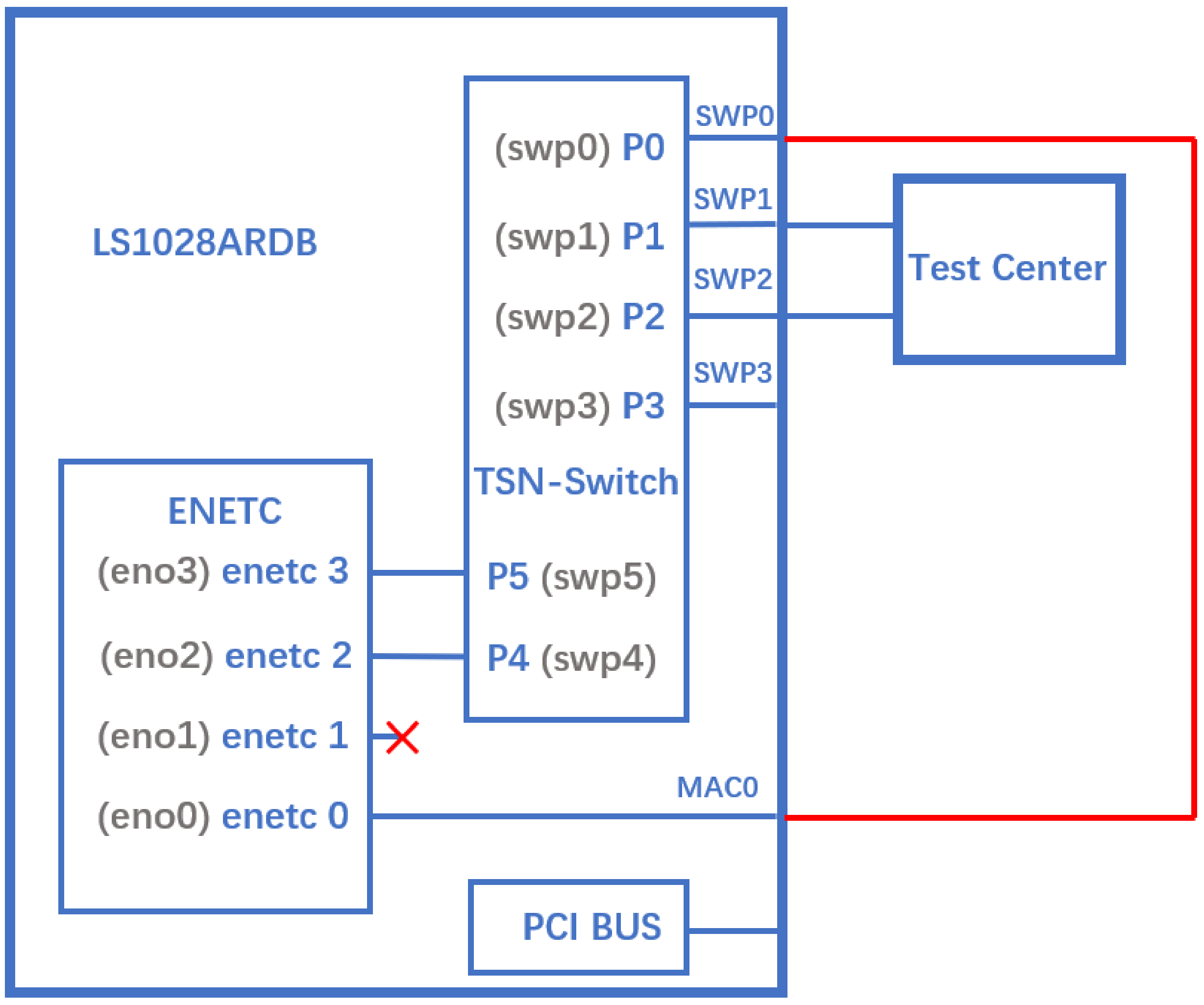


Figure 89. Qav test setup

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.*

1. 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
 queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 hw 1
```

2. Run the following commands:

```
tsntool cbsset --device eno0 --tc 7 --percentage 60
tsntool cbsset --device eno0 --tc 6 --percentage 20
```

3. Check each queue bandwidth (pktgen requires enabling NET_PKTGEN in kernel)

```
/usr/share/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.

```
/usr/share/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.

5.1.5.1.5.2 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 queues
 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@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)
/usr/share/samples/pktgen/pktgen_sample01_simple.sh -i eno0 -q 7 -s 500 -n 20000
/usr/share/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
```

5.1.5.2 TSN configuration on Felix switch

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

5.1.5.2.1 Switch configuration

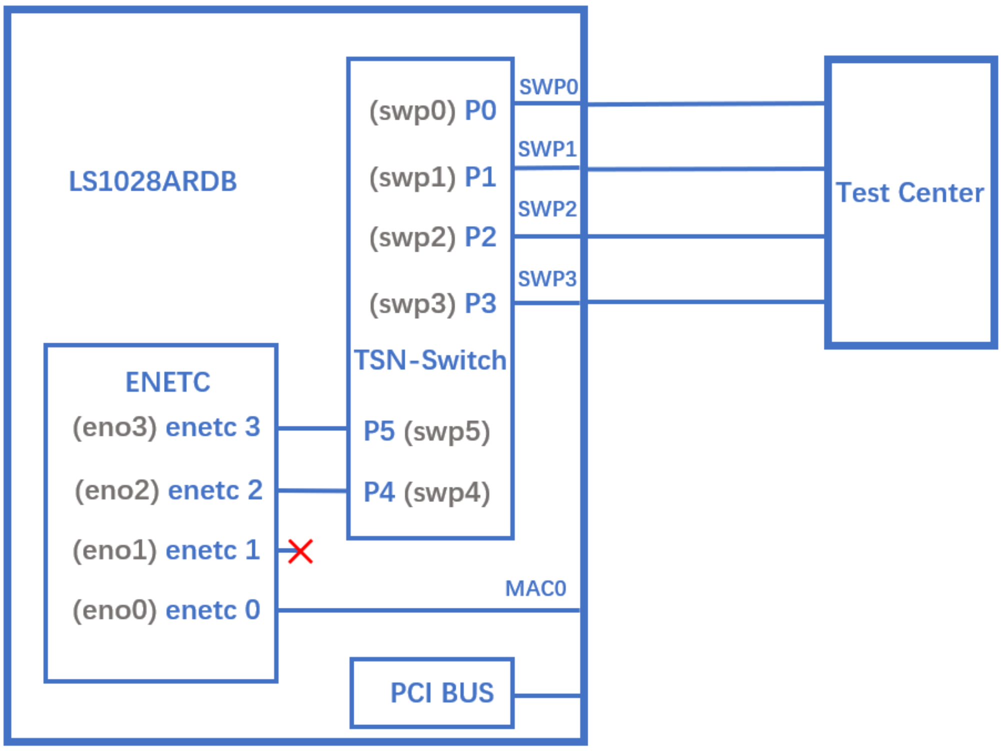


Figure 90. TSN switch configuration

Use the following commands to configure the 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
```

5.1.5.2.2 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 described in Section 5.1.5.2.1. An available IP should be configured on the bridge. Do not configure the 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 20000 protocol 0x88f7 flower skip_sw action
 trap action goto chain 21000
tc filter add dev swp3 ingress chain 20000 protocol ip flower skip_sw dst_ip
 224.0.1.129 action trap action goto chain 21000
tc filter add dev swp3 ingress chain 20000 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
```

5.1.5.2.3 Qbv test setup for LS1028ARDB

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

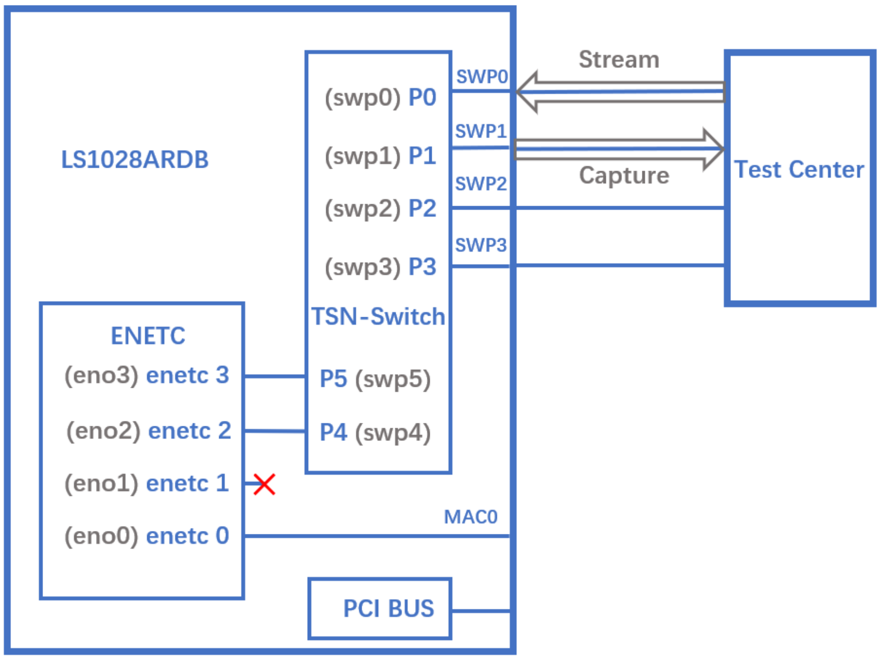


Figure 91. Qbv test

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

For buffer reservation description, refer to https://docs.nxp.com/bundle/UG10143/page/topics/buffer_reservation_watermarks.html.

```
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
```

5.1.5.2.3.1 Using tsntool

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 this parameter is not set, use default 1518Bytes, the least entry time is (1518*8)/1G≈12us.*

5.1.5.2.3.2 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
```

5.1.5.2.4 Qbu

Figure 92 illustrates the setup for performing the Qbu test using the TSN switch.

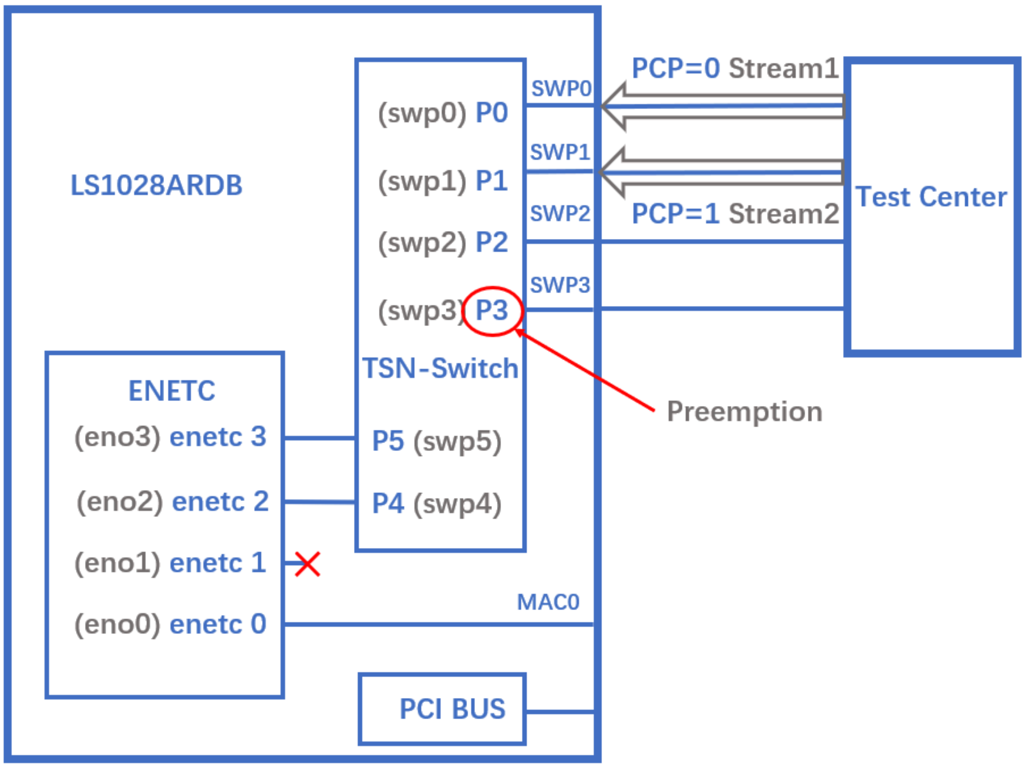


Figure 92. Qbu test on switch

5.1.5.2.4.1 Using tsntool for Qbu test

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 either 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 -I --show-mm swp3 | grep MACMergeFragCountTx
```

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.

5.1.5.2.4.2 Ethtool usage

1. Set queue 1 to be preemptable. There are two ways to set preemptable queues. Users can choose either **tsntool** or **ethtool** to set it.

```
ethtool --set-mm swp3 tx-enabled on pmac-enabled on verify-enabled off  tx-
min-frag-size 60
tc qdisc add dev swp3 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 fp E P E E E E E E hw 1
```

   Explanation:
   - fp: Takes the form of an array (one element per traffic class) with values being 'E' (for express) or 'P' (for preemptible)
   - `tx-min-frag-size`: specifies the least frame bytes that have been transmitted in the fragment. The minimum non-final fragment size is 60, 124, 188, 252.
2. Send two streams from the TestCenter. Set the packet size to be 1500 bytes and bandwidth to be 1 G. Now, check the number of additional mPackets transmitted by the PMAC:

```
ethtool -I --show-mm swp3 | grep MACMergeFragCountTx
```

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 verification. Use two boards to test preemption verification on swp0-3. Refer to Section 5.1.4.4.1.

5.1.5.2.5 Qci

Figure 93 illustrates the Qci test case setup.

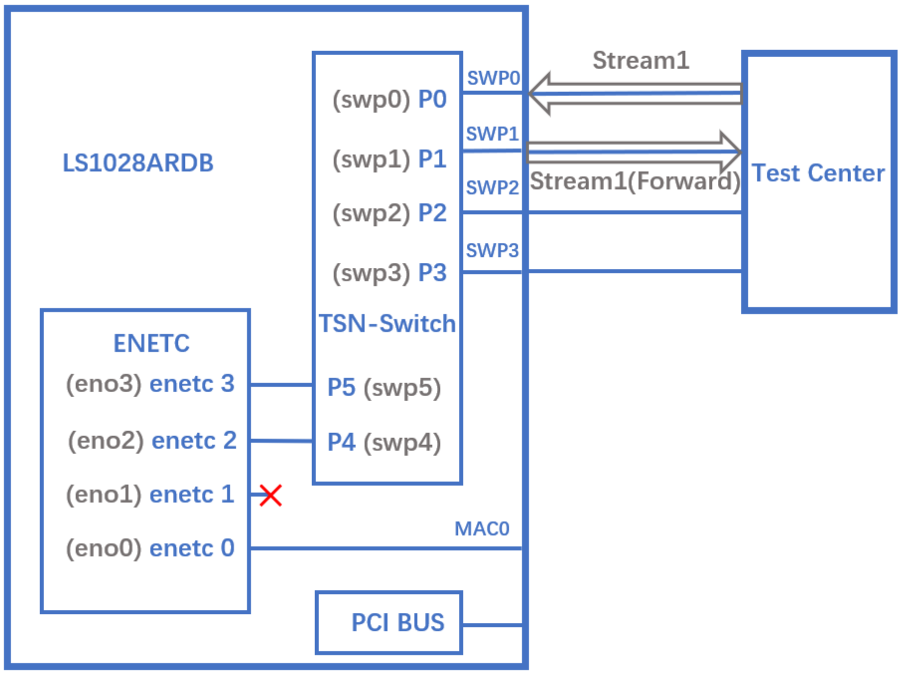


Figure 93. Qci test case

5.1.5.2.5.1 Using Tsntool for stream identification and other tests

The following section describes how to use tsntool for stream identification, stream gate control, SFI maxSDU test, FMI test, and Port-based SFI set.

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`: sets the device port which is the stream forwarded to. If the {destmac, VID} is already learned by the switch, the switch does not care for the device port.
- `nulltagged`: switch only supports `nulltagged=1` mode. Therefore, it is not required to set this parameter.
- `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

Disable the cut-through mode on swp0 and swp1:

```
tsntool> ctset --device swp0 --queue_stat 0
tsntool> ctset --device swp1 --queue_stat 0
```

Use the following command to run this test:

```
tsntool> qcisfiset --device swp0 --index 1 --gateid 1 --priority 0 --flowmeterid
 68 --maxsdu 200
```

Now, send one frame (frame size > 200) on TestCenter.

```
ethtool -S swp1
```

Users can observe that the frame could not pass.

FMI test

Use the following set of commands for the FMI test.

1. Reserve buffer for each queue on ingress port to receive yellow frames(dp=1) in switch.

   ```
   ingressport=0
   for tc in {0..7}; do {
       devlink sb tc bind set pci/0000:00:00.5/$ingressport sb 0 tc $tc type
    ingress pool 0 th 3000
       devlink sb tc bind set pci/0000:00:00.5/$ingressport sb 1 tc $tc type
    ingress pool 0 th 10
   }
   done
   ```
2. Run the command:

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

   Note:

   - *The 'device' in above command can be any one of the switch ports.*
   - *The index of `qcifmiset` must be the same as flowmeterid of `qcisfiset`.*
3. Now, send one stream (rate = 100M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   Note that all frames pass and get all green frames.
4. Now, send one stream (rate = 200M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   Observe that all frames pass and get green and yellow frames.
5. Send one stream (rate = 300M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   Note that not all frames could pass and get green, yellow, and red frames.
6. Send one yellow stream (rate = 100M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   All frames pass and get all yellow frames.
7. Send one yellow stream (rate = 200M) on TestCenter.

   ```
   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.

5.1.5.2.5.2 Tc-flower usage

Figure 94 illustrates the TC-flower-based Qci test case setup.

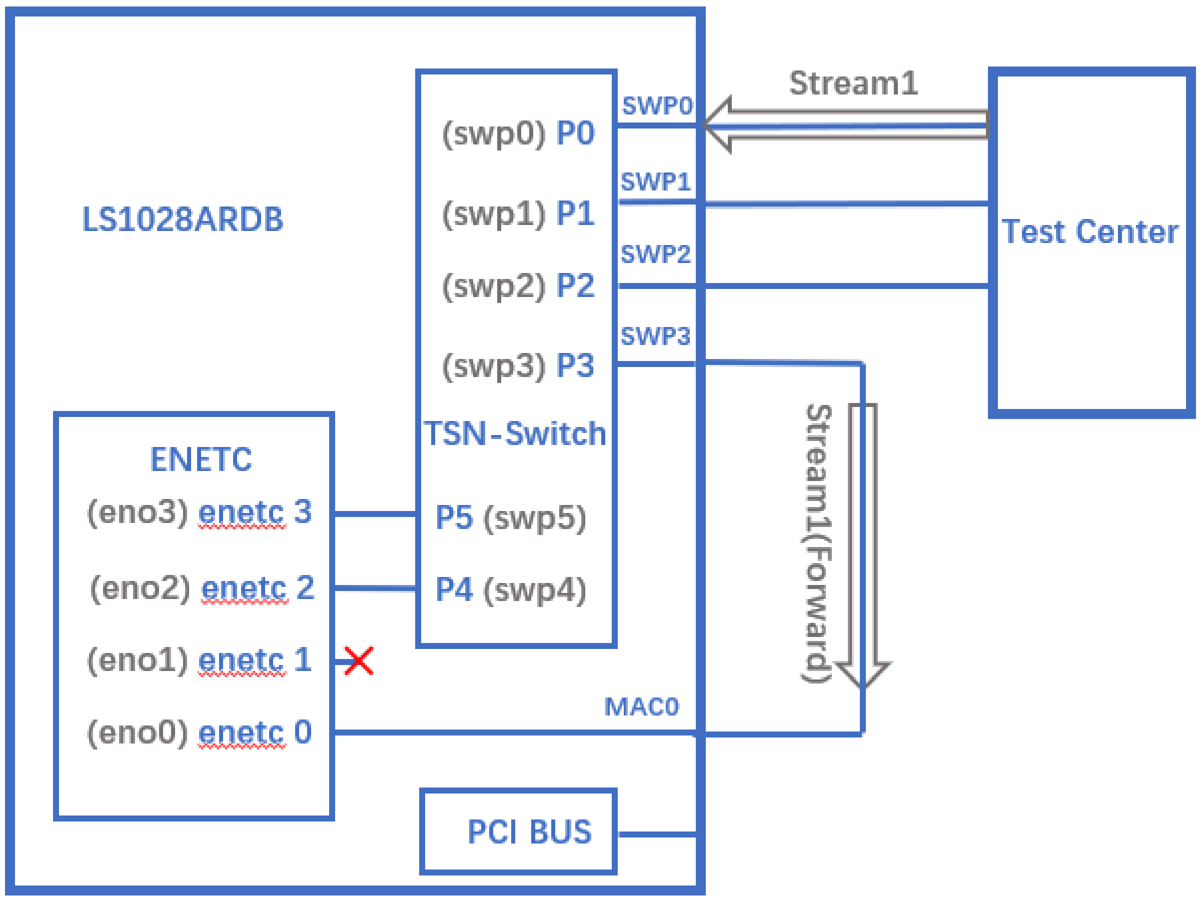


Figure 94. TC-flower based Qci test case

1. Add the MAC address '`CA:9C:00:BC:6D:68`' in the 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 5.4.2.

```
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.*

5.1.5.2.6 Qav

Figure 95 illustrates the Qav test case setup.

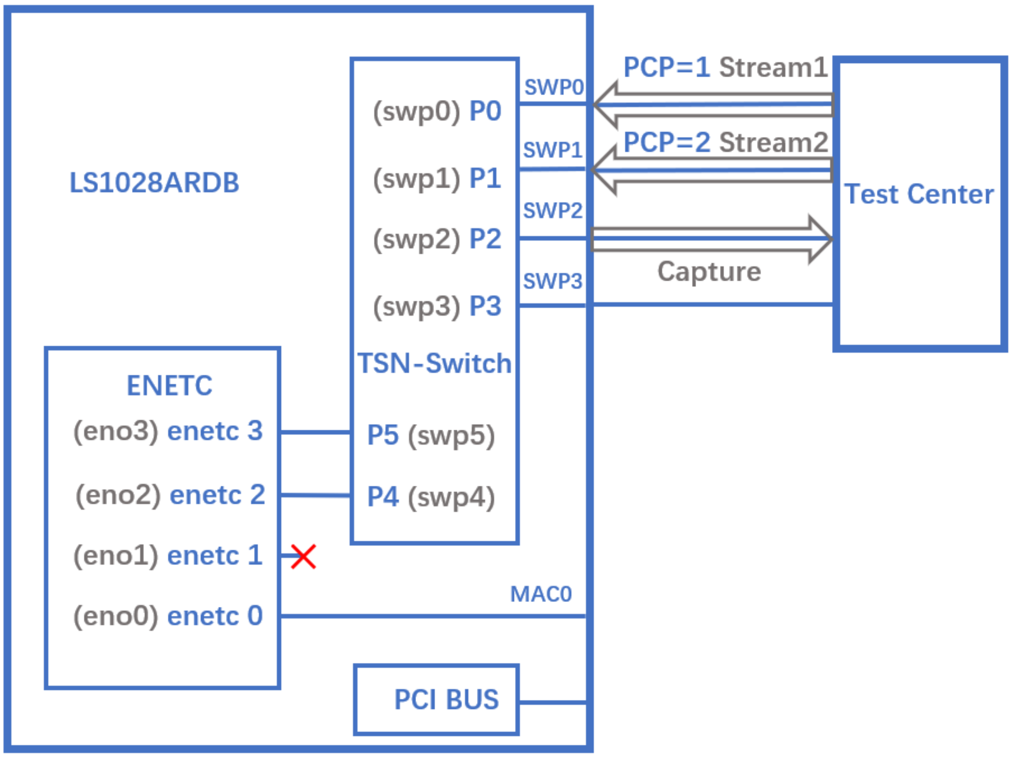


Figure 95. Qav test case

5.1.5.2.6.1 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),…`

5.1.5.2.6.2 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
```

5.1.5.2.7 802.1CB

The Figure 96 describes the test setup for the seamless redundancy test case.

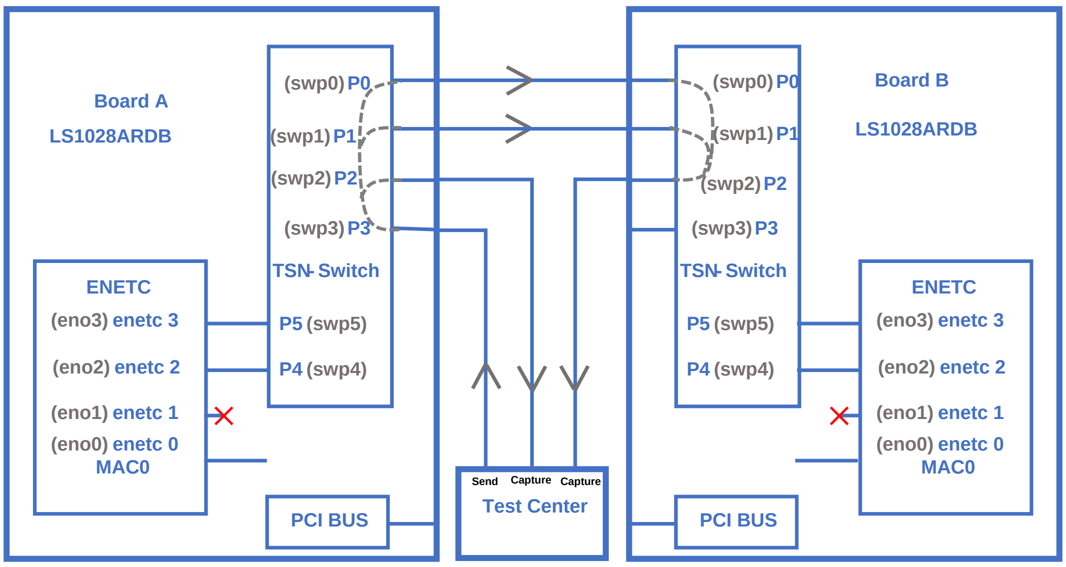


Figure 96. Seamless redundancy test

5.1.5.2.7.1 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.

5.1.5.2.7.2 Sequence Recover test

Use the following steps for the **Sequence Recover** test:

1. On board B, run the following commands:

```
bridge fdb add 7E:A8:8C:9B:41:DD dev swp2 vlan 1 master static
tsntool> cbstreamidset --device swp2 --index 1  --nullstreamid --nulldmac
 0x7EA88C9B41DD --nullvid 1  --streamhandle 1
tsntool> 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 the destination MAC address to be 7E:A8:8C:9B:41:DD.
3. Capture frames from swp2 of board B on the TestCenter, we can get only one frame without sequence tag.

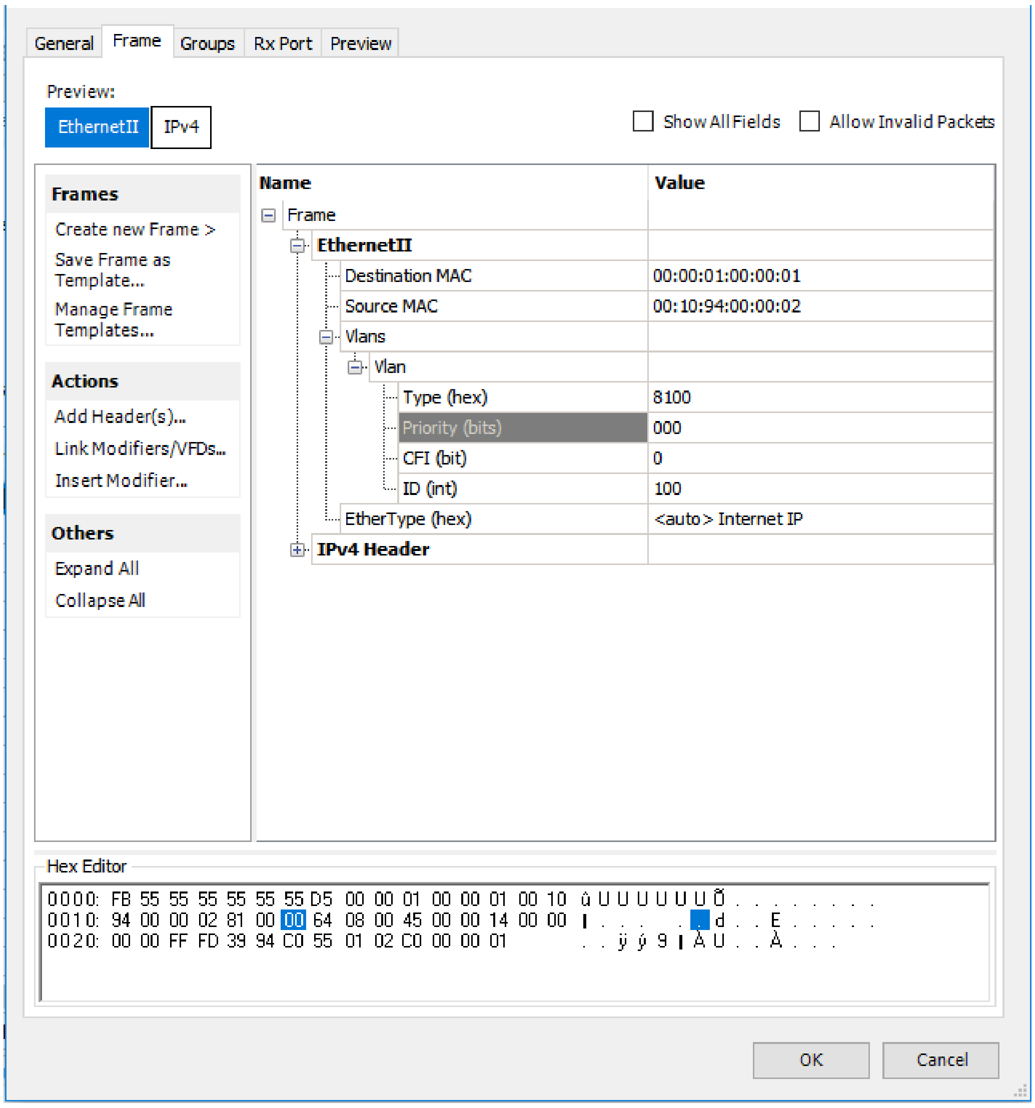
5.1.5.2.8 TSN stream identification

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.

5.1.5.2.8.1 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.



Name	Value
Frame	
EthernetII	
Destination MAC	00:00:01:00:00:01
Source MAC	00:10:94:00:00:02
Vlans	
Vlan	
Type (hex)	8100
Priority (bits)	000
CFI (bit)	0
ID (int)	100
EtherType (hex)	<auto> Internet IP
IPv4 Header	

Hex Editor

```
0000:  FB 55 55 55 55 55 55 D5  00 00 01 00 00 01 00 10  û U U U U U U Õ . . . . . . . .
0010:  94 00 00 02 81 00 00 64  08 00 45 00 00 14 00 00  ▌ . . . . . . d . . E . . . . .
0020:  00 00 FF FD 39 94 C0 55  01 02 C0 00 00 01        . . ÿ ý 9 ▌ À U . . À . . .
```

Figure 97. Using PCP value of Vlan tag

5.1.5.2.8.2 Based on DSCP of ToS tag

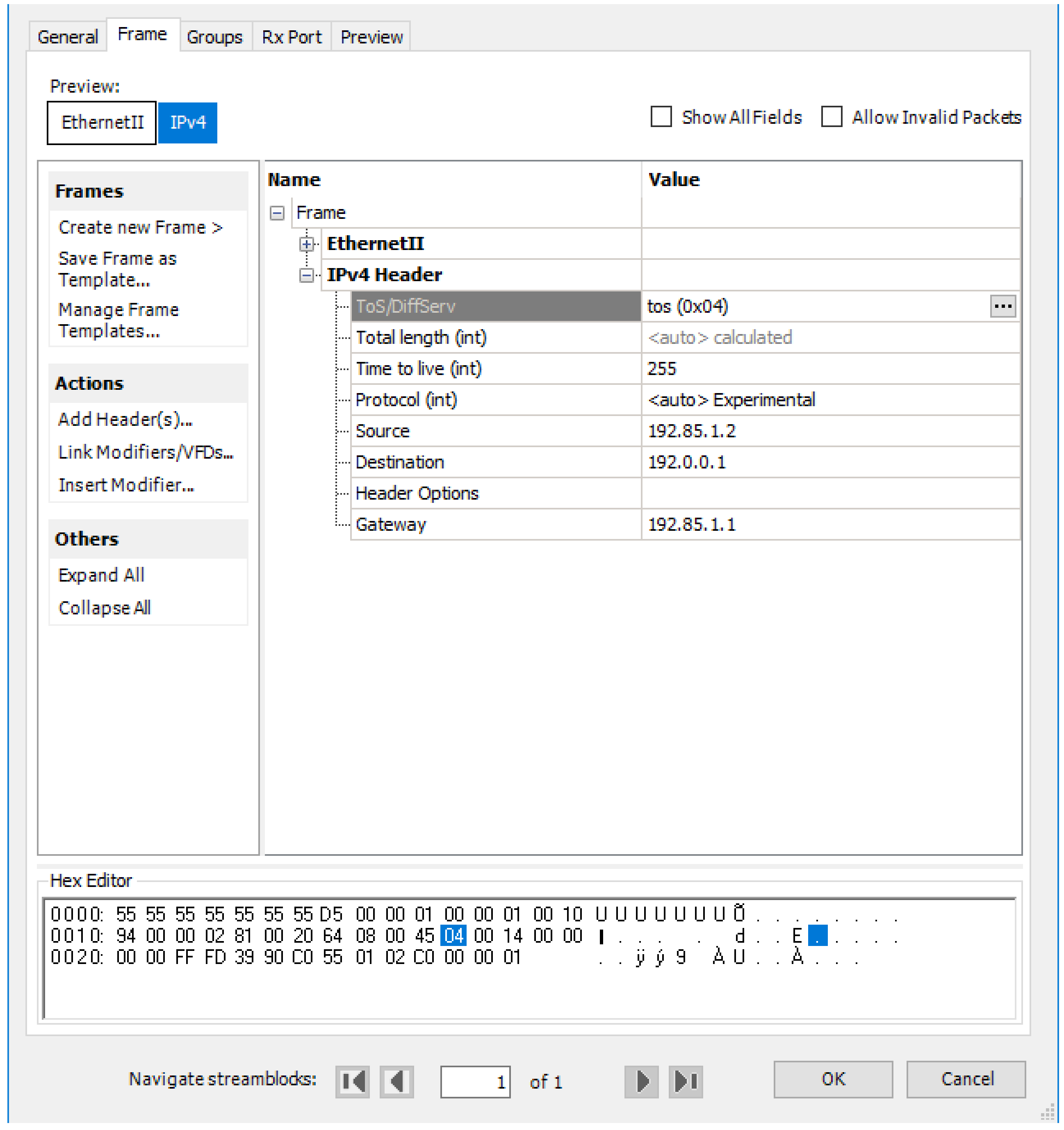
Use the below steps to identify stream based on DSCP value of ToS tag.

1. Map the DSCP value to a specific QoS class using the command below:

   ```
   tsntool> dscpset --device swp0 --index 1 --cos 1 --dpl 0
   ```

   Explanation:

   - `index`: DSCP value of stream, 0-63.
   - `cos`: QoS class which is mapped to.
   - `dpl`: Drop level which is mapped to.
2. Set the DSCP value on TestCenter. DSCP value is the upper six bits of ToS in IP header, set the DSCP value on TestCenter as shown in Figure 98.



Name	Value
Frame	
EthernetII	
IPv4 Header	
ToS/DiffServ	tos (0x04)
Total length (int)	<auto> calculated
Time to live (int)	255
Protocol (int)	<auto> Experimental
Source	192.85.1.2
Destination	192.0.0.1
Header Options	
Gateway	192.85.1.1

Hex Editor

```
0000: 55 55 55 55 55 55 55 D5 00 00 01 00 00 01 00 10 U U U U U U U Õ . . . . . . . .
0010: 94 00 00 02 81 00 20 64 08 00 45 04 00 14 00 00 . . . .   .   d . . E . . . . .
0020: 00 00 FF FD 39 90 C0 55 01 02 C0 00 00 01       . . ÿ ý 9   À U . . À . . .
```

Figure 98. Setting DSCP value on TestCenter

5.1.5.2.8.3 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.*

5.2 GenAVB/TSN stack

This section describes the steps for configuring the GenAVB/TSN stack.

5.2.1 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.

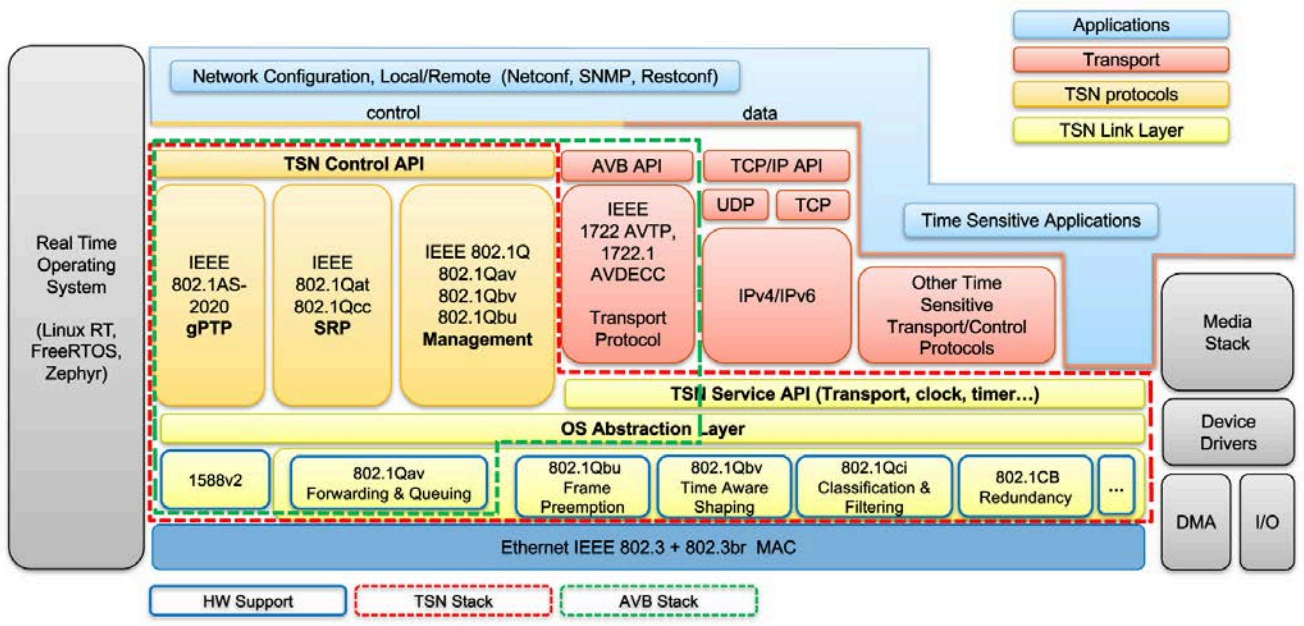

Applications
Transport
TSN protocols
TSN Link Layer
Network Configuration, Local/Remote (Netconf, SNMP, Restconf)
control
data
Real Time Operating System (Linux RT, FreeRTOS, Zephyr)
TSN Control API
IEEE 802.1AS-2020 gPTP
IEEE 802.1Qat 802.1Qcc SRP
IEEE 802.1Q 802.1Qav 802.1Qbv 802.1Qbu Management
AVB API
IEEE 1722 AVTP, 1722.1 AVDECC Transport Protocol
TCP/IP API
UDP
TCP
IPv4/IPv6
Other Time Sensitive Transport/Control Protocols
Time Sensitive Applications
Media Stack
Device Drivers
DMA
I/O
TSN Service API (Transport, clock, timer...)
OS Abstraction Layer
1588v2
802.1Qav Forwarding & Queuing
802.1Qbu Frame Preemption
802.1Qbv Time Aware Shaping
802.1Qci Classification & Filtering
802.1CB Redundancy
...
Ethernet IEEE 802.3 + 802.3br MAC
HW Support
TSN Stack
AVB Stack

Figure 99. GenAVB/TSN Stack Conceptual Diagram

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.

5.2.1.1 GenAVB/TSN Roles

The GenAVB/TSN stack supports the following roles:

- **AVB Endpoint**

AVB Endpoint functionality is provided in i.MX 93, i.MX 8M Plus, i.MX 8DXL, i.MX 8M Mini, and i.MX 6ULL SoCs (using hardware support if available).

- **AVB Bridge**

AVB Bridge functionality requires AVB hardware support, available in LS1028A SoC and SJA1105Q-EVB (paired with i.MX 93 14x14 EVK or i.MX 8DXL EVK).

- **AVB Hybrid**

AVB Hybrid functionality requires AVB hardware support, available in i.MX 93 14x14 EVK SoC paired with a SJA1105Q-EVB.

- **TSN Endpoint**

TSN Endpoint functionality requires TSN hardware support, available on i.MX 95, i.MX 943, i.MX 93, i.MX 8M Plus, and i.MX 8DXL SoCs.

- **TSN Bridge**

TSN Bridge functionality requires TSN hardware support, available in LS1028A SoC.

5.2.1.2 GenAVB/TSN Protocols

The GenAVB/TSN stack includes following protocol stacks running in standalone userspaces processes.

5.2.1.2.1 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.

5.2.1.2.2 SRP stack

The SRP stack implements MRP, MVRP, and MSRP defined in IEEE 802.1Q-2022, 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.

5.2.1.2.3 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.

5.2.1.2.4 AVDECC/Milan Stack

The AVDECC stack implements IEEE 1722.1-2013 standard, and supports Talker, Listener and Controller entities. The stack also implements "Milan Specification v1.2", 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.

5.2.1.2.5 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.

5.2.1.3 User Library

The GenAVB/TSN stack provides a public C API through a user space dynamic library that can be linked into customer applications. These API's cover initialization of the stack, run time configuration as well as AVB and TSN endpoint functionality (AVTP Listener/Talker streaming, Layer 2 network sockets, ...).

5.2.1.4 AVB Kernel Module

Integration of AVB functionality in the Linux kernel is provided by a dedicated kernel module. The main role of this module is to add real time guarantees to the processing of AVTP packets.

5.2.1.5 Supported configurations

GenAVB/TSN stack currently supports the following boards and the associated roles:

- LS1028ARDB: gPTP Time-aware Bridge and SRP Bridge
- i.MX 95 19x19 LPDDR5 EVK: gPTP Time-aware Endpoint station and TSN Endpoint.
- i.MX 943 19x19 LPDDR4 EVK: gPTP Time-aware Endpoint station and TSN Endpoint.
- i.MX 93 EVK: gPTP Time-aware Endpoint station, TSN Endpoint, and AVB Endpoint stack/applications.
- i.MX 93 9x9 LPDDR4 QSB: gPTP Time-aware Endpoint station and TSN Endpoint.
- i.MX 93 14x14 LPDDR4X EVK: gPTP Time-aware Endpoint station, TSN Endpoint and AVB Endpoint stack/ applications. Along with AVB bridge and AVB hybrid stacks when connecting SJA1105Q-EVB Hardware.
- 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 8DXL LPDDR4 EVK: gPTP Time-aware Endpoint station, TSN Endpoint and AVB Endpoint stack/ applications. Along with AVB bridge stack when connecting SJA1105Q-EVB Hardware.
- 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:

- ls1028ardb
- imx8mp-lpddr4-evk
- imx8dxlb0-lpddr4-evk
- imx93evk
- imx93-9x9-lpddr4-qsb
- imx93-14x14-lpddr4x-evk
- imx943-19x19-lpddr4-evk
- imx95-19x19-lpddr5-evk

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

- imx6ull14x14evk
- imx8mm-lpddr4-evk
- imx8mp-lpddr4-evk
- imx8dxlb0-lpddr4-evk
- imx93evk
- imx93-14x14-lpddr4x-evk

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

5.2.1.6 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.

5.2.1.7 TSN endpoint example application

The TSN example application provides example code and re-usable middleware exercising the GenAVB/TSN API. It can be used to exercise and verify the real time behavior of the local system as well as TSN properties of the network between endpoints. Figure 100 shows the TSN application cycle.

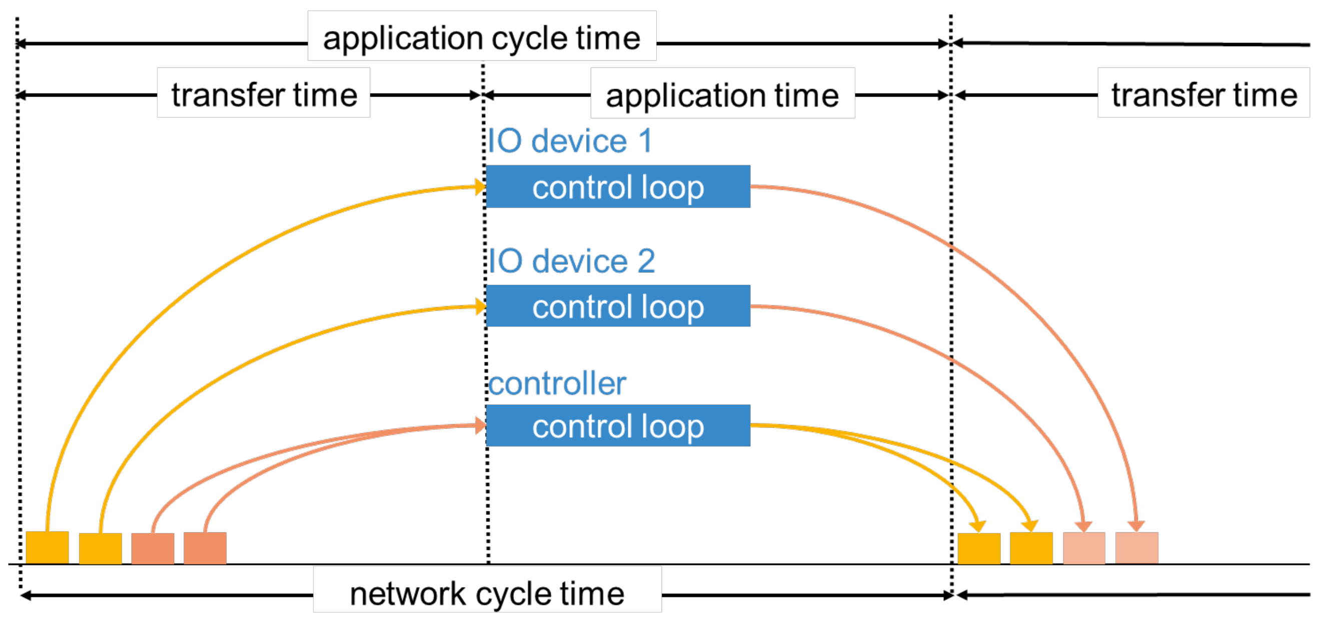


Figure 100. TSN application cycle

The TSN example application implements a control loop similar to industrial use cases requiring cyclic isochronous exchanges over the network.

The TSN endpoints run their application synchronized to a common time grid in the same gPTP domain so that they can send and receive network traffic in a cyclic isochronous pattern (the application cycle time is equal and synchronous to the network cycle time as shown in Figure 100). Currently the cycle is configured with a period of 2 ms, and periods as low as 1 ms have been confirmed to work as well. When the application is scheduled, frames from other endpoints are ready to be read and at the end of the application time frames are sent to other endpoints.

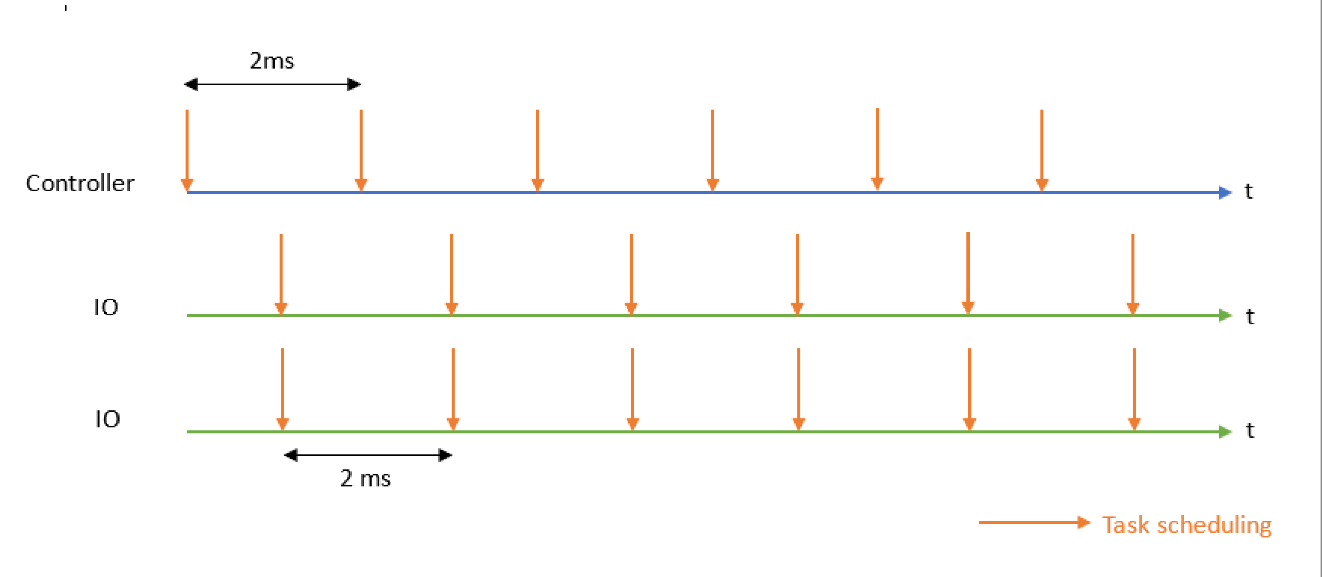


Figure 101. TSN application scheduling

As shown in Figure 101, 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.

5.2.2 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:

```
# tsn.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
```

5.2.3 Use cases description

5.2.3.1 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.

5.2.3.2 AVB Bridge

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

5.2.3.2.1 Requirements

5.2.3.2.1.1 Using LS1028ARDB as an AVB bridge

- Two AVB endpoints
- One AVB bridge (LS1028ARDB)

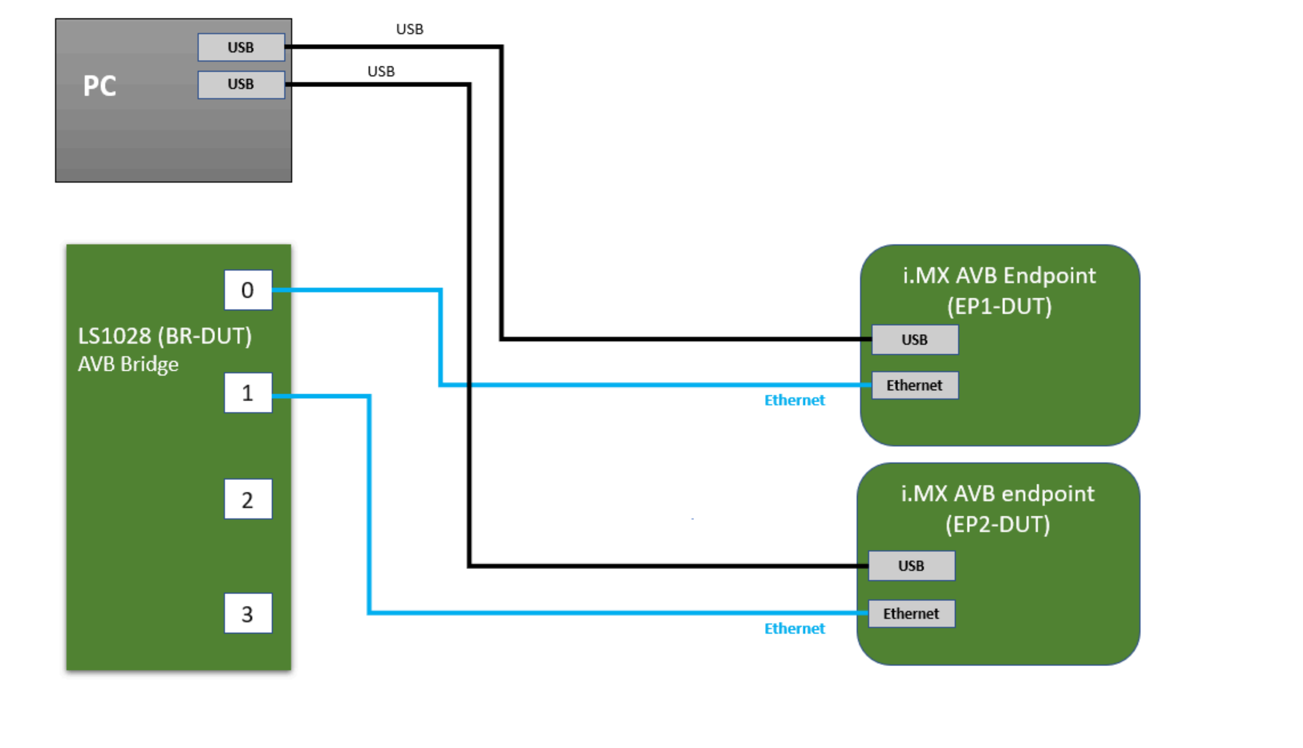


Figure 102. AVB Bridge setup

5.2.3.2.1.2 Using i.MX8DXL EVK with SJA1105Q EVB as AVB bridge

- Two AVB endpoints including:
  – an i.MX 8D XL with an IMX-RMII-PHY TJA1100
  – an i.MX 8M Plus with an Ethernet to BoardR-Reach converter
- One AVB bridge (i.MX8DXL with a SJA1105Q-EVB)

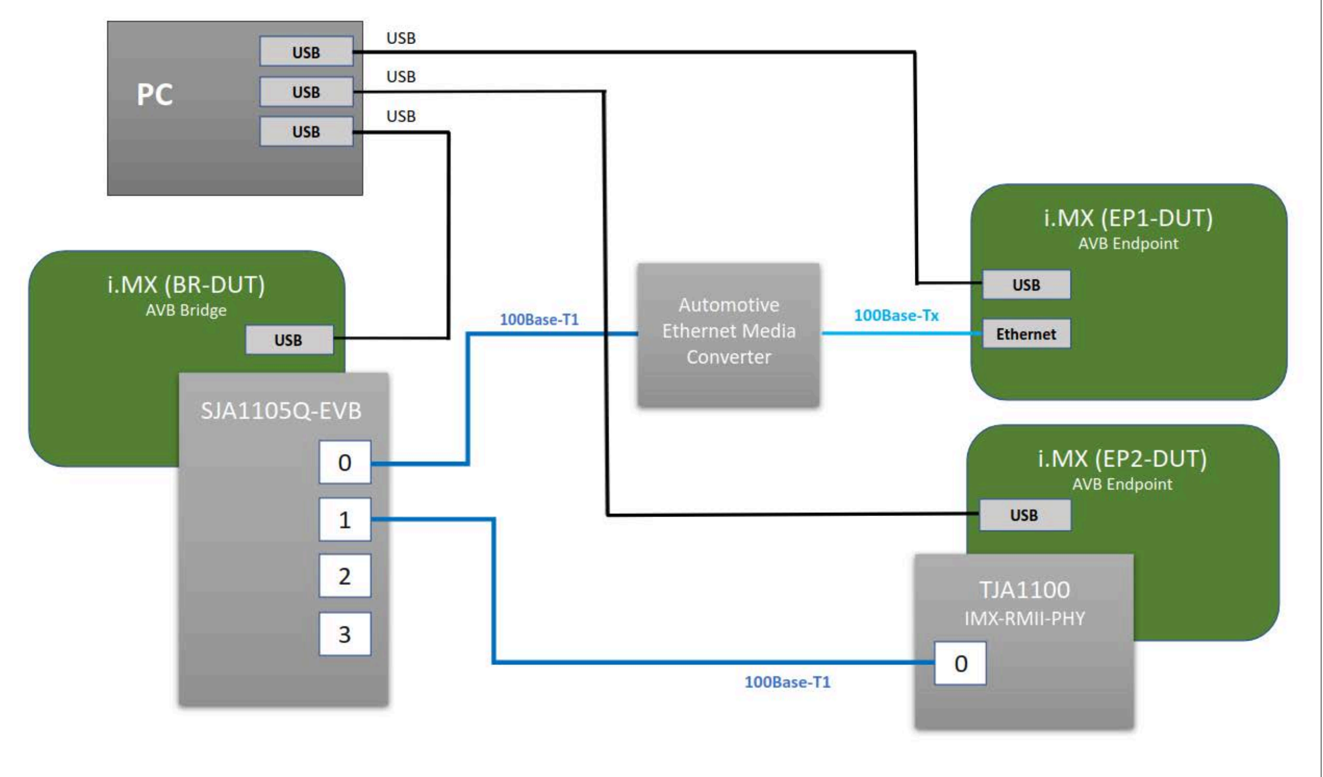


Figure 103. AVB Bridge setup

Setup the device tree:

- On the i.MX 8D XL Bridge with a SJA1105Q-EVB. Use the following device tree in U-boot:

```
setenv fdt_file imx8dxl-evk-enet0-sja1105.dtb
```

- On the i.MX 8DXL Endpoint with a TJA1100. Use the following device tree in U-boot:

```
setenv fdt_file imx8dxl-evk-enet0-tja1100-avb.dtb
```

Configure Leader / Follower role of the PHYs:

- On the Bridge with SJA1105Q-EVB, check the default PHY role for the connected ports:

```
ethtool swpX
```

Note: *The default role is Leader (ethtool reports "forced-master" under "master-slave" cfg).*

- On the endpoint with TJA1100 PHY Daughter card, set the PHY role accordingly (Leader if SJA1105 port reports being a Follower and vice versa):

```
ethtool -s eth0 master-slave forced-slave
```

- On the automotive media converter, set the role using the same method

5.2.3.2.1.3 Using i.MX 93 14x14 EVK with SJA1105Q EVB as AVB bridge

- Two AVB endpoints including the following:
  – an i.MX 8DXL board with an IMX-RMII-PHY TJA1100
  – an i.MX 8M Plus board with an Ethernet to BoardR-Reach converter
- One AVB bridge (i.MX 93 14x14 with a SJA1105Q-EVB connected on ENET2)

Note: *SJA1105Q-EVB is using RGMII interface and the ENET connector on i.MX 93 14x14 EVK is usually set to RMII by default. A Hardware rework may be needed to set the interface to RGMII (mount R267, R272, R275 and R278 and unmount R288 and R293). Refer to board's schematics for more details.*

See Figure 103

Setup the device tree:

- On the i.MX 93 14x14 Bridge with a SJA1105Q-EVB. Use the following device tree in U-boot:

```
setenv fdtfile imx93-14x14-evk-sja1105.dtb
```

- On the i.MX 8DXL Endpoint with a TJA1100. Use the following device tree in U-boot:

```
setenv fdt_file imx8dxl-evk-enet0-tja1100-avb.dtb
```

Configure Leader / Follower role of the PHYs:

- On the Bridge with SJA1105Q-EVB, check the default PHY role for the connected ports:

```
ethtool swpX
```

Note: *Default role is Leader (ethtool will report "forced-master" under "master-slave" cfg)*

- On the endpoint with TJA1100 PHY Daughter card, set the PHY role accordingly (Leader if SJA1105 port reports being a Follower and vice versa):

```
ethtool -s eth0 master-slave forced-slave
```

- On the automotive media converter, set the role using the same method

5.2.3.2.2 AVB network configuration

This topic describes AVB configuration.

5.2.3.2.2.1 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 73. 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 5.2.3.2.3 for the bridge PCP mapping configuration.

5.2.3.2.2.2 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 must 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 5.2.3.2.3 for the bridge qdisc configuration.

5.2.3.2.2.3 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 5.2.3.2.3 for the skb priorities remapping configuration.

5.2.3.2.2.4 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 must 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 5.2.3.2.3 for the bridge vlan configuration.

5.2.3.2.3 Setup preparation

This has two steps described in the following sections.

5.2.3.2.3.1 GenAVB/TSN stack configuration

This configuration must be done once and is saved accross reboots.

Edit the GenAVB/TSN configuration file using the following command at the Linux prompt:

```
# vi /etc/genavb/config
```

And set the GENAVB_TSN_CONFIG correctly as "Bridge AVB".

Note: *It's the default configuration on LS1028A while the Bridge with SJA1105Q must be changed accordingly*

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

- AVB Bridge on LS1028A

In the `/etc/genavb/fgptp-br.cfg`, apply the settings (rxDelayCompensation and txDelayCompensation) described in Table 89 on all bridge ports.

Attention: *The PHY Delay Compensation Values in Table 89 are calibrated for 1 Gbps links. The i.MX AVB endpoints are configured to run by default with 100 Mbps links. These compensation values must be enough to keep pDelay values under 800 ns (propagation time threshold), and therefore the port would still be declared as Capable. If, with these values, the calculated propagation delay is still above 800 ns (or too close to it), adapt them accordingly (increase rxDelayCompensation and/or txDelayCompensation).*

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

- AVB Bridge on SJA1105Q

On AVB bridge setup with SJA1105Q-EVB, the calculated propagation delay is exceeding default propagation delay threshold (800ns) and the link would not be asCapable. To bypass that, increase the neighborPropDelayThreshold on all devices, including the bridge:

```
vi /etc/genvb/fgptp-br.cfg
```

Set the neighborPropDelayThreshold to 8000ns

The same must be done on endpoints connected to the bridge: `/etc/genavb/fgptp.cfg`

5.2.3.2.3.2 Bridge configuration

This configuration must be done after each boot. The user can either enter these commands manually or **execute a ready to use script provided by GenAVB/TSN stack**.

1. If you are using an i.MX 8DXL EVK or i.MX 93 14x14 EVK Bridge, edit the GenAVB/TSN configuration file using the following command at the Linux prompt:

   ```
   # vi /etc/genavb/config
   ```

   And set the configuration GENAVB_TSN_CONFIG to use the config_avb_bridge file:

   ```
   GENAVB_TSN_CONFIG=3
   ```

2. Execute the **automated configuration script** and start the AVB bridge stack:

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

3. Alternatively, configure the bridge manually using the following commands:
   a. Setup bridge forwarding:

      ```
      # ip link add name br0 type bridge
      # ip link set br0 up
      # ip link set master br0 swp0 up
      # ip link set master br0 swp1 up
      # ip link set master br0 swp2 up
      # ip link set master br0 swp3 up
      ```

   b. Establish the PCP to QoS mapping for every port on the bridge using a custom tool (supported only on LS1028A):

      ```
      # pcp_to_qos_map=([0]="1" [1]="0" [2]="6" [3]="7" [4]="2" [5]="3" [6]="4" [7]="5"); \
      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
      ```

   c. Configure the qdiscs and shapers, with the correct handles, and optionally offload the pcp to qos mapping for every external port

      ```
      # pcp_to_qos_map=([0]="1" [1]="0" [2]="6" [3]="7" [4]="2" [5]="3" [6]="4" [7]="5"); \
      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 <HW Offload> ; \
          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
      ```

Note:
<HW offload>*: set to 1 if the driver supports MQPrio PCP mapping offload (SJA1105Q), otherwise 0 (LS1028A).*
The two most important CBS parameters for every port device are:

- *the* ***parent****, which must match the traffic class 6 and 7,*
- *the* ***handle****, which must be 0x9006 and 0x9007.*

The other parameters are initialization values and are overridden by the stack at runtime stream configuration:

- *`offload` is set to 1 to offload the operation to hardware,*
- *`idleslope` and `sendslope` are set depending on stream,*
- *`port bit rates` and the `credit values` are kept at their minimum and maximum values as they do not directly affect the hardware shaping operation.*

d. Setup skb priorities remapping for every external port:

```
# avb_ports="swp0 swp1 swp2 swp3"; \
for port in $avb_ports; do \
    tc qdisc add dev $port clsact; \
    tc filter add dev $port egress basic match 'meta(priority eq 2)' or
 'meta(priority eq 3)' action skbedit priority 0; \
done
```

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

```
# ip link set br0 type bridge vlan_filtering 1; \
avb_ports="swp0 swp1 swp2 swp3"; \
for port in $avb_ports; do \
    bridge vlan add dev $port vid 2 master; \
    bridge link set dev $port mcast_flood off; \
done
```

5. Start the AVB and gPTP stacks:

```
# avb.sh start
```

- Since multicast traffic flooding is now disabled, adding MDB entries for AVDECC (ACMP/ADP) and MAAP protocols multicast addresses is needed. The following commands must 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
```

5.2.3.2.4 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 must 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 must be forwarded correctly to the listener endpoint.

5.2.3.2.4.1 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 5.2.3.5, for more details on gPTP Bridge operation.

5.2.3.2.4.2 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 must 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
```

5.2.3.3 AVB Hybrid

AVB Hybrid use cases and example applications are described in the *GenAVB/TSN Stack AVB Endpoint User Guide* located in Real Time Edge Documentation.

5.2.3.4 gPTP Endpoint

Once the 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 appears 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 the *Grandmaster Parameters* described in section Section 5.2.4.2.2.

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 Delay Mechanism takes the value COMMON_P2P, else the value P2P.

For further details about gPTP logs, refer to section Section 5.2.5.1.

5.2.3.5 gPTP Bridge

LS1028ARDB, i.MX8DXL EVK, or i.MX 93 14x14 EVK paired with a SJA1105Q-EVB can be used as a generic time-aware bridge, connected to other time-aware end stations or bridges.

By default, they do not forward packets if no bridge interface is configured under Linux. Enabling the bridge interface depends on the board used. For example, the configuration of bridge interface is described in section Section 5.1.5.2.1.

Note: *The i.MX8DXL EVK uses eth0 instead of eno2.*

Once the gPTP stack is started, logs can be displayed with the following command:

```
# tail -f /var/log/fgptp-br
```

The og file displays the ports that are connected. One can also observe the ports that are currently communicating a synchronized time and 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, a log such as the following appears 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 the Section 5.2.4.2.2.

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 5.2.5.2.

5.2.3.6 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.

5.2.3.6.1 Requirements

The reference setup for gPTP multiple domains is made of:

- Two gPTP endpoints EP1-DUT and EP2-DUT such as:
  - i.MX 943 19x19 LPDDR4 EVK connected through TSN interface `eth1: enetc`
  - i.MX 95 19x19 LPDDR5 EVK connected through TSN interface `eth0: enetc`
  - i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK or i.MX 943 19x19 LPDDR4 EVK connected through TSN interface `eth1: dwmac`
  - i.MX 8DXL LPDDR4 EVK, i.MX 93 14x14 EVK using an IMXAI2ETH-ATH daughter card or i.MX 93 9x9 LPDDR4 QSB connected through TSN interface `eth0: dwmac`
- One gPTP bridge (LS1028ARDB): BR-DUT

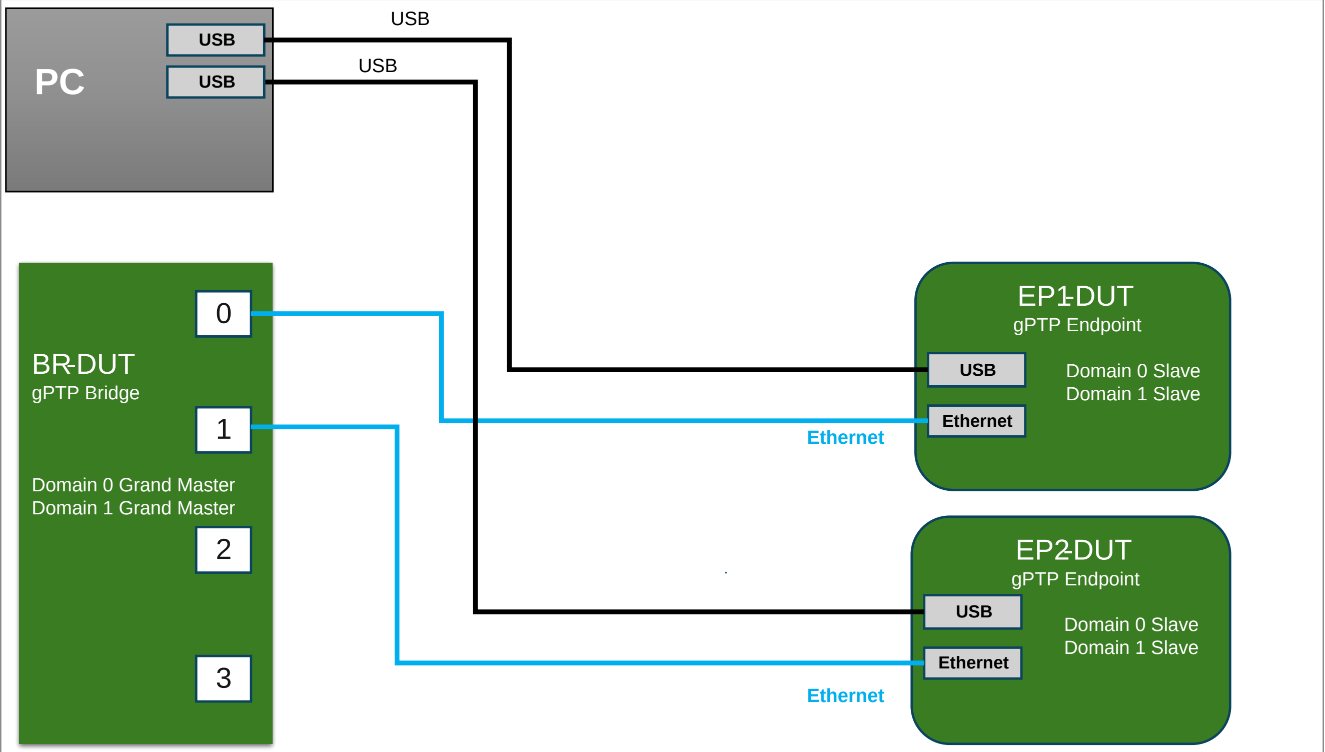


Figure 104. gPTP multiple domains setup

5.2.3.6.2 gPTP Stack Configuration

5.2.3.6.2.1 Domains 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).*

5.2.3.6.2.2 Clocks Configuration

The stack supports two types of virtual clocks as target gPTP clocks when enabling multi-domain support:

- Using internal software clocks
- Or using Linux Virtual PHC devices

Using internal software clocks

The GenAVB/TSN stack has support for userspace based software clocks that are used to track time in multiple domains.

By default, the stack is configured to use a software clock as target gPTP clock for the second domain (domain 20) while keeping the physical PHC device as target gPTP clock for the first domain (domain 0). Refer to `/etc/genavb/system.cfg` for full configuration details.

Using Linux Virtual PHCs on endpoints

Alternatively, the stack can be configured to use Virtual PHC devices on Linux to support gPTP multi domains, as follows:

1. Create two virtual clocks bounded to the PHC device, associated to the network interface, using the following command:
   ```
   # echo 2 > /sys/class/ptp/ptpX/n_vclocks
   ```
   Note:
   Replace ptpX with the right PHC device associated with the used network interface:
   - *`ptp0`: on i.MX 95 19x19 LPDDR5 EVK, i.MX 8DXL LPDDR4 EVK and i.MX 93 9x9 LPDDR4 QSB.*
   - *`ptp1`: on i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 943 19x19 LPDDR4 EVK.*
2. The new virtual clocks devices (ptp) appear in the dmesg log. For example:
   ```
   [  245.771356] ptp ptp0: new virtual clock ptp2
   [  245.772108] ptp ptp0: new virtual clock ptp3
   [  245.772144] ptp ptp0: guarantee physical clock free running
   ```

3. Configure `/etc/genavb/config` to prevent the system configuration file to be overridden by startup scripts:

```
# vi /etc/genavb/config
```

```
CFG_SYSTEM_CFG_NO_OVERRIDE=1
```

4. Assign one virtual clock to each domain in the file `/etc/genavb/system.cfg.tsn`:

```
# vi /etc/genavb/system.cfg.tsn
```

```
[LOGICAL_PORT]
endpoint = ethX

[CLOCK]
endpoint_gptp_0 = /dev/ptpX
endpoint_gptp_1 = /dev/ptp2
endpoint_local = /dev/ptp3
```

Note:
Replace ethX and ptpX with the right network interface and PHC devices:
- `ptp0` *and* `eth0`*: on i.MX 95 19x19 LPDDR5 EVK, i.MX 8DXL LPDDR4 EVK, and i.MX 93 9x9 LPDDR4 QSB.*
- `ptp1` *and* `eth1`*: on i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 943 19x19 LPDDR4 EVK.*
- `ptp1` *and* `eth0`*: on i.MX 93 14x14 EVK using an IMXAI2ETH-ATH daughter card.*
- *endpoint_gptp_0 and endpoint_gptp_1 must be set to each of the new virtual PHCs.*

5.2.3.6.3 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 must 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 must 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
```

5.2.3.7 TSN endpoint sample application

5.2.3.7.1 Requirements

- Two TSN endpoints among:
  - i.MX 8MPlus LPDDR4 EVK connected through TSN interface `eth1: dwmac`
  - i.MX 93 EVK connected through TSN interface `eth1: dwmac`
  - i.MX 93 14x14 EVK connected with IMXAI2ETH-ATH daughter card through TSN interface `eth0: dwmac`
  - i.MX 8DXL LPDDR4 EVK connected through TSN interface `eth0`
  - i.MX 93 9x9 LPDDR4 QSB connected through TSN interface `eth0`
  - i.MX 95 19x19 LPDDR5 EVK connected through TSN interface `eth0`
  - i.MX 943 19x19 LPDDR5 EVK connected through TSN interface `eth1`
  - optionally an i.MX RT1170 EVK
- One TSN bridge (LS1028ARDB)

Note: *The second IO Device is optional.*

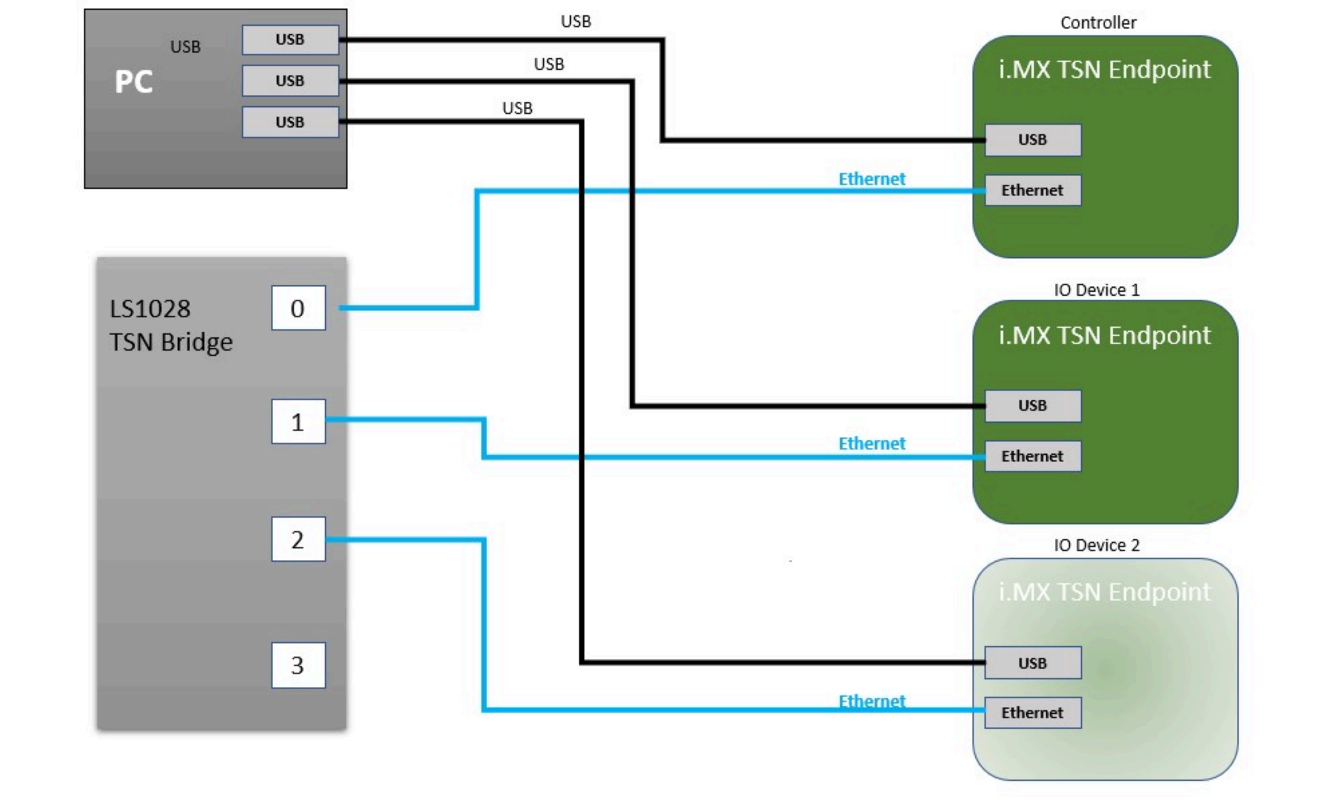


Figure 105. TSN endpoint sample application setup

5.2.3.7.1.1 Using i.MX 8MP EVK as Endpoint TSN

Make sure to update the boot parameter to use the device tree binary that includes the hardware description relative to the i.MX 8MP EVK board:

- Power on the board and stop the automatic boot process by pressing the space bar on the keyboard.
- Enter the following commands at the U-Boot prompt:

```
U-Boot > setenv fdtfile imx8mp-evk.dtb
U-Boot > saveenv
U-Boot > boot
```

The change is saved across reboots.

5.2.3.7.1.2 Using i.MX 8DXL EVK as Endpoint TSN

Make sure to update the boot parameter to use the device tree binary that includes the hardware description relative to the i.MX 8DXL EVK board:

- Power on the board and stop the automatic boot process by pressing the space bar on the keyboard.
- Enter the following commands at the U-Boot prompt:

```
U-Boot > setenv fdtfile imx8dxl-evk.dtb
U-Boot > saveenv
U-Boot > boot
```

The change is saved across reboots.

5.2.3.7.1.3 Using i.MX 93 EVK as Endpoint TSN

Make sure to update the boot parameter to use the device tree binary that includes the hardware description relative to the i.MX 93 EVK board:

- Power on the board and stop the automatic boot process by pressing the space bar on the keyboard.
- Enter the following commands at the U-Boot prompt:

```
U-Boot > setenv fdtfile imx93-11x11-evk.dtb
U-Boot > saveenv
U-Boot > boot
```

The change is saved across reboots.

5.2.3.7.1.4 Using i.MX 93 9x9 EVK as Endpoint TSN

Make sure to update the boot parameter to use the device tree binary that includes the hardware description relative to the i.MX 93 9x9 EVK board:

- Power on the board and stop the automatic boot process by pressing the space bar on the keyboard.
- Enter the following commands at the U-Boot prompt:

```
U-Boot > setenv fdtfile imx93-9x9-qsb.dtb
U-Boot > saveenv
U-Boot > boot
```

The change is saved across reboots.

5.2.3.7.1.5 Using i.MX 93 14x14 EVK as Endpoint TSN

To support TSN endpoint usecase with standard ethernet, an additional Ethernet PHY daughter card (IMXAI2ETH-ATH) is required. see Figure 106.

Note: *The IMXAI2ETH-ATH daughter card is using RGMII interface and the ENET connector on i.MX 93 14x14 EVK is usually set to RMII by default. A Hardware rework may be needed to set the interface to RGMII (mount R267, R272, R275 and R278 and unmount R288 and R293). Refer to board's schematics for more details.*

Figure 106. Atheros Ethernet add on card (IMXAI2ETH-ATH)

Make sure to update the boot parameter to use the device tree binary that includes the hardware description relative to the i.MX 93 14x14 EVK board (Using ENET2):

- Power on the board and stop the automatic boot process by pressing the space bar on the keyboard.
- Enter the following commands at the U-Boot prompt:

```
U-Boot > setenv fdtfile imx93-14x14-evk-imxai2eth-ath.dtb
U-Boot > saveenv
U-Boot > boot
```

The change is saved across reboots.

5.2.3.7.1.6 Using i.MX 95 19x19 LPDDR5 EVK as Endpoint TSN

Make sure to update the boot parameter to use the device tree binary that includes the hardware description relative to the i.MX 95 19x19 LPDDR5 EVK board:

- Power on the board and stop the automatic boot process by pressing the space bar on the keyboard.
- Enter the following commands at the U-Boot prompt:

```
U-Boot > setenv fdtfile imx95-19x19-evk.dtb
U-Boot > saveenv
U-Boot > boot
```

The change is saved across reboots.

5.2.3.7.1.7 Using i.MX 943 19x19 LPDDR4 EVK as Endpoint TSN

Make sure to update the boot parameter to use the device tree binary that includes the hardware description relative to the i.MX 943 19x19 LPDDR4 EVK board:

- Power on the board and stop the automatic boot process by pressing the space bar on the keyboard.

- Enter the following commands at the U-Boot prompt:

```
U-Boot > setenv fdtfile imx943-19x19-evk.dtb
U-Boot > saveenv
U-Boot > boot
```

The change is saved across reboots.

5.2.3.7.2 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 must be set to the right configuration using the file `/etc/genavb/config`:

```
# avb.sh stop_all
# vi /etc/genavb/config
```

Platforms that support both Endpoint AVB and Endpoint TSN (for example i.MX 8MP, i.MX 8DXL and i.MX 93), should have:

```
GENAVB_TSN_CONFIG=1
```

For Endpoint TSN mode, the change from one profile to another is made by modifying the `/etc/genavb/config_tsn` file. This file specifies the application configuration file. `APPS_CFG_FILE` (apps-*.cfg) points to a file containing a demo configuration (application to use, options...). It is parsed by the startup script `avb.sh`.

TSN configuration profile is made of the application configuration profile. The file `/etc/genavb/config_tsn` already lists the supported `cfg` files. Set the `PROFILE` variable to choose the desired configuration profile.

5.2.3.7.3 TSN network configuration

This topic describes TSN configuration.

5.2.3.7.3.1 Streams

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

Table 74. 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)

5.2.3.7.3.2 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 101, 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.

5.2.3.7.4 Setup preparation

One of the TSN endpoint must be configured as the “controller” and the other one as an “IO device”. Both endpoints are connected to the TSN bridge.

Note:

1. *On i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 943 19x19 LPDDR5 EVK : the TSN interface used is `eth1`.*
2. *On i.MX 95 19x19 LPDDR5 EVK, i.MX 8DXL LPDDR4 EVK, i.MX 93 14x14 EVK using an IMXAI2ETH-ATH daughter card and i.MX 93 9x9 LPDDR4 QSB: the TSN interface used is `eth0` rather than `eth1`.*

5.2.3.7.4.1 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.
   Perform the below steps at each boot:
4. Execute the `tsn.sh` script to make sure the gPTP stack is running/synchronized before configuring the Transmission Qdiscs (TC taprio) through the `tsn-app-setup.sh` below
   ```
   tsn.sh start
   ```

Note: On i.MX 95 19x19 LPDDR5 EVK and i.MX 943 19x19 LPDDR5 EVK, make sure to re-setup the Taprio Qdisc (with `tsn-app-setup.sh`) after gPTP has synchronized (and after each gPTP synchronization loss)

5. The system configuration required for the tsn-app can be performed (after setting the correct profile) by using the following command (replace `ethX` with the right TSN network interface):
   - `eth1` on i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 943 19x19 LPDDR5 EVK
   - `eth0` on i.MX 95 19x19 LPDDR5 EVK, i.MX 8DXL LPDDR4 EVK, i.MX 93 14x14 EVK using an IMXAI2ETH-ATH daughter card and i.MX 93 9x9 LPDDR4 QSB

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

   Note: *This script sets many different settings to improve real time system behavior and to setup proper network configuration*
   - *VLAN configuration: the script sets vlan id 2 on the TSN interface as VLAN hardware filtering is enabled by default in kernel.*
   - *Low latency settings on network interface: the script disables coalescing and flow control on the 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 enables threaded NAPI in kernel and isolate tasks processing TSN traffic on a separate CPU core.*
6. Start the TSN demo application by using the following command:

   ```
   avb.sh start
   ```

5.2.3.7.4.2 Preparing the IO devices

Perform the steps below 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.

Perform the below steps at each boot:

1. Execute the `tsn.sh` script to make sure the gPTP stack is running or synchronized before configuring the Transmission Qdiscs (TC taprio) through the `tsn-app-setup.sh` below:

   ```
   tsn.sh start
   ```

   Note: *On i.MX 95 19x19 LPDDR5 EVK and i.MX 943 19x19 LPDDR5 EVK, ensure to perform the re-setup the Taprio Qdisc (with `tsn-app-setup.sh`) after gPTP has synchronized (and after each gPTP synchronization loss).*
2. The system configuration required for the `tsn-app` can be performed (after setting the correct configuration profile) by using the following command (replace `ethX` with the right TSN network interface):
   - `eth1` on i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 943 19x19 LPDDR5 EVK
   - `eth0` on i.MX 95 19x19 LPDDR5 EVK, i.MX 8DXL LPDDR4 EVK, i.MX 93 14x14 EVK using an IMXAI2ETH-ATH daughter card and i.MX 93 9x9 LPDDR4 QSB

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

Note: *This script sets many different settings to improve real time system behavior and to set up the 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 the TSN interface.*
- *Qdiscs and filters: the script sets a 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 the kernel and isolate tasks processing TSN traffic on a separate CPU core.*

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

```
avb.sh start
```

5.2.3.7.4.3 Preparing the Bridge

Refer to section Section 5.1.2 and Section 5.1.5.2.3.2 to configure scheduled traffic on the LS1028ARDB board.

Follow the schedule described in Section "Bridges".

Follow the below steps at each boot:

1. Set up bridge forwarding:

```
# ip link add name br0 type bridge
# ip link set br0 up
# ip link set master br0 swp0 up
# ip link set master br0 swp1 up
# ip link set master br0 swp2 up
# ip link set master br0 swp3 up
```

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
```

5.2.3.7.4.4 Preparing the OPC UA client

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.

5.2.3.7.5 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, it can be observed that the TSN endpoints are synchronized through gPTP and exchange packets at the rate of 500 packets per second (pps). Check the logs to observe this behavior.

5.2.3.7.5.1 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 5.2.3.4 for more details):

```
gptp_stats_dump: Port(0) domain(0,0) : Role: Slave Link : Up AS_Capable: Yes
 DelayMechanism: P2P
```

If the device is Grandmaster, the role field must be “Master”. Otherwise it must be “Slave”. The line appears periodically, but the role must not change over time, except for significant events (such as a cable disconnection).

5.2.3.7.5.2 Baseline tsn-app operation

If the TSN endpoint sample application is running correctly and receiving valid packets, the following points must be verified in the `tsn_app` log file (refer to Section 5.2.5.4 for more details).

The following line must appear at regular intervals:

```
socket_stats_print : link up
```

The 'valid frames' counter must 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 must not increment. However, it is normal to have nonzero values as below. These values occur because of the startup period when gPTP or the remote tsn-app endpoint (or both) are not 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.

5.2.3.7.5.3 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
```

5.2.3.7.5.4 Scheduled traffic evaluation with TX best-effort traffic

1. Connect a PC to the 4th port of the LS1028ARDB switch (swp3).
2. Run iperf3 in server mode on the PC (replace ethX by the PC interface connected to the LS1028):

```
# ifconfig ethX 192.168.1.10 up
# iperf3 -s &
# iperf3 -s -p 5202 &
# iperf3 -s -p 5203 &
# iperf3 -s -p 5204 &
```

3. Run iperf3 in client mode on the controller (replace ethX by the controller interface connected to the LS1028):

```
# ifconfig ethX 192.168.1.80
# taskset b iperf3 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
# taskset b iperf3 -p 5202 -c 192.168.1.10 -u -b 0 -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 statistics in the tsn-app log files (a 2nd terminal may have to be opened through SSH). The values should match the table below (in µs):

Table 75. Statistics of scheduled traffic evaluation on LS1028ARDB

Parameter	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

5.2.3.7.5.5 Scheduled traffic evaluation with RX best-effort traffic

Note:

On SoCs using EnetQos/dwmac as ethernet controller, part of the TSN traffic (untagged gPTP 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.

On SoCs using ENETC controller (e.g i.MX 95 and i.MX 943), the RX classification is able to streer gPTP traffic to different hardware queues than the best effort untagged traffic.

1. Connect a PC to the 4th port of the LS1028ARDB switch (swp3).
2. Run iperf3 in server mode on the controller (replace ethX by the controller interface connected to the LS1028):

```
# ip link add link ethX name ethX.5 type vlan id 5
# ifconfig ethX.5 192.168.5.80 up
# taskset b iperf3 -s &
# taskset b iperf3 -s -p 5202 &
# taskset b iperf3 -s -p 5203 &
# taskset b iperf3 -s -p 5204 &
```

3. Run iperf3 in client mode on the PC (replace ethX by the PC interface connected to the LS1028):

```
# 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):

Table 76. Statistics displayed in the tsn-app log file

Parameter name	Min	Mean	Max	SStddev^2
Sched err (controller)	9	13	26	
Processing time (controller)	25	33	70	
Traffic latency (controller and IO device)	503	503	503	<130000

5.2.3.7.5.6 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.
   In the sample command below, replace `ethX` with the right TSN network interface:
   - `eth1` on i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 943 19x19 LPDDR5 EVK
   - `eth0` on i.MX 8DXL LPDDR4 EVK, i.MX 95 19x19 LPDDR5 EVK, i.MX 93 14x14 EVK using an IMXAI2ETH-ATH daughter card and i.MX 93 9x9 LPDDR4 QSB

```
# tc qdisc del dev ethX root
#tc qdisc replace dev ethX 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.
   In the sample command below, replace `ethX` with the right TSN network interface:
   - `eth1` on i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK and i.MX 943 19x19 LPDDR5 EVK
   - `eth0` on i.MX 8DXL LPDDR4 EVK, i.MX 95 19x19 LPDDR5 EVK, i.MX 93 14x14 EVK using an IMXAI2ETH-ATH daughter card and i.MX 93 9x9 LPDDR4 QSB

   ```
   # tc qdisc del dev ethX root
   #tc qdisc replace dev ethX 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.

5.2.3.7.5.7 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.MX 943 19x19 LPDDR5 EVK, i.MX 95 19x19 LPDDR5 EVK, i.MX 8M Plus LPDDR4 EVK, i.MX 8DXL LPDDR4 EVK, i.MX 93 EVK, or i.MX 93 9x9 LPDDR4 QSB board to use `AF_XDP` sockets.

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

   ```
   # 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 TSN interface. This step can be done at any time, even if the TSN sample application is still running with its default configuration, as long as it is done before restarting it in AF_XDP mode.

   ```
   # ip l set dev ethX 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 must 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 must be ignored. The tables below summarize typical values (in µs), on a setup using a 1 ms period.

Table 77. Timing statistics without any concurrent traffic

Parameter name	Min	Mean	Max
Sched err (controller)	6	7	16
Processing time (controller)	10	13	19

Table 77. Timing statistics without any concurrent traffic*...continued*

Parameter name	Min	Mean	Max
Total time (controller)	16	20	33

Table 78. Timing statistics with TX best-effort traffic

Parameter name	Min	Mean	Max
Sched err (controller)	18	25	51
Processing time (controller)	21	26	52
Total time (controller)	42	51	108

Table 79. Timing statistics with RX best-effort traffic

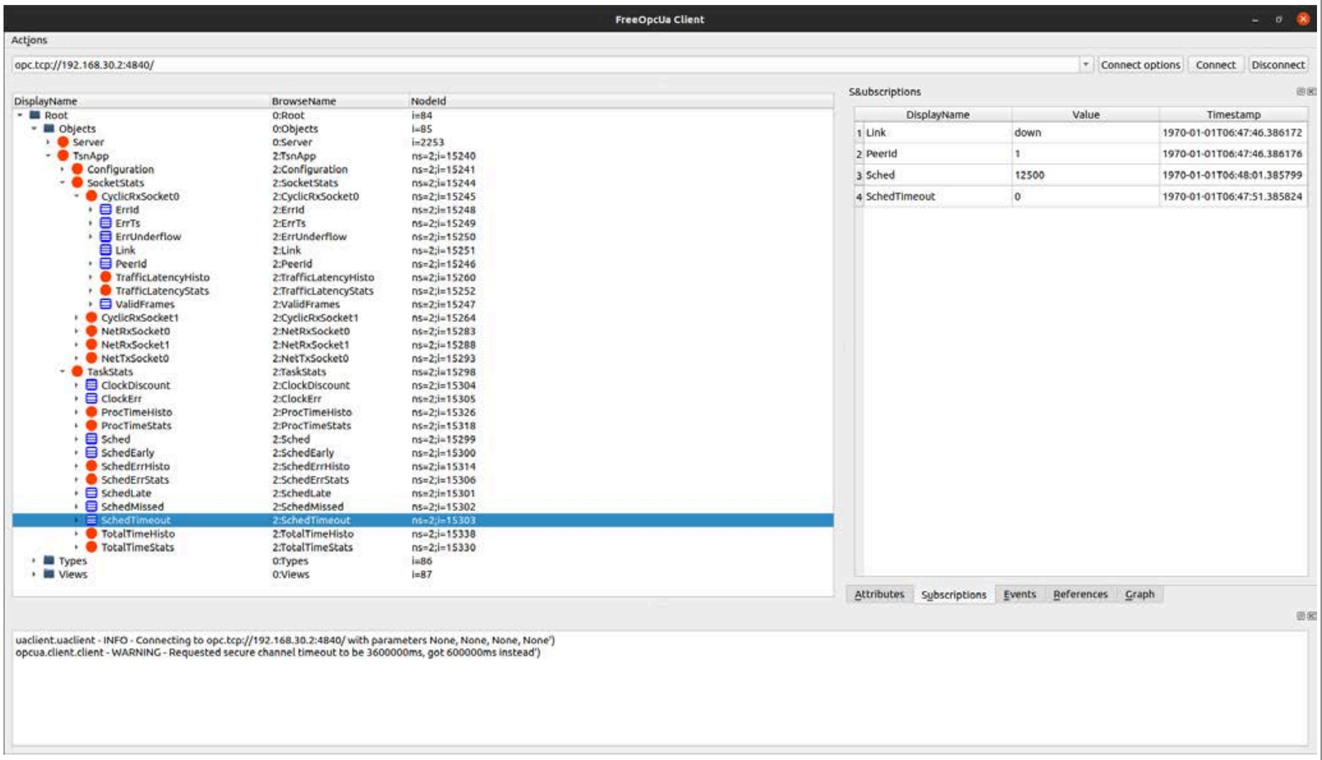
Parameter name	Min	Mean	Max
Sched err (controller)	7	9	34
Processing time (controller)	8	11	25
Total time (controller)	15	21	55

5.2.3.7.5.8 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 the below screenshot using FreeOPCUA GUI client:



opc.tcp://192.168.30.2:4840/

```
DisplayName                     BrowseName                NodeId
▾ Root                          0:Root                    i=84
  ▾ Objects                     0:Objects                 i=85
    ▸ Server                    0:Server                  i=2253
    ▾ TsnApp                    2:TsnApp                  ns=2;i=15240
      ▸ Configuration           2:Configuration           ns=2;i=15241
      ▾ SocketStats             2:SocketStats             ns=2;i=15244
        ▾ CyclicRxSocket0       2:CyclicRxSocket0         ns=2;i=15245
          ▸ ErrId               2:ErrId                   ns=2;i=15248
          ▸ ErrTs               2:ErrTs                   ns=2;i=15249
          ▸ ErrUnderflow        2:ErrUnderflow            ns=2;i=15250
            Link                2:Link                    ns=2;i=15251
          ▸ PeerId              2:PeerId                  ns=2;i=15246
          ▸ TrafficLatencyHisto 2:TrafficLatencyHisto     ns=2;i=15260
          ▸ TrafficLatencyStats 2:TrafficLatencyStats     ns=2;i=15252
          ▸ ValidFrames         2:ValidFrames             ns=2;i=15247
        ▸ CyclicRxSocket1       2:CyclicRxSocket1         ns=2;i=15264
        ▸ NetRxSocket0          2:NetRxSocket0            ns=2;i=15283
        ▸ NetRxSocket1          2:NetRxSocket1            ns=2;i=15288
        ▸ NetTxSocket0          2:NetTxSocket0            ns=2;i=15293
      ▾ TaskStats               2:TaskStats               ns=2;i=15298
        ▸ ClockDiscount         2:ClockDiscount           ns=2;i=15304
        ▸ ClockErr              2:ClockErr                ns=2;i=15305
        ▸ ProcTimeHisto         2:ProcTimeHisto           ns=2;i=15326
        ▸ ProcTimeStats         2:ProcTimeStats           ns=2;i=15318
        ▸ Sched                 2:Sched                   ns=2;i=15299
        ▸ SchedEarly            2:SchedEarly              ns=2;i=15300
        ▸ SchedErrHisto         2:SchedErrHisto           ns=2;i=15314
        ▸ SchedErrStats         2:SchedErrStats           ns=2;i=15306
        ▸ SchedLate             2:SchedLate               ns=2;i=15301
        ▸ SchedMissed           2:SchedMissed             ns=2;i=15302
        ▸ SchedTimeout          2:SchedTimeout            ns=2;i=15303
        ▸ TotalTimeHisto        2:TotalTimeHisto          ns=2;i=15338
        ▸ TotalTimeStats        2:TotalTimeStats          ns=2;i=15330
  ▸ Types                       0:Types                   i=86
  ▸ Views                       0:Views                   i=87
```

	DisplayName	Value	Timestamp
1	Link	down	1970-01-01T06:47:46.386172
2	PeerId	1	1970-01-01T06:47:46.386176
3	Sched	12500	1970-01-01T06:48:01.385799
4	SchedTimeout	0	1970-01-01T06:47:51.385824

```
uaclient.uaclient - INFO - Connecting to opc.tcp://192.168.30.2:4840/ with parameters None, None, None, None')
opcua.client.client - WARNING - Requested secure channel timeout to be 3600000ms, got 600000ms instead')
```

Figure 107. FreeOPCUA GUI client

5.2.4 Configuration files

5.2.4.1 System

The system configuration files (`system.cfg.<configuration>` depend on configuration: Endpoint AVB, Endpoint TSN, Bridge, or Hybrid AVB). These files are located under `/etc/genavb/` and list the system network interface names, PTP hardware clock device names, XDP queues configuration and the Network protocols supported modes. On script startup, a symlink with the name `/etc/genavb/system.cfg` is created to the right configuration file and is used by the stack. The default values are used if the configuration file or the option key are missing. The values in the installed file might also be required to be updated to match the system configuration.

Table 80. 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 must be set to “off”
Endpoint Hybrid Interface	`endpoint_hybrid_port`	off	Endpoint network interface name connected to the on-board bridge. Only valid for hybrid configuration, otherwise must be set to “off”

Table 80. Logical ports*...continued*

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
Bridge Name	`bridge`	br0	Bridge name. Only valid for bridge and hybrid package, otherwise must be set to “off”
Bridge 0 Interfaces	`bridge_0`	swp0, swp1, swp2, swp3, swp_cpu	Bridge 0 network interface names (comma separated). Only valid for bridge and hybrid package, otherwise must be set to “off”
Bridge Hybrid Interface	`bridge_hybrid_port`	off	Bridge network interface name connected to the on-board endpoint. Only valid for hybrid configuration, otherwise must be set to “off”

Table 81. 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
Endpoints gPTP domain 0 target clock	`endpoint_gptp_0`	`/dev/ptp0`	List of PTP devices for each endpoint's target clock for domain 0. Only valid for endpoint package.
Endpoints gPTP domain 1 target clock	`endpoint_gptp_1`	`sw_clock`	List of PTP devices for each endpoint's target clock for domain 1. 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.

Table 82. XDP

This section lists HW queues indexes for XDP packets.

Name	Key	Default value	Description
Endpoint XDP receive queues	`endpoint_queue_rx`	`0, 0`	Endpoint receive HW queues (one per port, comma seperated) for XDP packets.
Endpoint XDP transmit queues	`endpoint_queue_tx`	`1, 1`	Endpoint transmit HW queues (one per port, comma seperated) for XDP packets.

Table 83. Net Modes

This section lists the network modes used by the stack (TSN and AVB processes).
Network modes are configured per network protocol:
- *AVB: using custom network sockets through the AVB kernel module.*
- *STD: using linux standard API sockets.*
- *XDP: using AF_XDP sockets (only available using sockets API in shared library).*

Name	Key	Default value	Supported values	Description
AVTP net mode	`avtp_net_mode`	`avb`	`avb, std`	Net mode for the AVTP protocol.
AVDECC net mode	`avdecc_net_mode`	`avb`	`avb, std`	Net mode for the AVDECC protocol.
Endpoint SRP net mode	`endpoint_srp_net_mode`	`avb`	`avb, std`	Net mode for the SRP protocol on the Endpoint.
Bridge SRP net mode	`bridge_srp_net_mode`	`std`	`avb, std`	Net mode for the SRP protocol on the Bridge.
Endpoint GPTP net mode	`endpoint_gptp_net_mode`	`avb`	`avb, std`	Net mode for the GPTP protocol on the Endpoint.
Bridge GPTP net mode	`bridge_gptp_net_mode`	`std`	`avb, std`	Net mode for the GPTP protocol on the Bridge.

5.2.4.2 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`)

5.2.4.2.1 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 Grandmaster 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) must 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 time receiver back to the Grandmaster.

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 84. 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. (This setting is ignored in case of the 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.

Table 84. General parameters*...continued*

General configuration parameters[1]

Name	Key	Default value	Range	Description
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.

5.2.4.2.2 Grandmaster parameters

This section defines the native Grandmaster capabilities of a time-aware system (see IEEE 802.1AS-2020 - 8.6.2 PTP Instance attributes). Grandmaster 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 Grandmaster is elected dynamically by the BTCA. 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). The parameters priority1, priority2, clockClass, clockAccuracy and offsetScaledLogVariance are used by the Best Time Transmitter Clock algorithm to determine which of the Grandmaster capable node within the gPTP domain has the highest priority/quality. Note that the lowest value for these parameters matches the highest priority/quality.

Note: *The parameters in this section are configurable for all supported domains.*

Table 85. Grandmaster capabilities parameters

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 a time receiver.
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

Table 85. Grandmaster capabilities parameters*...continued*

Name	Key	Default value	Range	Description
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

5.2.4.2.3 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 can update its nvram database and the stored value is 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 86. Automotive parameters

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

5.2.4.2.4 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 87. Timing parameters

Name	Key	Default value	Value and Range	Description
Initial pdelay request interval value	initialLogPdelayReqInterval	0	0 to 3	• Sets 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	• Sets 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	• Sets 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	• Sets pdelay request transmit interval used during normal operation state. • Expressed in log2 unit (default 0 -> 1s).
Operational sync interval value	operLogSyncInterval	-3	-5 to 0	• Sets sync transmit interval used during normal operation state. • Expressed in log2 unit (default -3 -> 125ms).

5.2.4.2.5 PORTn

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

Table 88. 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 time transmitter clock selection functions of the port should be used (ref. 802.1AS-2011, sections 14.6.4 and 10.2.4.12).

Table 88. Port related parameters*...continued*

Name	Key	Default value	Value & Range	Description

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. In general, if a port is using the Common Mean Link Delay Service (CMLDS), the DelayMechanism must be set to 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 89. PHY Delay Compensation Values

Board Type	PHY Name	Speed (Mbit/s)	Port	rxDelay Compensation	txDelay Compensation
LS1028ARDB	VSC8514	1000	ENETC2	274	349
i.MX 8M Plus EVK	RTL8211F	1000	ENET2	569	184
i.MX 8M Plus EVK	RTL8211F	100	ENET2	748	640

Note: *The table indicates computed values but it is recommended to use a 50 ns propagation delay margin to avoid negative propagation delay measurements due to propagation delay jitter.*

5.2.4.3 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 90. 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 91. MSRP

This section lists MSRP parameters.

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

5.2.5 Log files

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

5.2.5.1 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 Grandmaster 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 Grandmaster capabilities are reported upon new Grandmaster selection. *Root Identity* represents the clock ID of the currently selected Grandmaster. *Priority1, Priority2, Class* and *Accuracy* describe the clock quality of the selected Grandmaster. Finally, the *Source Port Identity* of the peer time transmitter port (for example, the bridge port the local time receiver 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 Grandmaster selection (SYNCHRONIZED) or when no Grandmaster 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 time transmitter. *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 time receiver and time transmitter entities:

Table 92. Port statistics displayed on time receiver and time transmitter entities

Receive counters	Description
`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
Transmit counters	**Description**
`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

Table 92. Port statistics displayed on time receiver and time transmitter entities*...continued*

Receive counters	Description
`PortStatTxErrAlloc`	Number of transmit packets allocation errors
Miscellaneous counters	**Description**
`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 time receiver endpoint (e.g. Grandmaster change, Grandmaster reference clock discontinuity...)
`PortStatNumNotAsCapable`	Number of transitions from AS_Capable=TRUE to AS_Capable=FALSE

5.2.5.2 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
```

5.2.5.3 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
```

5.2.5.4 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.

5.2.5.4.1 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 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 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 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
```

5.2.5.4.2 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
```

5.2.5.4.3 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 299998 0
```

5.3 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.

5.3.1 Introduction

The i.MX and Layerscape platforms provided by NXPsupport hardware assist for 1588-compliant time stamping with the 1588 timer module. The software components require to run the IEEE 1588/802.1AS protocol utilizing hardware features that 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 whereas the IEEE 802.1AS mentioned is IEEE 802.1AS-2011.*

5.3.2 IEEE 1588 device types

There are five basic types of PTP devices in IEEE 1588.

1. **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 can serve as a source of time (if used as a master clock) or can synchronize with another clock (if used as a slave clock).
2. **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).
3. **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.
4. **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.
5. **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.*

5.3.3 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.

5.3.4 Software stacks

5.3.4.1 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), 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 must 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.

5.3.4.2 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 BTCA
- Supports Grandmaster, Clock time transmitter, and Clock time receiver 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.

5.3.5 Quick start for IEEE 1588

The following sections describe the clock verification process for ordinary, boundary, and transparent clocks for IEEE 1588.

5.3.5.1 Ordinary clock verification

Connect two network interfaces in a back-to-back manner for two boards. Make sure that 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 starts. Thereby, the slave linuxptp selected automatically synchronizes to the master. It also displays synchronization messages such as time offset, path delay and so on. For example, see the below log:

```
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

By default, the BMC (Best Master Clock) algorithm selects the master clock. 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 the current case, specify one clock as master with the below option. (The other clock uses the default priority1 value 128.)

```
--priority1=127
```

One-step timestamping

Currently one-step timestamping is supported only on DPAA2. To use one-step timestamping, add the below option for ptp4l running.

```
--twoStepFlag=0
```

5.3.5.2 Boundary clock verification

At least three boards are needed for boundary clock verification. Below is an example for a three-boards network connection. Make sure that there is no MAC address conflict on the boards and the IP addresses are set properly. Then, ping the test network.

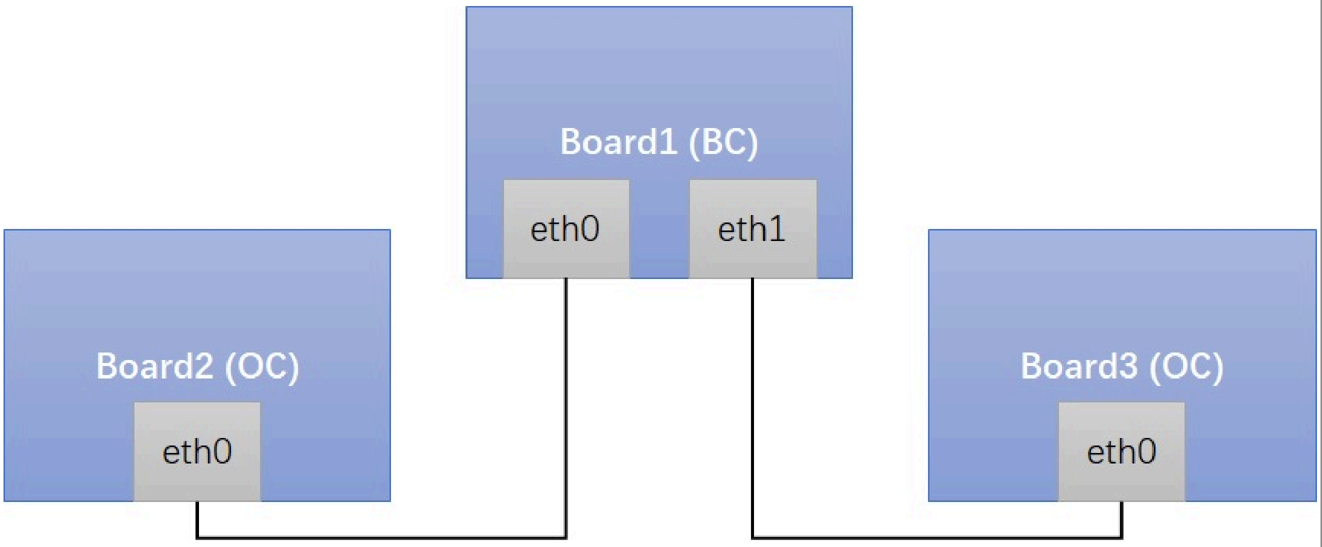


Figure 108. Hardware setup for boundary clock verification

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

By default, the BMC (Best Master Clock) algorithm selects the master clock. 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 using the below option. (The other clock is using the default priority1 value 128.)

```
--priority1=127
```

One-step timestamping

Currently one-step timestamping is supported only on DPAA2. To use one-step timestamping, add the below option for ptp4l running.

```
--twoStepFlag=0
```

5.3.5.3 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.

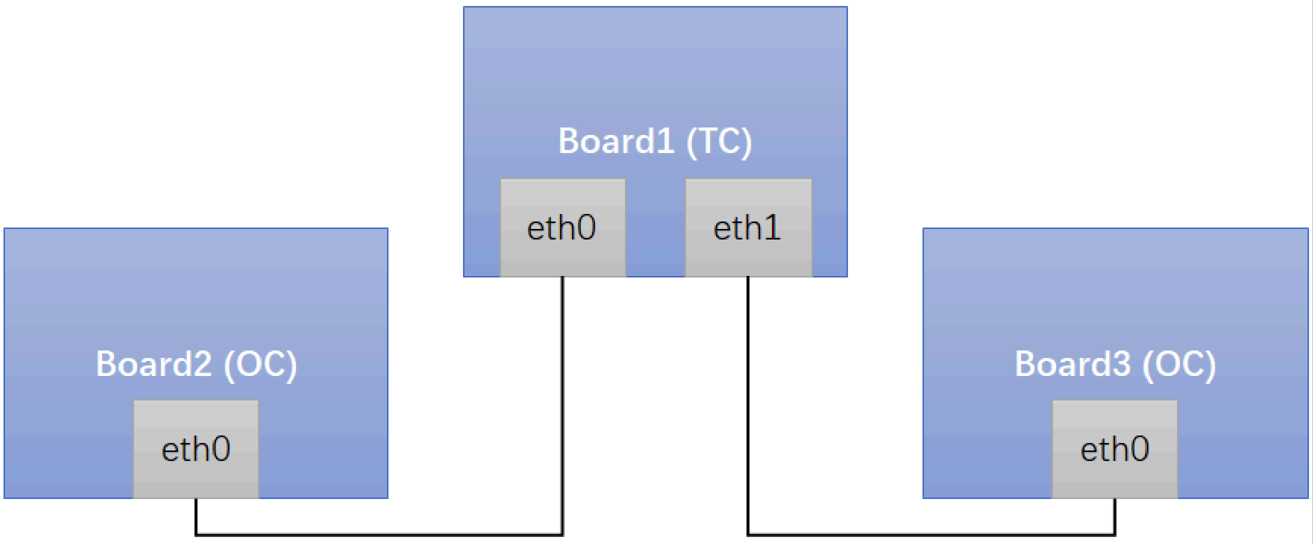


Figure 109. Sample network connection of three boards

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.

5.3.6 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.

5.3.6.1 Time-aware end station verification

Connect two network interfaces in a back-to-back manner for two boards. Make sure that there is no MAC address conflict on the boards. Set the IP address properly and then ping the test network.

Remove the below option in the `/etc/linuxptp/gPTP.cfg` file to use by default the larger value, because the estimate path delay including PHY delay may exceed 800 ns since the 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
```

On running the command above, time synchronization starts and the slave linuxptp selected automatically synchronizes to the master. It also displays synchronization messages such as time offset, path delay, and so on.

5.3.6.2 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. Ensure that there is no MAC address conflict on the boards.

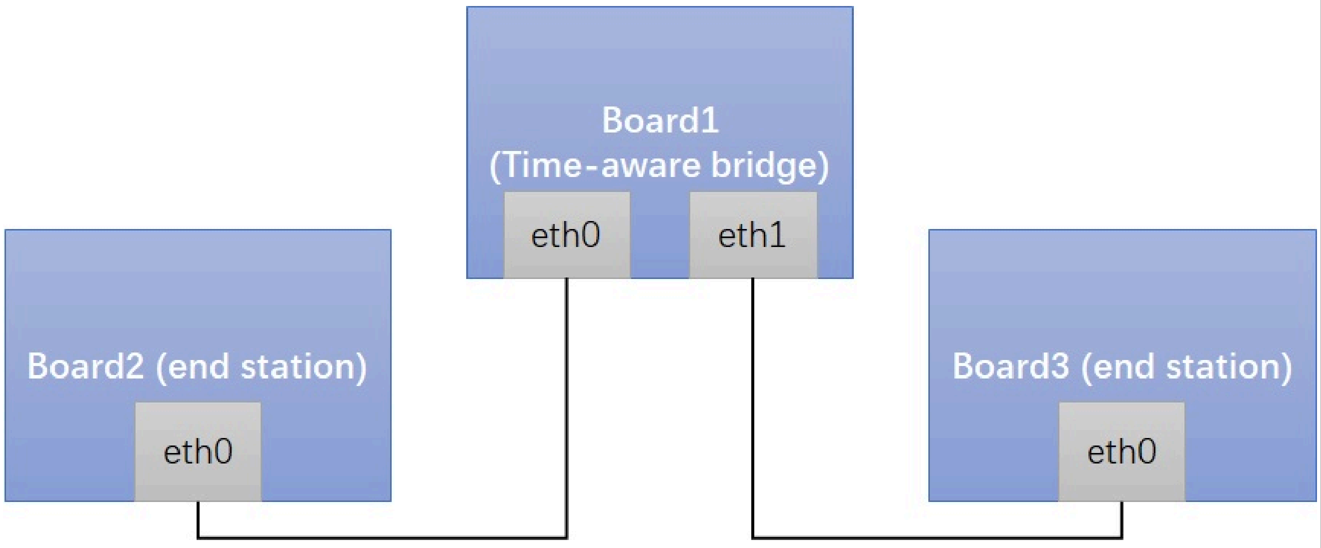


Figure 110. Setup for time-aware bridge verification

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 starts between the three boards, and the linuxptp slaves selected automatically synchronize to the unique master with synchronization messages displayed (such as time offset, path delay and so on).

5.3.7 Virtual clock use case

1. Connect the eth0 of two boards
2. Enable virtual clock on two boards. Ensure that the network interface eth0 uses ptp0. Create two virtual clocks for eth0:

   ```
   $ echo 2 > /sys/class/ptp/ptp0/n_vclocks
   ```

3. Run linuxptp on the two boards:

   ```
   $ ptp4l -i eth0 --phc_index 2 -f /etc/ptp4l_cfg/gPTP.cfg --domainNumber 2 -m
    &
   $ ptp4l -i eth0 --phc_index 3 -f /etc/ptp4l_cfg/gPTP.cfg --domainNumber 3 -m
    &
   ```

5.3.8 Long term test

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

5.3.8.1 Virtual clock use case

To run the virtual clock use case, follow the steps below:

1. Connect the eth0 of two boards.
2. Enable virtual clock on two boards
3. Ensure that the network interface eth0 uses ptp0. Then, create two virtual clocks for eth0 as shown below:
   ```
   $ echo 2 > /sys/class/ptp/ptp0/n_vclocks
   ```
4. Run linuxptp on the two boards:
   ```
   $ ptp4l -i eth0 --phc_index 2 -f /etc/ptp4l_cfg/gPTP.cfg --domainNumber 2 -m
    &
   $ ptp4l -i eth0 --phc_index 3 -f /etc/ptp4l_cfg/gPTP.cfg --domainNumber 3 -m
    &
   ```

5.3.9 Known issues and limitations

1. When the LS1028A TSN switch in Linux is configured as an L2 switch, the interfaces should not be configured with IP addresses. Running linuxptp on these interfaces must use the 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, such as ethtool queries and PTP operations. So, the workaround is to do operations on the eth1 and PTP of dwmac only after running "`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` while 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
   ```

5.4 Networking

5.4.1 Q-in-Q on LS1028A Felix switch

1. **Q-in-Q feature**
   Q-in-Q feature allows service providers to create a Layer 2 Ethernet connection between two user sites. Providers can segregate VLAN traffic of different users on a link or bundle using different user VLANs. When using Q-in-Q, the service VLAN tag `(S-TAG: 0x88A8)` prepend the `802.1Q VLAN` tags `(C-TAG:0x8100)` of the user.
2. **Q-in-Q application scenario**
   In the following scenario, port **swp0** of the switch connects with Customer 1's LAN, **swp1** connects with the MAN of the ISP.
   The traffic with VLAN tag is shown below:

uplink: Customer LAN `(only C-TAG) -> swp0 -> swp1 (add S-TAG) -> ISP MAN (S-TAG + C-TAG)`
downlink: ISP MAN `(S-TAG + C-TAG) -> swp1 (pop S-TAG)-> swp0 (only C-TAG) -> Customer LAN`

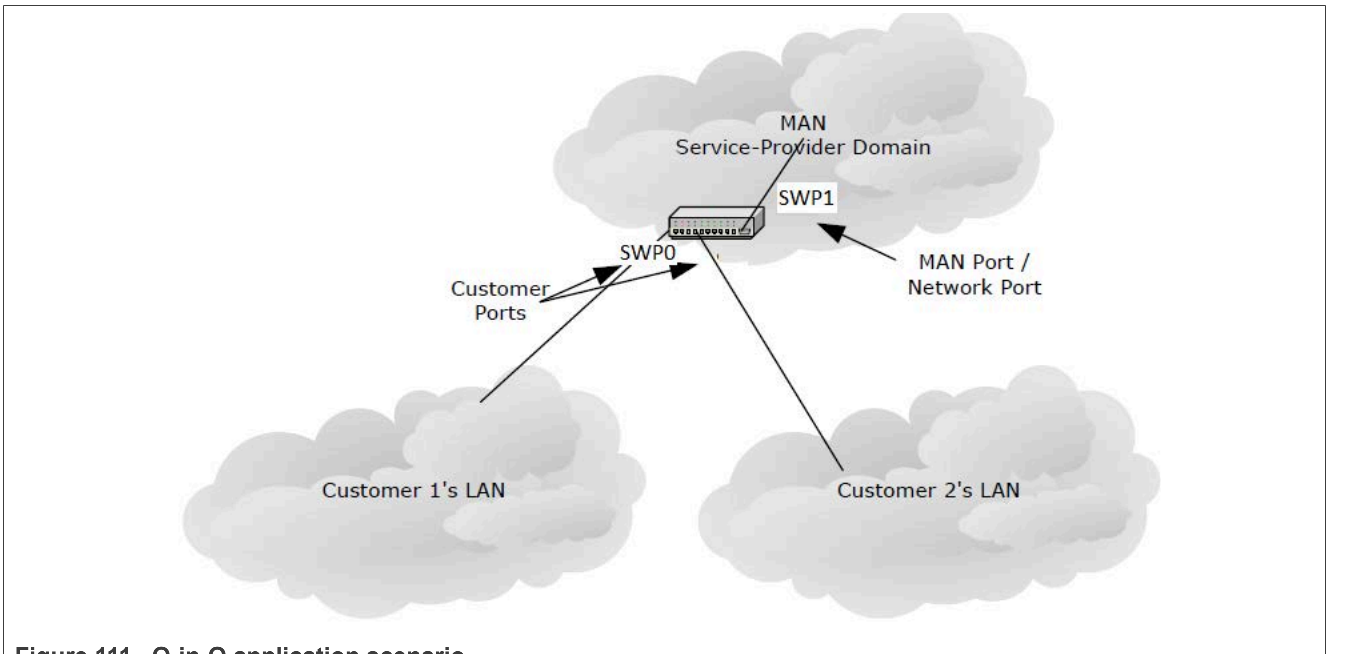


Figure 111. Q-in-Q application scenario

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 the 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. The result is displayed below:

```
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)
```

5.4.2 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. The 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. Table 93 shows the chain allocation:

Table 93. 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.
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,

Table 93. Chain allocation*...continued*

chain ID	Actions	Hardware module	keys
			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 must 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:

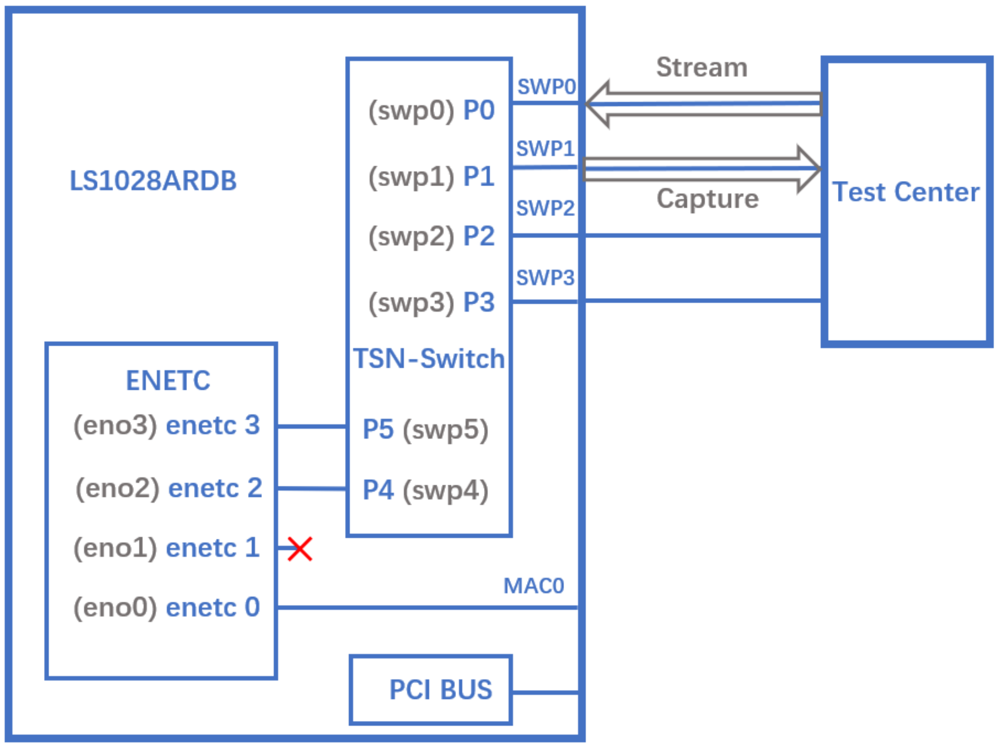


Figure 112. VCAP test

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)`

6 Protocols

This section describes the protocols supported.

6.1 EtherCAT MainDevice

Real-time Edge supports the usage of EtherCAT (Ethernet for Control Automation Technology) MainDevice and integrates the IGH EtherCAT MainDevice stack and SOEM. EtherCAT is verified on NXP platforms.

6.1.1 Introduction

EtherCAT is an Ethernet-based fieldbus system, invented by BECKHOFF Automation. The protocol is standardized in IEC 61158 and is suitable for both hard and soft real-time computing requirements in automation technology. EtherCAT was developed with an objective to apply Ethernet for automation applications requiring short data update times, low communication jitter, and reduced hardware costs.

Note:

- *Data update time is also called cycle time.*
- *Low communication jitter implies less than 100 μs and for precise synchronization purposes, it is less than 1 μs.*

- EtherCAT is Fast: 1000 digital I/O: 30 μs, 100 SubDevice: 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, and therefore inexpensive.
- EtherCAT is Easy: everybody knows Ethernet and it is simple to use.

Currently, there are two open source EtherCAT MainDevices that Real-time Edge software supports. These are IGH for Cortex-A core and SOEM for Cortex-M core. Both IgH and SOEM are solutions that help users get rid of the low-level development directly on the EtherCAT protocol, and provide a common API for real-time applications. For more information, see https://rt-labs.com/ethercat/ and https://www.etherlab.org.

IgH shows superior real-time characteristics since it is integrated into the Preempt-RT Linux kernel and uses native Ethernet drivers (only supported on a few specific NXP platforms). Also, IgH is more powerful and integrated by providing users with auxiliary functions such that it encourages users to focus on specific high-level programs. For example, IgH runs with an automatic MainDevice FSM (Finite State Machine). FSM is a EtherCAT SubDevice manager dynamically adapting the SubDevice configurations to the new topology and responding to requests from the application-layer. In addition, IgH provides a command-line tool in user space to display detailed information about MainDevice/SubDevice configurations and to list SDO dictionaries and reading/writing addresses.

SOEM (Simple Open EtherCAT MainDevice) is a C library that offers methods to send and receive EtherCAT frames. It is very light-weight but still powerful and stable. It can run on multiple operating systems, for example, Linux, Windows, or FreeRTOS. It can even run without an operating system (such as 'BareMetal'). Unlike IgH, users must 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 the Cortex-M core on i.MX 8M Mini LPDDR4 EVK and i.MX 8M Plus LPDDR4 EVK platforms and runs on FreeRTOS or BareMetal.

6.1.2 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 SubDevice is possible, but must be initiated by the MainDevice.
- If IP routing is required, the EtherCAT protocol can be inserted into UDP/IP datagrams. This also enables any control with Ethernet protocol stack to address EtherCAT systems.
- It does not support shortened frames.

The Figure 113 shows the EtherCAT frame structure.

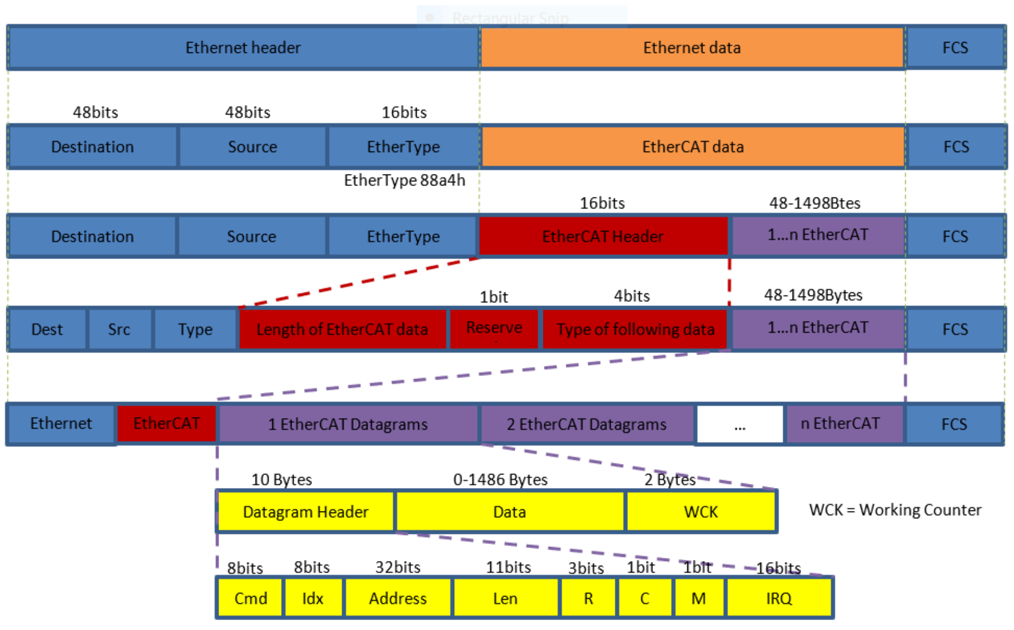


Figure 113. EtherCAT frame structure

6.1.3 IGH EtherCAT architecture

The components of the MainDevice environment are described below:

- **MainDevice module:** This is the kernel module containing one or more EtherCAT MainDevice 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 MainDevice via the device interface. These modified network drivers can handle network devices used for EtherCAT operation and 'normal' Ethernet devices in parallel. A MainDevice can accept a certain device and then, is able to send and receive EtherCAT frames. Ethernet devices declined by the MainDevice module are connected to the kernel's network stack, as usual.
- **Application:** A program that uses the EtherCAT MainDevice (usually for cyclic exchange of process data with EtherCAT SubDevice). These programs are not part of the EtherCAT MainDevice code, but require to be generated or written by the user. An application can request a MainDevice through the application interface. If this succeeds, it has the control over the MainDevice: It can provide a bus configuration and exchange process data. Applications can be kernel modules that use the kernel application interface directly. They also include user space programs, which use the application interface via the EtherCAT library or the RTDM library. The Figure 114 shows the IGH EtherCAT MainDevice architecture.

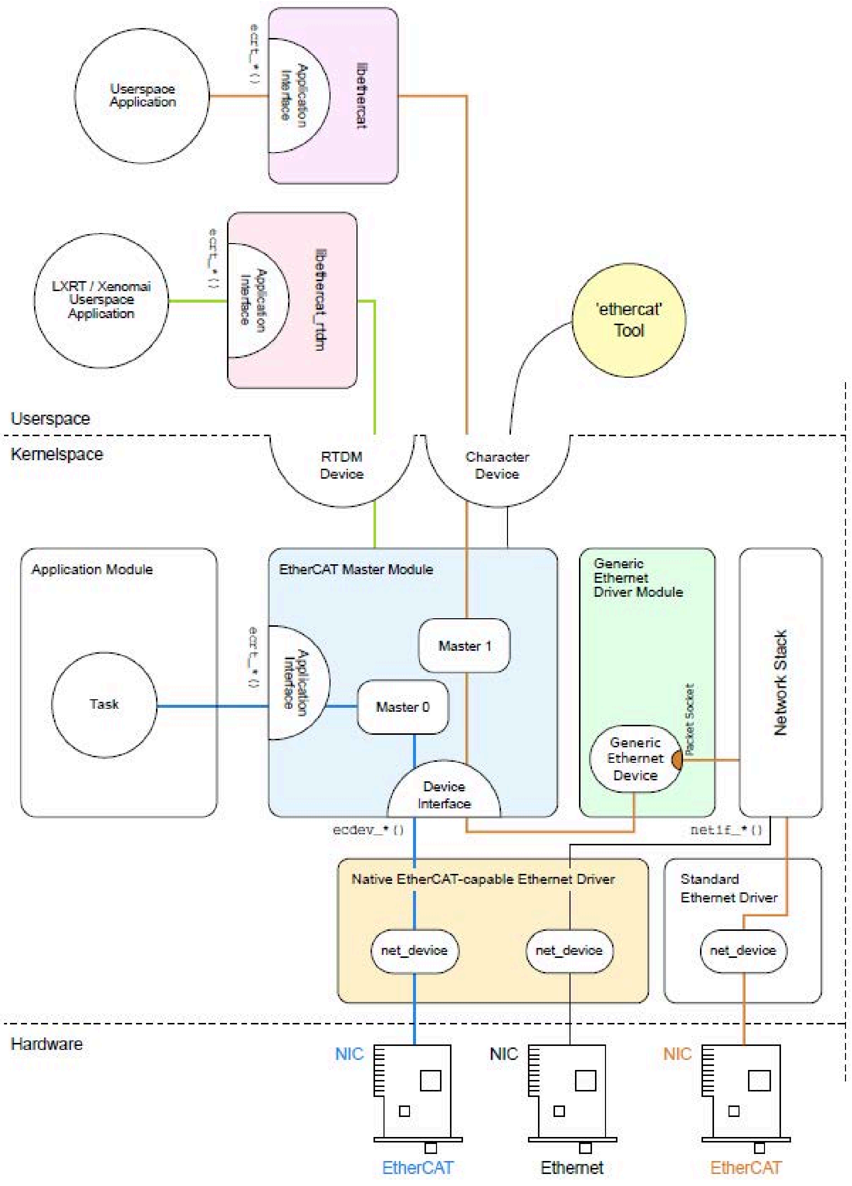


Figure 114. IGH EtherCAT MainDevice architecture

6.1.3.1 IGH EtherCAT Device Drivers

The EtherCAT protocol is based on the Ethernet standard, so a MainDevice 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 MainDevice direct and exclusive access to the Ethernet hardware. This implies that the network device must not be connected to the kernel's stack as usual, which allows a high Real-time performance. In Real-time Edge software, there are three native Ethernet drivers:

- `ec_fec`: the native driver `ec_fec` can be used for the FEC MAC on i.MX 8DXL LPDDR4 EVK, i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, and i.MX 93 EVK. This driver is not verified on other i.MX platforms in this release. Please note that the original Ethernet fec driver must be recompiled to a module by reconfiguring Linux with command "`make menuconfig`" when using the `ec_fec` native driver.

- `ec_enetc`: the native driver "`ec_enetc`" is used for ENETC MAC on the LS1028ARDB platform.
- `ec_enetc4`: the native driver "`ec_enetc4`" is used for ENETC4 MAC on the i.MX95 and i.MX943 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. Therefore, it can be used by all the hardware platforms that Real-time Edge supports. However, the performance by using a generic Ethernet Device driver is a bit worse than the native driver, because the Ethernet frame data has to traverse the Linux network stack.

6.1.3.2 IGH EtherCAT setup

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

6.1.3.2.1 Specifying the Ethernet device

The Ethernet device is specified by setting `MASTER0_DEVICE` variable to the MAC address of the Ethernet device to use as the EtherCAT network interface as below:

```
MASTER0_DEVICE="00:04:9f:07:11:a6"
```

For LS1046ARDB or LS1043ARDB platforms, if multiple MainDevice are required, adding a non-empty variable `MASTER1_DEVICE` creates a second master, and so on.

6.1.3.2.2 Generic Ethernet driver

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

```
DEVICE_MODULES="generic"
```

6.1.3.2.3 Native Ethernet driver for i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK and i.MX 93 EVK

The native Ethernet driver "`ec_fec`" is enabled on Real-time Edge images by default for i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK and i.MX 93 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.

6.1.3.2.4 Native Ethernet driver for LS1028ARDB

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

```
DEVICE_MODULES="enetc"
```

6.1.3.2.5 Native Ethernet driver for i.MX95 and i.MX943

The native Ethernet driver `ec_enetc4` is enabled on Real-time Edge images by default.

It can be specified by setting `DEVICE_MODULES` variable to `enetc4` as shown in the below `/etc/ethercat.conf` file.

```
DEVICE_MODULES="enetc4"
```

6.1.3.2.6 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 MainDevice and redundancy features can be supported on LS1046ARDB and LS1043ARDB platforms.

DPAA is a network co-processer that is more complex than ordinary network cards and is different from other native drivers mentioned above, . So the native driver "`ec_dpaa1`" must be configured by "`ethercat_port`" variable in U-Boot to notify the DPAA co-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 MainDevice with index 'x', while the `master<x>_backup` variable specifies the backup Ethernet port if redundancy is required. Every MainDevice must be bundled a specified CPU core. and core_index variable is used to specific which CPU core to be bundled for this MainDevice.

The relationship between Ethernet port name on LS1046ARDB chassis and the name in the Linux is in the Table 94:

Table 94. 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

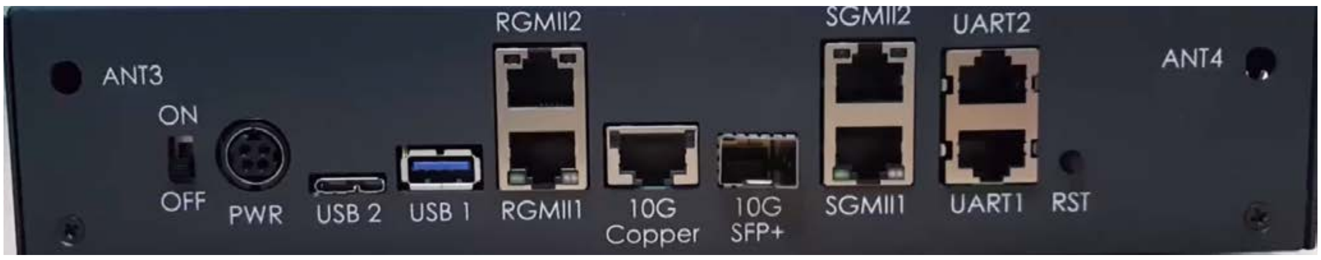


Figure 115. LS1046ARDB Ethernet chassis

The relationship between Ethernet port name on LS1043ARDB chassis and the name in the Linux is shown in Table 95.

Table 95. 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

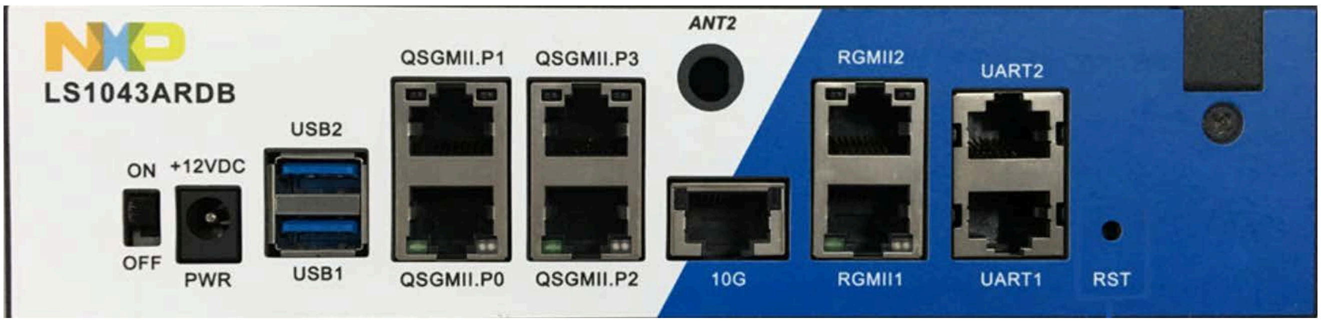


Figure 116. LS1043ARDB Ethernet chassis

There are 3 instances of ec_dpaa1 configuration for LS1046ARDB.

1. There is only one network topology, and no backup port for redundancy. The Ethernet port of this network is RGMII1, and the MainDevice 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 MainDevice 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 MainDevice 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 MainDevice of

this network is bundled to CPU core 3. The main port of the other network is SRGMII1, and the MainDevice 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 MainDevice 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 MainDevice.

6.1.3.2.7 Starting IGH EtherCAT

Check the mac address under uboot(i.MX6ULL/i.MX 8M Mini/i.MX 8M Plus/i.MX93):

```
u-boot=> print ethaddr
```

Bind dtb file under uboot:

```
# for i.MX 6ULL EVK
u-boot=> setenv fdt_file imx6ull-14x14-evk-igh.dtb

# for i.MX 8M Mini LPDDR4 EVK
u-boot=> setenv fdtfile imx8mm-evk-igh.dtb

# for i.MX 8M Plus LPDDR4 EVK
u-boot=> setenv fdtfile imx8mp-evk-igh.dtb

# for i.MX 93 EVK
u-boot=> setenv fdtfile imx93-11x11-evk-igh.dtb

u-boot=> run bootcmd
```

After filling in the MAC address found in U-Boot into "`MASTER0_DEVICE`" and "`DEVICE_MODULES="fec"`" in file “`/etc/ethercat.conf`”, use the below command to start the IGH EtherCAT daemon:

```
$ ethercatctl start
```

For i.MX95/i.MX943

```
# Input mac address to“MASTER0_DEVICE”and “DEVICE_MODULES="enetc4"” in“/etc/
ethercat.conf”

# for i.mx95-15x15-lpddr4x-evk, connect servo to ENET1 port (mac addr: eth0)
ENET1:
        echo -n 0001:00:00.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind

# for i.mx95-19x19-lpddr5-evk, connect servo to ENET1 port (mac addr: eth0)
ENET1:
        echo -n 0002:00:00.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind

# for imx943-19x19-lpddr4-evk, connect servo to ENETC1 port (mac addr: eth1)
ENETC1:
        echo -n 0001:01:08.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind

# for imx943-19x19-lpddr5-evk, connect servo to ENETC1 port (mac addr: eth1)
ENETC1:
        echo -n 0001:01:08.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
```

After filling in the MAC address found in U-Boot into "`MASTER0_DEVICE`" and "`DEVICE_MODULES="enetc4"`" in file “`/etc/ethercat.conf`”, use the below command to start the IGH EtherCAT daemon:

```
$ ethercatctl restart
```

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 MainDevice and all SubDevice information and status. The usage is as below:

```
Usage: ethercat <COMMAND> [OPTIONS] [ARGUMENTS]
    Commands (can be abbreviated):
      alias      Write alias addresses.
      config     Show slave configurations.
      crc        CRC error register diagnosis.
      cstruct    Generate slave PDO information in C language.
      data       Output binary domain process data.
      debug      Set the master's debug level.
      domains    Show configured domains.
      download   Write an SDO entry to a slave.
      eoe        Display Ethernet over EtherCAT statistics.
      foe_read   Read a file from a slave via FoE.
      foe_write  Store a file on a slave via FoE.
      graph      Output the bus topology as a graph.
      master     Show master and Ethernet device information.
      pdos       List Sync managers, PDO assignment and mapping.
      reg_read   Output a slave's register contents.
      reg_write  Write data to a slave's registers.
      rescan     Rescan the bus.
      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
```

6.1.3.3 IGH EtherCAT User Space setup

IGH EtherCAT User Space supports i.MX 8M Mini LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK, i.MX95 evk and i.MX943 EVK boards.

6.1.3.3.1 IGH EtherCAT User Space Introduction

IGH EtherCAT stack has both code in the kernel space and user space. In kernel space, it integrates multiple drivers for different networking devices. In the user space, it supplies a library (`libetherat.so/.a`) for user's application. Users can set/get the running parameters of the device using API of this library. The API reads or writes data via ioctl mechanism of Linux kernel.

Ioctl invokes the routine implemented in MainDevice module (`ec_master.ko`). Then MainDevice module then calls the RX/TX function in different drivers. Ioctl is a system call supplied by Linux. The updates for running parameters triggers system call of the kernel, though this introduces more system jitter.

On the other hand, there are mainly two important threads for using IGH EtherCAT stack. One is MainDevice operation thread running in kernel space, which is responsible for parsing the frames received from EtherCAT devices. The other thread is RT thread implemented by user, which triggers RX/TX functionality cyclically according to cycle time. Due to these two threads are running in different spaces, a context switch between user space and kernel space is required, which also increases system jitter and time latency.

The user space IGH EtherCAT stack runs wholly in user space. Therefore, it does not need ioctl. User can invoke the functions of MainDevice module directly, this can avoid system call, it can decrease jitter of system obviously.

From the perspective of threads, in user space stack, MainDevice operation thread is also running in user space, so we can refactor OP thread to one callback function, this callback function is called in the user's cyclic RT thread. In such a case, there is only one main thread in whole system, which avoids context switching. This also improves the performance, such as jitter and latency and other such parameters.

User space IGH EtherCAT stack has no code that runs in the kernel space, which makes it entirely independent of the kernel version.

6.1.3.3.2 Configure EtherCAT User Space dtb

The EtherCAT User Space dtb can be reconfigured by adding the below lines in the "`source/meta-real-time-edge/conf/distro/include/real-time-edge-base.inc`" file.

```
# dtb for EtherCAT userspace application
KERNEL_DEVICETREE:append:imx8mm-lpddr4-evk = " freescale/imx8mm-evk-ecat-
userspace.dtb"
KERNEL_DEVICETREE:append:imx8mp-lpddr4-evk = " freescale/imx8mp-evk-ecat-
userspace.dtb"
KERNEL_DEVICETREE:append:imx93evk = " freescale/imx93-11x11-evk-ecat-
userspace.dtb"
```

6.1.3.3.3 Starting IGH EtherCAT User Space

Use the steps below to start the IGH EtherCAT User Space.

1. To update the `dtb` file and boot into the kernel, run the following command:

```
# on i.MX 8M Mini
uboot => setenv fdtfile imx8mm-evk-ecat-userspace.dtb
# on i.MX 8M Plus
uboot => setenv fdtfile imx8mp-evk-ecat-userspace.dtb
# on i.MX 93
```

```
uboot => setenv fdtfile imx93-11x11-evk-ecat-userspace.dtb
u-boot=> run bootcmd
```

2. For i.MX 8MP, i.MX 8MM, and i.MX 93 platforms, you must limit the DDR memory in 4G bytes via the '`mem`' variable in U-Boot bootargs.

```
# Example
uboot => mem=4096M
```

3. The EtherCAT tool (named `ethercat_userspace`) invokes the `ioctl` command to scan and display EtherCAT devices. For the convenience of using the EtherCAT tool, one virtual ioctl interface is implemented to support this tool. One backend program (named `ethercat_master`) supplies the service for the tool. Firstly, run the backend program:

```
# mkdir -p /dev/hugepages
# mount -t hugetlbfs hugetlbfs /dev/hugepages
# echo 448 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
# ethercat_master &
```

   Then, run EtherCAT tool:

```
# ethercat_userspace slaves
```

4. Now, exit `ethercat_master` program before running the `motor control` example:

```
# fg
# Type 'ctrl + c' to exit IGH
   // where "IGH" --> "ethercat_master program"
# cd /examples/igh-ethercat-userspace/
```

5. Running user space EtherCAT stack on i.MX95 and i.MX943 platform

```
# i.MX95
# eth0:
   insmod ./kpage_ncache.ko
    echo -n 0002:00:00.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
    echo -n enetc_pci_uio > /sys/bus/pci/devices/0002:00:00.0/driver_override
    echo -n 0002:00:00.0 > /sys/bus/pci/drivers/enetc_pci_uio/bind
    export ECAT_MAIN_DEV="enetc4@0002:00:00.0"
# eth1:
    insmod ./kpage_ncache.ko
    echo -n 0002:00:10.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
    echo -n enetc_pci_uio > /sys/bus/pci/devices/0002:00:10.0/driver_override
    echo -n 0002:00:10.0 > /sys/bus/pci/drivers/enetc_pci_uio/bind
    export ECAT_MAIN_DEV="enetc4@0002:00:10.0"
```

```
# i.MX943
# eth1:
   insmod ./kpage_ncache.ko
    echo -n 0001:01:08.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
    echo -n enetc_pci_uio > /sys/bus/pci/devices/0001:01:08.0/driver_override
    echo -n 0001:01:08.0 > /sys/bus/pci/drivers/enetc_pci_uio/bind
    export ECAT_MAIN_DEV="enetc4@0001:01:08.0"
# eth2:
    insmod ./kpage_ncache.ko
    echo -n 0001:01:10.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
    echo -n enetc_pci_uio > /sys/bus/pci/devices/0001:01:10.0/driver_override
    echo -n 0001:01:10.0 > /sys/bus/pci/drivers/enetc_pci_uio/bind
    export ECAT_MAIN_DEV="enetc4@0001:01:10.0"
```

Note: *Please download dpdk-extras repo to compile kpage_ncache.ko .*

```
# Refer to README in linux/kpage_ncache directory.
$ git clone https://github.com/nxp-qoriq/dpdk-extras
```

Note: *To run the user application, exit and close `ethercat_master` program.*

- *For example, see the below `ec_motor_example` to control motor rotation:*

```
./ec_motor_example 131072 1
```

In the above command, `131072` is the encoding rate of the motor, and `1` is the number of motors.

Note: *For better performance, you can isolate the working core from Linux by using `isolcpus=1` for U-Boot bootargs. By default, user space IgH EtherCAT works on Core1.*

Note: *For details about how to use the user space IgH EtherCAT library, refer to the readme file available on the below compile path:*

```
{$build_dir}/build-imx8mpevk-real-time-edge/tmp/work/imx8mp_lpddr4_evk-poky-
linux/igh-ethercat-userspace/1.6.4/git/Readme_For_User_Space_IGH_EtherCAT.txt
```

6.1.3.4 The 'real-time-edge-servo' stack

The ***real-time-edge-servo*** is a CiA402 (also referred to as DS402) profile framework based on IgH CoE interface. (For details of an EtherCAT MainDevice stack, see Section 6.1). 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, SubDevice, and MainDevice configurations, and definitions of all the axles.

The stack has been tested on below CoE servo production: DELTA ASDA-B3, HCFA SV-X6EB, SV-X3EB, Just Motion Control 2HSS458-EC, and INOVANCE InoSV680N.

6.1.3.4.1 CoE network

The Figure 117 illustrates a typical CoE network.

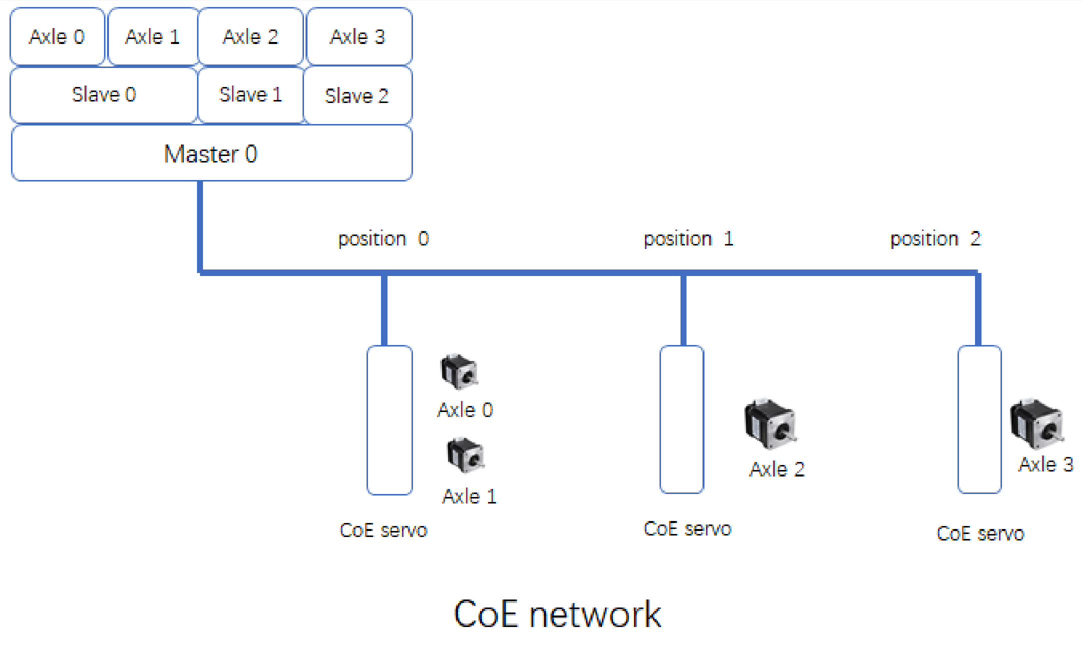


Figure 117. CoE network

There are three CoE servos on this network and are named SubDevice ***x*** according to their position. Each CoE servo can have more than one axle. The libnservo initiates the CoE network and encapsulates the details of network topology into axle nodes. Therefore, the developer can focus on each axle operation without bothering about the network topology.

6.1.3.4.2 Libnservo architecture

real-time-edge-servo runs on top of the *Igh* EtherCAT stack. The *Igh* stack provides CoE communication mechanisms - Mailbox and Process Data. Using these mechanisms, real-time-edge-servo can access the CiA Object Dictionary located on CoE servo. The Figure 118 shows the Libservo architecture.

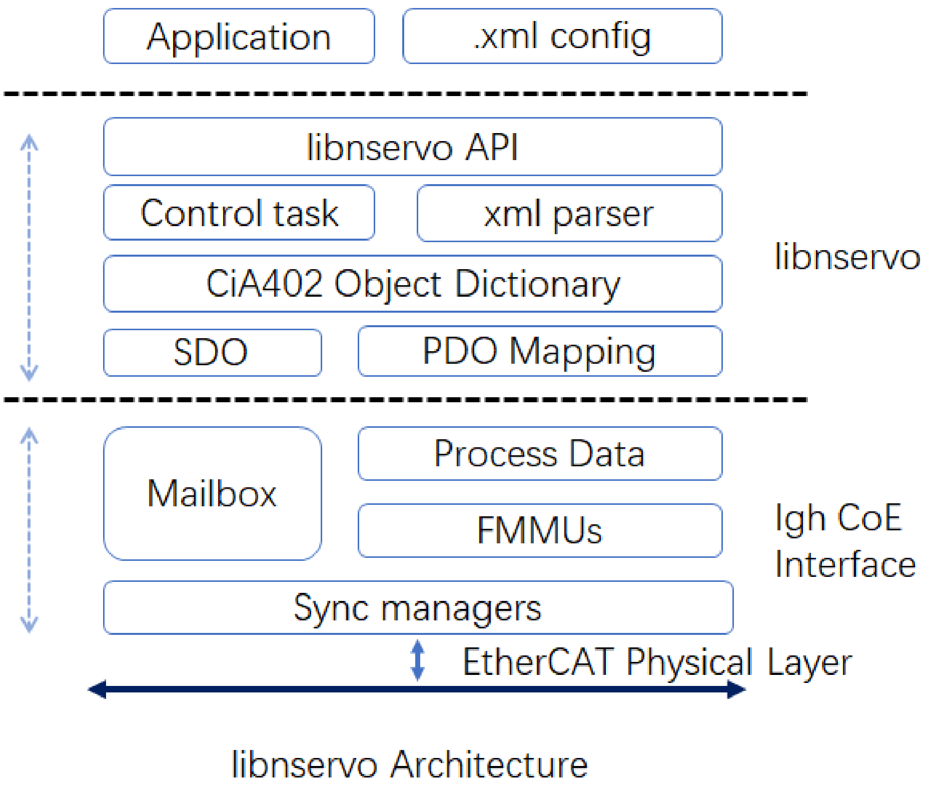


Figure 118. Libnservo architecture

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

6.1.3.4.3 Xml configuration for real-time-edge-servo

This section describes how the xml config file that describes a CoE network must be configured.

The basic framework of the XML configuration is shown in the 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 MainDevice status update. the value `#1` means update the MainDevice status every task period.
- `<slave_status_update_freq>` element indicates the frequency of SubDevice status update. the value `#1` signifies to update the SubDevice 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 MainDevice element. For most cases, there is only one MainDevice on a host.
- `<Axles>` element can contain more than one Axle element, which is an important feature for the developers.

6.1.3.4.3.1 MainDevice element

As shown in the Section 6.1.3.4.1, the MainDevice can have many SubDevice, so the MainDevice element might consist of few *SubDevice* 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 indicates the index of the MainDevice. As mentioned above, for many cases, there is only one MainDevice, so the value of this element is always #0.
- `<Reference_clock>` element indicates the SubDevice that is used the reference clock.
- `<Slave>` element indicates that there is a SubDevice on this MainDevice.

SubDevice 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 SubDevice.
- *slave_position* attribute means which position of the SubDevice is on this network.
- <Name>element is the name of the SubDevice.
- <Emerg_size> element is always 8 for all CoE device.
- <WatchDog> element is used to set the watch dog of this SubDevice.
- <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 that is required to be mapped to.
  - <Index> PDO address.
  - <Name> PDO name
  - <Entry> the object dictionary we want to mapped.

The Entry element is used to describe an object dictionary that is required to be mapped to.

```
<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.

6.1.3.4.3.2 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* belongs to.
- `slave_position` attribute indicates which *slave* this *axle* belongs to.
- `AxleOffset` attribute indicates which *axle* this *axle* is on the SubDevice. As mentioned above, a CoE SubDevice could have more than one *axle* . If this axle is the second axle on the SubDevice, set *AxleOffset="#1"* .
- `<Mode>` indicates the mode this axle works 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>
```

6.1.3.4.4 Testing a CoE servo system

6.1.3.4.4.1 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, the i.MX 8M Mini LPDDR4 EVK hardware platform is used.

Figure 119. '2HSS458-EC' servo system

Figure 120. ASDA-B3-E Servo System

6.1.3.4.4.2 Software preparation

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

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

6.1.3.4.4.3 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
```

6.1.3.4.4.4 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 /root/nservo_example/hss248_ec_config_pp.xml &
```

- Check whether the status of the SubDevice has been transferred from "PREOP" to "OP".

```
[root]# ethercat slaves
0  0:0  OP  +  2HSS458-EC
```

- Check whether the phase of the MainDevice 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 /root/nservo_example/hss248_ec_config_pv.xml &
```

- Check whether the status of the SubDevice has been transferred from "PREOP" to "OP".

```
[root]# ethercat slaves
0  0:0  OP  +  2HSS458-EC
```

- Check whether the phase of the MainDevice 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
```

6.1.3.4.4.5 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 /root/nservo_example/Delta-ASDA-B3-pp.xml &
   ```

   Check whether the status of the SubDevice has been transferred from "PREOP" to "OP".

   ```
    [root]# ethercat slaves
    0 0:0 OP + Delta ASDA-B3-E EtherCAT(CoE) Drive Rev0.00
   ```

2. Check whether the phase of the MainDevice has been transferred from "Idle" to "Operation".

   ```
   [root]# ethercat master | grep Phase Phase: Operation
   ```

3. Run the below commands to test whether the motor works.
   a. Get the current mode of axle 0.

      ```
      [root]# nservo_client -a 0 -c get_mode
      get_mode of the axle 0 : Profile Position Mode
      ```

   b. Get the current position of axle 0.

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

   c. Get the profile speed of axle 0.

      ```
      [root# nservo_client -a 0 -c get_profile_velocity
      get_profile_velocity of the axle 0 : 0
      ```

   d. Set profile speed of axle 0.

      ```
      [root]# nservo_client -a 0 -c set_profile_velocity:16777216
      set_profile_velocity of the axle 0 : 16777216
      ```

   e. Set profile speed to 1 revolutions per second. Set target position of axle 0.

      ```
      [root]# nservo_client -c set_target_position:167772160
      set_target_position of the axle 0 : 167772160
      ```

      The value 167772160 means that the motor turns 10 rounds. (target_position: 167772160 - current_position:0) / 16777216 = 10

   f. Get the current position of axle 0.

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

4. Exit.

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

Profile Velocity mode test

1. Start the test service as below.
   ```
   [root]# nservo_run -f /root/nservo_example/Delta-ASDA-B3-pv.xml &
   ```
2. Check whether the status of the SubDevice 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 MainDevice has been transferred from "Idle" to "Operation".
   ```
   [root]# ethercat master | grep Phase Phase: Operation
   ```
4. Run the below commands to test whether the motor works.
   a. Get current mode of axle 0.
      ```
      [root]# nservo_client -a 0 -c get_mode
           get_mode of the axle 0 : Profile Velocity Mode
      ```
   b. Set target speed of axle 0.
      ```
      [root]# nservo_client -a 0 -c set_target_velocity:600
          set_target_velocity of the axle 0 : 600
      ```
      The value 600 means that the motor turns with 60 revolutions per minute for ASDA-B3-E servo.
   c. Get current speed of axle 0.
      ```
      [root]# nservo_client -a 0 -c get_current_velocity
          get_current_velocity of the axle 0 : 600
      ```
   d. Get target speed of axle 0.
      ```
      [root]# nservo_client -a 0 -c get_target_velocity
         get_target_velocity of the axle 0 : 600
      ```
5. Exit
   ```
    [root]# nservo_client -c exit
   ```

Cyclic Sync Position mode test

1. Start the test service as below.
   ```
   [root]# nservo_run -f /root/nservo_example/Delta-ASDA-B3-csp.xml &
   ```
2. Check whether the status of the SubDevice 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 MainDevice 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 the command line for axle 0.

```
./nservo_client -a 0 -c set_tparrays:"Cyclic=1; Scale=46603; Bias=0;
 Accel=8; Decel=8; Max_speed=3600; TpArrays=[(0:1000),(45:1000),(45:1000),
(90:1000)];"
```

The command parameters are described below:

- *Cyclic: if this field is set, the motor will come back to the first point of TpArrays and begin running again*
- *Scale*: this field is used to set the resolution of per unit. For example, the unit of position point in TpArrays is degree. So the Scale must be set to 167772160/360 = *46603.*
  167772160 is the resolution per motor revolution.
- *Bias: this field is used to set the bias value for the position point in TpArrays.*
- *Accel: the acceleration, unit^2 per second.*
- *Decel: the deceleration, unit^2 per second.*
- *Max_speed: The maximum speed, unit per second.*
- TpArrays: is used to save *trajectory planning* information. Each element represents a position point and the time taken for the motor to rotate to this point from the last point. The unit of the time field is ms. Load trajectory planning information from a `tp` file for axle 0. Except loading the trajectory planning information from command line. This information could also be loaded from a `tp` file. The format of a sample `tp` file is shown below.

```

cat example/x6e_sv680_delta_tp_arrays
Axis=0; Cyclic=1; Scale=364; Bias=0; Accel=8; Decel=8; Max_speed=3600;
 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, the motor will come back to the first point of TpArrays and run again.
- *Scale*: this field is used to set the resolution of per unit. For example, the unit of position point in TpArrays is degree. So the Scale must be set to 167772160/360 = *46603.*
  167772160 is the resolution per motor revolution.
- *Bias: this field is used to set bias value for the position point in TpArrays.*
- *Accel: the acceleration in unit^2 per second.*
- *Decel: the deceleration in unit^2 per second.*
- *Max_speed: The maximum speed, unit per second.*
- *TpArrays: are used to save trajectory planning* information. Each element represents a position point and the time taken for the motor rotating to the this point from the last point. The unit of the time field is ms.

```
./nservo_client  -c load_tp_file:"/root/nservo_example/Delta-ASDA-B3-
tp_arrays"
```

- *Cyclic: if this field is set, the motor will come back to the first point of TpArrays and begin running again.*
- *Scale*: this field is used to set the resolution of per unit. For example, the unit of the position point in TpArrays is degree. So the Scale must be set to 167772160/360 = *46603.* 167772160 is the resolution per motor revolution. /
- *Accel: the acceleration, unit^2 per second.*

- *Decel: the deceleration, unit^2 per second.*
- *Max_speed: The maximum speed, unit per second.*
- *TpArrays: used to save* trajectory planning information. Each element represents a position point and the time taken for the motor to rotate to this point from the last point. The unit of the time field is ms. Set the axle 0 to start running using the below command:

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

- Get the *current_position* of axle 0 using the below command:

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

- Set axle 0 to stop running using the below command:

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

- Set all axles in CSP mode to start running using the below command:

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

  All CSP axis in ready status start to run.
- Set all axles in CSP mode to stop running using the below command:

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

  All CSP axis in running status begin to stop.
- Exit

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

6.1.3.4.5 EtherCAT multiple axes control system

6.1.3.4.5.1 HCFA 60-axes servo system

This section demonstrates a 60-axes servo system built using HCFA. This system consists of 60 X3E servo motors and also 60 pointers in the screen that can rotate 360 degrees under the corresponding servo motor control. As shown in Figure 121, this system can render any character on the screen by rotating these pointers.

The software is based on real-time-edge-servo stack. Any platform supported on Real-time Edge can be used as the controller of this servo system.

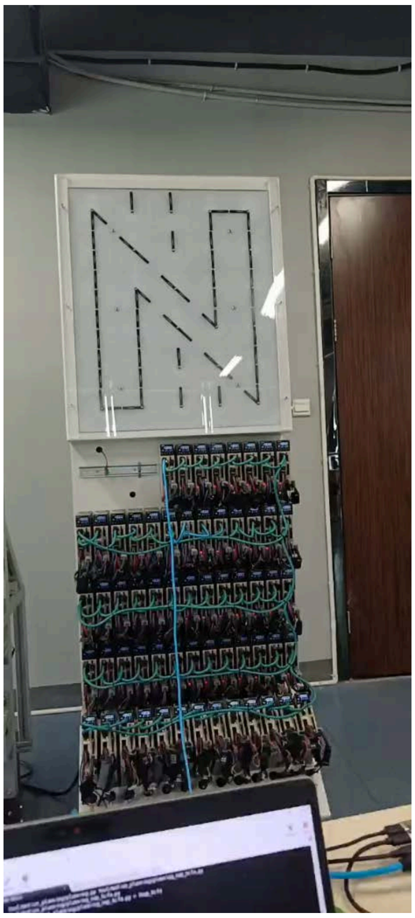


Figure 121. 60-axes servo system built by HCFA

HCFA performance

The task period is 1 ms, and servo control mode is CSP.

- Native EtherCAT driver + IGH stack: 26 μs
- Schedule latency: 200 μs on i.MX 8MP, 220 μs on i.MX 8M Mini
- Link propagation latency: 64 μs
- Customer task: 690-700 μs saved for app

Running this case (HCFA)

All software associated with this controller is integrated in Real-time Edge by default, and can be set up using the below steps:

- Start the nservo service as below:

```
[root]# nservo_run -f /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:"/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 motors:
  After the trajectory planning information file is loaded, use the below command to start the system.

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

6.1.4 SOEM EtherCAT MainDevice

Simple Open EtherCAT MainDevice (SOEM) is an open source EtherCAT MainDevice stack, designed to be easy to use and maintains a small footprint. The optimized native driver is implemented by building upon this stack and introduces various profiles and multi-axes use cases. These use cases are elaborated below:

- Native Ethernet driver for Ethernet controller of i.MX 8M Plus, i.MX 8M Mini, i.MX 93, i.MX943, i.MX95, and i.MX943
- HAL layer for FreeRTOS and BareMetal
- CiA401 profile for digital IO
- CiA402 profile for motor control
- Multi-axes use case with market available servos and **NXP XSERVO-MTR-PSG Board** as the EtherCAT SubDevice with different control loop time

SOEM can run on BM/FreeRTOS on MCU, Cortex-M core of MPU, also can run FreeRTOS/Zephyr on Cortex-A core of MPU.

Refer to MCUXpresso SDK for running SOEM on the supported NXP MCUs. On Real-time Edge software, SOEM examples can be run on Cortex-M Core and Cortex-A Core through the Heterogeneous Multicore Framework (see Section 3.9).

SOEM use cases

SOEM is supported on the below platforms:

- i.MX 8M Plus LPDDR4 EVK platform
- i.MX 8M Mini LPDDR4 EVK platform
- i.MX 93 EVK platform
- i.MX 95-15x15-lpddr4x EVK plarform
- i.MX 95-19x19-lpddr5 EVK plarform
- i.MX 943-19x19-lpddr4 EVK platform
- i.MX 943-19x19-lpddr5 EVK platform

For the i.MX 8M Plus as an example, it is equipped with four Cortex-A53 cores and one Cortex-M7 core. Table 96 describes the use cases that can be executed on this MPU platform.

Table 96. SOEM use cases

SOEM examples	M7	A53	A53	A53	A53
digital_io	baremetal/FreeRTOS	FreeRTOS/Zephyr			
servo_motor	baremetal/FreeRTOS	FreeRTOS/Zephyr			
servo_motor_rt1180	baremetal/FreeRTOS	FreeRTOS/Zephyr			

Three examples are provided in this release:

- SOEM `digital_io` example for IO control using BECKHOFF EK1100 as the EtherCAT SubDevice.
- SOEM `servo_motor` example using Inovance SV680 servo as the EtherCAT SubDevice.
- SOEM `servo_motor_rt1180` example using **NXP XSERVO-MTR-PSG Board** as the EtherCAT SubDevice, which is the i.MX RT1180 Multi-Motor Control Board.

MCUXpresso SDK supports running SOEM on NXP MCUs using Real-time Edge software. For details of the process, refer to MCUXpresso SDK documentation. You can run SOEM examples on the Cortex-M and Cortex-A Cores through the Heterogeneous Multicore Framework.

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

6.1.4.1.1 Setup hardware environment

For the **digital_io** example, the below hardware is required:

- 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
  - 24 V DC power supply

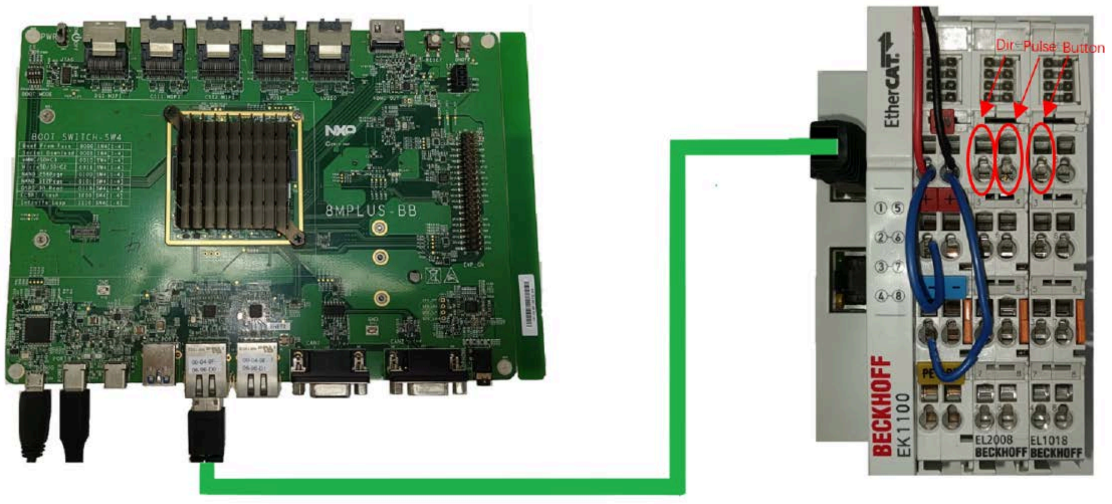


Figure 122. Running the digital_io example on i.MX 8M Plus LPDDR4 EVK

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.

For **servo_motor** and **servo_motor_rt1180** examples, the below hardware is required:

- i.MX 8M Plus LPDDR4 EVK platform
- Inovance SV680 / **NXP XSERVO-MTR-PSG Board**
- Network Cable

There are two types of servos supported, Inovance SV680 and **NXP XSERVO-MTR-PSG Board** as EtherCAT SubDevice.

1. **Inovance SV680 as EtherCAT SubDevice**: Running SOEM Servo Motor example on i.MX 8M Plus EVK to control Inovance SV680.

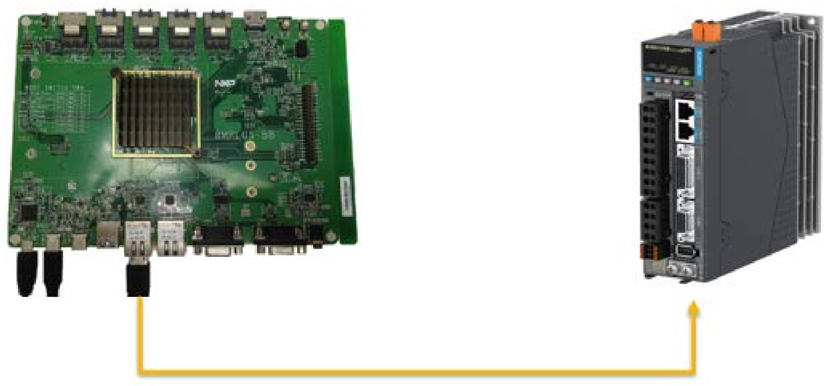


Figure 123. Running SOEM Servo Motor example on i.MX 8M Plus EVK (Inovance SV680 as EtherCAT SubDevice)

2. **Running motion_control example on NXP XSERVO-MTR-PSG Board as EtherCAT SubDevice**. Running SOEM servo_motor_rt1180 example on i.MX 8M Plus EVK to control motor on **NXP XSERVO-MTR-PSG Board**, which is RT1180 Multi-Motor Control Board.

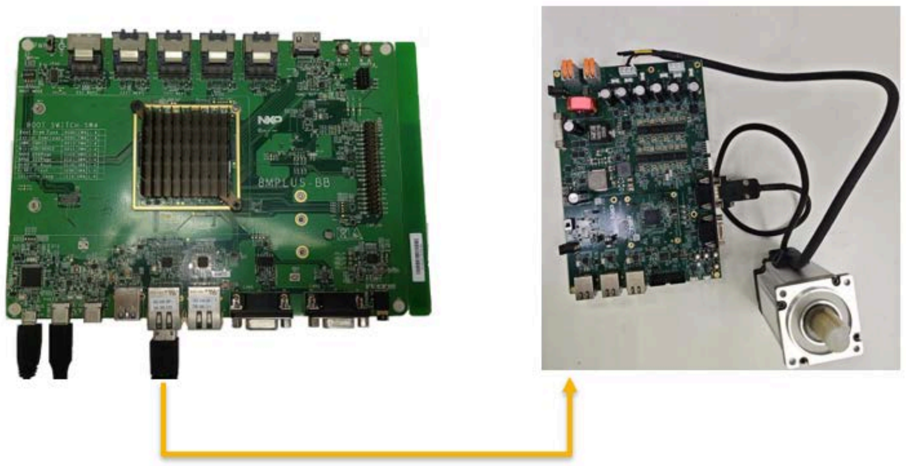


Figure 124. Running servo_motor_rt1180 example on i.MX 8M Plus EVK (NXP XSERVO-MTR-PSG as EtherCAT SubDevice)

Note: *Before running servo_motor example, you must download the motor control source code for NXP XSERVO-MTR-PSG Board from github: https://github.com/nxp-appcodehub/rd-motion-control-slave-servo-mimxrt1180*

6.1.4.1.2 Building the demo images of Cortex-M

For the **digital_io** example, the demo images "`soem_gpio_pulse_freertos`" and "`soem_gpio_pulse_bm`" are compiled with the i.MX 8M Plus LPDDR4 EVK target image compiling, and are installed in the "`/example`" directory of the target rootfs. For JTAG download, these images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx8mpevk/examples/`" on the build host.

```
cd <image-build-dir>/
```

```
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/mcuxsdk/soem-gpio-pulse-bm
|── soem_gpio_pulse_bm_cm7.elf
|── soem_gpio_pulse_bm_cm7.bin
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/mcuxsdk/soem-gpio-pulse-
freertos
|── soem_gpio_pulse_freertos_cm7.elf
|── soem_gpio_pulse_freertos_cm7.bin
```

Also, the demo images can be compiled using the below command on i.MX 8M Plus LPDDR4 EVK build directory:

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

Or

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

For **servo_motor** and **servo_motor_rt1180** example, the demo images "`soem_servo_motor_freertos`", "`soem_servo_motor_bm`", “`soem_servo_motor_rt1180_freertos`” and "`soem_servo_motor_rt1180_bm`" are compiled with the i.MX 8M Plus LPDDR4 EVK target image. These demo images are installed into the "`/example`" directory of the target rootfs. For JTAG download, these images can also be found in the "`<image-build-dir>/tmp/deploy/images/imx8mpevk/examples/`" directory on the build host.

Note: *`soem_servo_motor_freertos` and `soem_servo_motor_bm` are used for the **Inovance SV680** use case whereas “soem_servo_motor_rt1180_freertos” and "soem_servo_motor_rt1180_bm" are used for the **NXP XSERVO-MTR-PSG Board** use case.*

***Inovance SV680** use case*

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/mcuxsdk/soem-servo-motor-bm
|── soem_servo_motor_bm_cm7.elf
|── soem_servo_motor_bm_cm7.bin
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/mcuxsdk/soem-servo-motor-
freertos
|── soem_servo_motor_freertos_cm7.elf
|── soem_servo_motor_freertos_cm7.bin
```

***NXP XSERVO-MTR-PSG Board** use case*

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/mcuxsdk/soem-servo-motor-
rt1180-bm
|── soem_servo_motor_rt1180_bm_cm7.elf
|── soem_servo_motor_rt1180_bm_cm7.bin
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/mcuxsdk/soem-servo-motor-
rt1180-freertos
|── soem_servo_motor_rt1180_freertos_cm7.elf
|── soem_servo_motor_rt1180_freertos_cm7.bin
```

The demo images can also be compiled using the below commands on i.MX 8M Plus LPDDR4 EVK build directory:

```
bitbake soem-servo-motor-bm
or
bitbake soem-servo-motor-freertos
```

```
or
bitbake soem-servo-motor-rt1180-bm
or
bitbake soem-servo-motor-rt1180-freertos
```

6.1.4.1.3 Building the demo images of Cortex-A

For the **digital_io** example, the demo images "`soem_digital_io`" are compiled along with the i.MX 8M Plus LPDDR4 EVK target images, and are installed into the directory "/example" of the target rootfs. Images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx8mpevk/examples/`" on the build host.

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mpevk/examples/heterogeneous-multicore/soem-digital-
io-freertos
|── soem_digital_io_ca53.elf
|── soem_digital_io_ca53.bin

tree tmp/deploy/images/imx8mpevk/examples/heterogeneous-multicore/soem-digital-
io-zephyr
|── soem_digital_io_ca53.elf
|── soem_digital_io_ca53.bin
```

The demo images can also be compiled using the below command on the i.MX 8M Plus LPDDR4 EVK build directory:

```
bitbake soem-digital-io
```

For the **servo_motor** and the **servo_motor_rt1180** example, the demo images "`soem_servo_ca53`", and "`soem_servo_rt1180_ca53`" are compiled with the i.MX 8M Plus LPDDR4 EVK target image compiling. They are installed into the "/example" directory of the target rootfs. Images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx8mpevk/examples/`" on the build host.

Note: *The "soem_servo_ca53" image is for the "Inovance SV680" use case. The "soem_servo_rt1180_ca53" images are for the "NXP XSERVO-MTR-PSG Board" case.*

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/heterogeneous-multicore/soem-
servo-freertos
|── soem_servo_ca53.elf
|── soem_servo_ca53.bin
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/heterogeneous-multicore/soem-
servo-zephyr
|── soem_servo_ca53.elf
|── soem_servo_ca53.bin
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/heterogeneous-multicore/soem-
servo-rt1180-freertos
|── soem_servo_rt1180_ca53.elf
|── soem_servo_rt1180_ca53.bin
tree tmp/deploy/images/imx8mp-lpddr4-evk/examples/heterogeneous-multicore/soem-
servo-rt1180-zephyr
|── soem_servo_rt1180_ca53.elf
|── soem_servo_rt1180_ca53.bin
```

Also, the demo images can be compiled using the below command on i.MX 8M Plus LPDDR4 EVK build directory:

```
bitbake soem-servo
or
bitbake soem-servo-rt1180
```

6.1.4.1.4 Running SOEM demo images using J-Link GDB Server

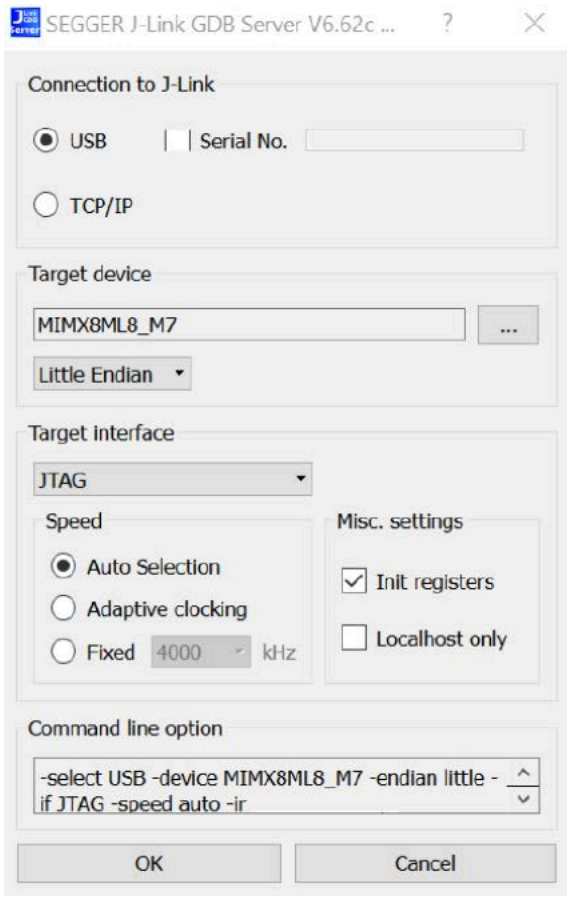
This section describes the steps to run a demo application using J-Link GDB Server application. After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the i.MX 8M Plus LPDDR4 EVK platform to your PC via USB cable between the USB-UART connector and the PC USB connector. If using a standalone J-Link debug pod, also connect it to the SWD/JTAG connector of the board.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug serial port number. Configure the terminal with these settings:
   - 115200 baud rate
   - No parity
   - 8 data bits
   - 1 stop bit
3. Open the J-Link GDB Server application on Linux host. If the OS of your PC is Linux, use the below configuration to set up GDB Server. Assuming that the J-Link software is installed, launch the application 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 the 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 the server is launched, modify the settings as below. The target device selected for this demo is MIMX8ML8_M7.



Connection to J-Link: [✓ USB] Serial No. | TCP/IP

Target device: MIMX8ML8_M7

Little Endian

Target interface: JTAG

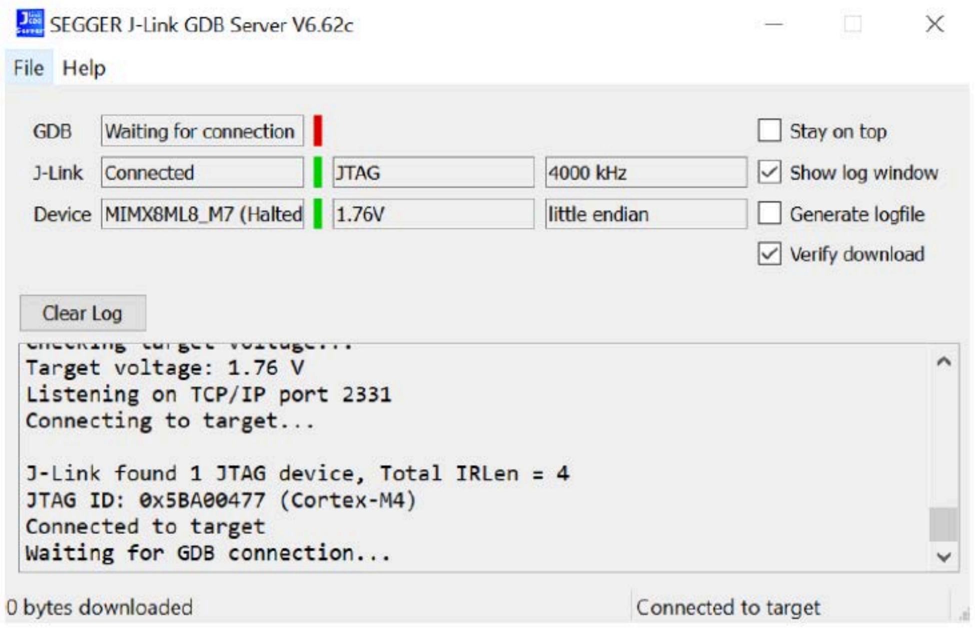
Speed: [✓ Auto Selection] | Adaptive clocking | Fixed 4000 kHz

Misc. settings: [✓ Init registers] | Localhost only

Command line option: -select USB -device MIMX8ML8_M7 -endian little -if JTAG -speed auto -ir

Figure 125. Setting up the GDB Server

5. After the GDB server is running, the screen looks like as below:



GDB: Waiting for connection
J-Link: Connected | JTAG | 4000 kHz
Device: MIMX8ML8_M7 (Halted | 1.76V | little endian

☐ Stay on top
☑ Show log window
☐ Generate logfile
☑ Verify download

```
Target voltage: 1.76 V
Listening on TCP/IP port 2331
Connecting to target...

J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x5BA00477 (Cortex-M4)
Connected to target
Waiting for GDB connection...
```

0 bytes downloaded | Connected to target

Figure 126. GDB server running

6. Start the GDB client:
7. Ensure that the `arm-none-eabi-gdb` is installed, 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:
<https://www.gnu.org/software/gdb/bugs/>
Find the GDB manual and other documentation resources online at:
<https://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)
```

8. 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
```

9.

```
(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 now downloads and stops at the reset vector. Execute the `monitor go` command to start the demo application.

```
(gdb) monitor go
```

6.1.4.1.5 Running SOEM demo images using U-Boot (i.MX 8MP)

This section describes the steps to write an 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** and 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, identify the TTY device with the name **/dev/ttyUSB*** to determine your debug port. Similar to Windows OS, opening both is beneficial for development.
3. Build and flash the `nxp-image-real-time-edge` image to an SD card and insert the SD card to the target board. Make sure to use the default boot SD slot and check the DIP boot switch configuration.
4. Open your preferred serial terminal for the serial devices. Set the speed to 115,200 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 the autoboot in the terminal window. Then, enter the U-Boot command-line mode. You can then write the image and run it from TCM or DRAM with the following commands:
6. **Running on Cortex-M**
   a. If the `soem_gpio_pulse.bin` is made from the `<examples/..>/` target, which means the binary file runs at TCM, use the following commands to boot:

   ```
   => ext4load mmc 1:2 0x48000000 /examples/mcuxsdk/soem-gpio-pulse/
   soem_gpio_pulse_bm_cm7.bin;
   => cp.b 0x48000000 0x7e0000 0x20000;
   => bootaux 0x7e0000
   ```

   b. If the `soem_servo_motor.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/mcuxsdk/soem-servo-motor/
   soem_servo_motor_bm_cm7.bin;
   => cp.b 0x48000000 0x7e0000 0x20000;
   => bootaux 0x7e0000
   ```

   Or

   c.
   ```
   => ext4load mmc 1:2 0x48000000 /examples/mcuxsdk/soem-servo-motor-rt1180/
   soem_servo_motor_rt1180_bm_cm7.bin;
   ```

```
=> cp.b 0x48000000 0x7e0000 0x20000;
=> bootaux 0x7e0000
```

Note: *If the Linux OS kernel runs together with M7, make sure that 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
```

If the SOEM app is required to work while Linux is running, modification must be done. M7 core needs exclusive access to the Ethernet. So, remove the Ethernet access from the Linux kernel by performing the following steps:

a. *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
```

b. *Copy the new device tree and the kernel image to the SD bootpartition .*

c. *Now, SOEM still works while Linux is running.*

7. **Running FreeRTOS on Cortex-A**

```
=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/soem-
digital-io-freertos/soem_digital_io_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0xC0000000
```

Or

```
 => ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/soem-servo-
freertos/soem_servo_ca53.bin;
 => dcache flush; icache flush;
 => cpu 2 release 0xC0000000
```

Or

```
=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/soem-servo-
rt1180-freertos/soem_servo_rt1180_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0xC0000000
```

8. **Running Zephyr on Cortex-A**

```
=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/soem-
digital-io-zephyr/soem_digital_io_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0xC0000000
```

Or

```
 => ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/soem-servo-
zephyr/soem_servo_ca53.bin;
 => dcache flush; icache flush;
 => cpu 2 release 0xC0000000
```

Or

```
=> ext4load mmc 1:2 0xC0000000 /examples/heterogeneous-multicore/soem-servo-
rt1180-zephyr/soem_servo_rt1180_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0xC0000000
```

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

6.1.4.2.1 Setting up the hardware environment

For the **digital_io** example, the below hardware is required:

- i.MX 8M Mini LPDDR4 EVK platform
- EK1100 - EtherCAT coupler
- EL2008 - EtherCAT Terminal, 8-channel digital output, 24 V DC
- EL1018 - EtherCAT Terminal, 8-channel digital input, 24 V DC
- 24 V DC power supply

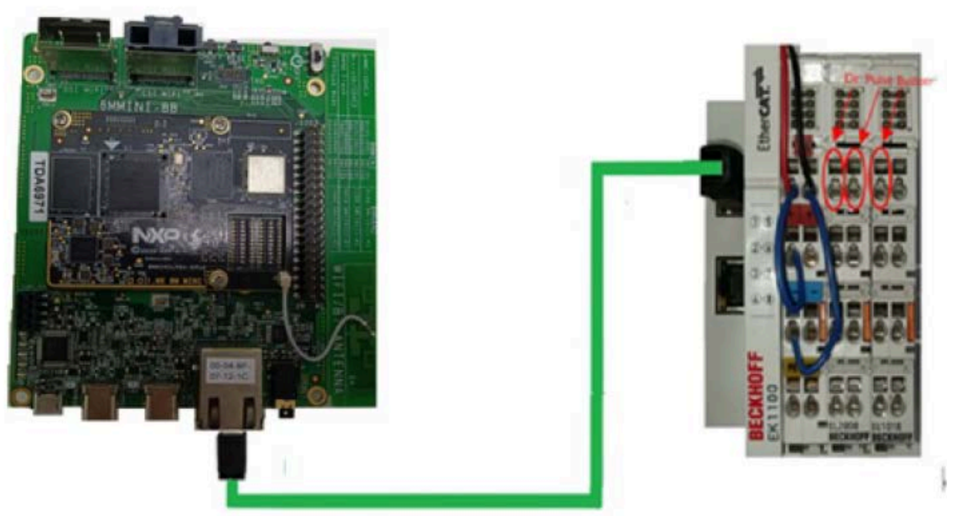


Figure 127. Hardware setup for *digital_io* example on i.MX 8M Mini LPDDR4 EVK board

The channel-1 of the EL1018 is connected to a button to generate a 24 V pulse. This pulse is used as an input signal to change the output of the channel-1 of the EL2008 labeled as "Dir". The channel-2 of the EL2008 outputs a square wave signal with a period of 250 μs.

For the**servo_motor** and the **servo_motor_rt1180** example, the below hardware is required:

- i.MX 8M Mini LPDDR4 EVK platform
- Inovance SV680 / **NXP XSERVO-MTR-PSG Board**
- Network cable

There are two ways to build the example:

1. Inovance SV680 as EtherCAT SubDevice. Running SOEM Servo Motor example on i.MX 8M Plus EVK to control Inovance SV680.

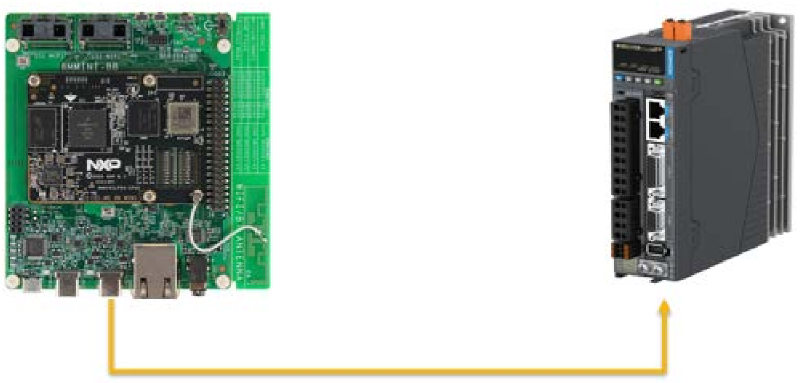


Figure 128. Running SOEM Servo Motor example on i.MX 8M Mini LPDDR4 EVK board (using Inovance SV680 as EtherCAT SubDevice)

Or

1. Running `motion_control` example on NXP XSERVO-MTR-PSG Board as EtherCAT SubDevice. Running SOEM servo_motor_rt1180 example on i.MX 8M Plus EVK to control motor on NXP XSERVO-MTR-PSG Board, which is RT1180 Multi-Motor Control Board.

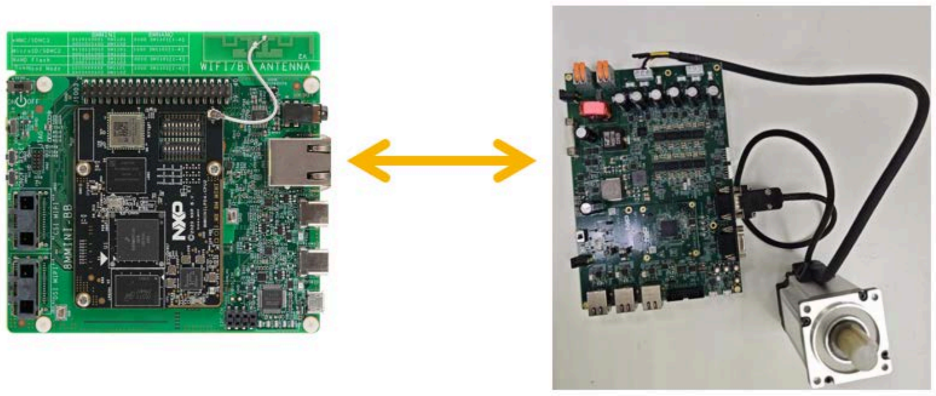


Figure 129. Running SOEM servo_motor_rt1180 example on i.MX 8M Mini LPDDR4 EVK board (using NXP XSERVO-MTR-PSG Board as EtherCAT SubDevice)

Note: *Before running servo_motor example, you must download the motor control source code for NXP XSERVO-MTR-PSG Board from github: https://github.com/nxp-appcodehub/rd-motion-control-slave-servo-mimxrt1180*

6.1.4.2.2 Building the demo images of Cortex-M

For **digital_io** example, the demo images "`freertos_soem_gpio_pulse`" and "`soem_gpio_pulse`" are compiled with the i.MX 8M Mini LPDDR4 EVK target image compiling, and are installed into the directory "/example" of the target rootfs. For JTAG download, these images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx8mmevk/examples/`" on the build host.

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/mcuxsdk/soem-gpio-pulse-bm
|── soem_gpio_pulse_bm_cm4.elf
|── soem_gpio_pulse_bm_cm4.bin
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/mcuxsdk/soem-gpio-pulse-
freertos
|── soem_gpio_pulse_freertos_cm4.elf
|── soem_gpio_pulse_freertos_cm4.bin
```

Also, the demo images can be compiled using the below command on i.MX 8M Mini LPDDR4 EVK build directory:

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

Or

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

For **servo_motor** and **servo_motor_rt1180** example, the demo images "`soem_servo_motor_freertos`", "`soem_servo_moto_bm`r", “`soem_servo_motor_rt1180_freertos`” and "`soem_servo_motor_rt1180_bm`" are compiled with the i.MX 8M Mini LPDDR4 EVK target image compiling, and they are installed into the directory "`/example`" of the target rootfs. For JTAG download, these images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx8mmevk/examples/`" on building host.

Note: *"`soem_servo_motor_freertos`", "`soem_servo_motor_bm`" are for “Inovance SV680” case, “soem_servo_motor_rt1180_freertos” and "soem_servo_motor_rt1180_bm" are for* ***NXP XSERVO-MTR-PSG Board*** *case.*

Inovance SV680 *use case*

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/mcuxsdk/soem-servo-motor-bm
|── soem_servo_motor_bm_cm4.elf
|── soem_servo_motor_bm_cm4.bin
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/mcuxsdk/soem-servo-motor-
freertos
|── soem_servo_motor_freertos_cm4.elf
|── soem_servo_motor_freertos_cm4.bin
```

NXP XSERVO-MTR-PSG Board use case

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/mcuxsdk/soem-servo-motor-
rt1180-bm
|── soem_servo_motor_rt1180_bm_cm4.elf
|── soem_servo_motor_rt1180_bm_cm4.bin
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/mcuxsdk/soem-servo-motor-
rt1180-freertos
|── soem_servo_motor_rt1180_freertos_cm4.elf
|── soem_servo_motor_rt1180_freertos_cm4.bin
```

Also, the demo images can be compiled using the below command on i.MX 8M Mini LPDDR4 EVK build directory:

```
bitbake soem-servo-motor-bm
or
bitbake soem-servo-motor-freertos
or
bitbake soem-servo-motor-rt1180-bm
```

```
or
bitbake soem-servo-motor-rt1180-freertos
```

6.1.4.2.3 Building the demo images of Cortex-A

For **digital_io** example, the demo images "`soem_digital_io_ca53`" are compiled along with the i.MX 8M Mini LPDDR4 EVK target image compilation, and are installed into the directory "`/example`" of the target rootfs. Images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx8mmevk/examples/`" on the build host.

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/heterogeneous-multicore/soem-
digital-io-freertos
|── soem_digital_io_ca53.elf
|── soem_digital_io_ca53.bin

tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/heterogeneous-multicore/soem-
digital-io-zephyr
|── soem_digital_io_ca53.elf
|── soem_digital_io_ca53.bin
```

Also, the demo images can be compiled using the below command on i.MX 8M Mini LPDDR4 EVK build directory:

```
bitbake soem-digital-io
```

For **servo_motor** and **servo_motor_rt1180** example, the demo images "soem_servo_ca53", “soem_servo_rt1180_ca53” are compiled along with the i.MX 8M Mini LPDDR4 EVK target image, and are installed into the directory "`/example`" of the target rootfs. Images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx8mmevk/examples/`" on building host.

Note: *"`soem_servo_ca53`" file is for “`Inovance SV680`” case, whereas the "`soem_servo_rt1180_ca53`" file is for "`NXP XSERVO-MTR-PSG Board`" case.*

```
cd <image-build-dir>/
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/heterogeneous-multicore/soem-
servo-freertos
|── soem_servo_ca53.elf
|── soem_servo_ca53.bin
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/heterogeneous-multicore/soem-
servo-zephyr
|── soem_servo_ca53.elf
|── soem_servo_ca53.bin
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/heterogeneous-multicore/soem-
servo-rt1180-freertos
|── soem_servo_rt1180_ca53.elf
|── soem_servo_rt1180_ca53.bin
tree tmp/deploy/images/imx8mm-lpddr4-evk/examples/heterogeneous-multicore/soem-
servo-rt1180-zephyr
|── soem_servo_rt1180_ca53.elf
|── soem_servo_rt1180_ca53.bin
```

The demo images can also be compiled using the below command on i.MX 8M Mini LPDDR4 EVK build directory:

```
bitbake soem-servo
or
```

```
bitbake soem-servo-rt1180
```

6.1.4.2.4 Running SOEM demo images using J-Link GDB Server

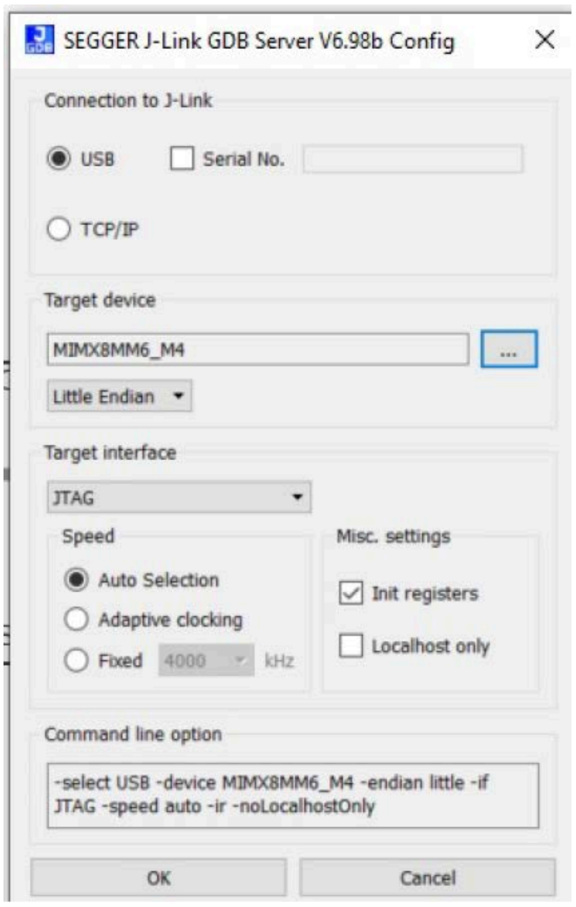
This section describes the steps to run a demo application using J-Link GDB Server application. After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

1. Connect the i.MX 8M 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:
   a. 115200 baud rate
   b. No parity
   c. 8 data bits
   d. 1 stop bit
3. Open the J-Link GDB Server application on Linux host. If the OS of your PC is Linux, using the below configuration to set up GDB Server. Assuming the J-Link software is installed, the application can be launched from a new terminal for the MIMX8MM6_M4 device:

```

$ JLinkGDBServer -if JTAG -device
SEGGER J-Link GDB Server Command Line Version
JLinkARM.dll
Command line: -if JTAG -device MIMX8MM6_M4
-----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.
   - Ensure that the J-Link software is installed
   - Launch the application 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 shown in Figure 130. The target device selected for this demo is **MIMX8MM6_M4**.

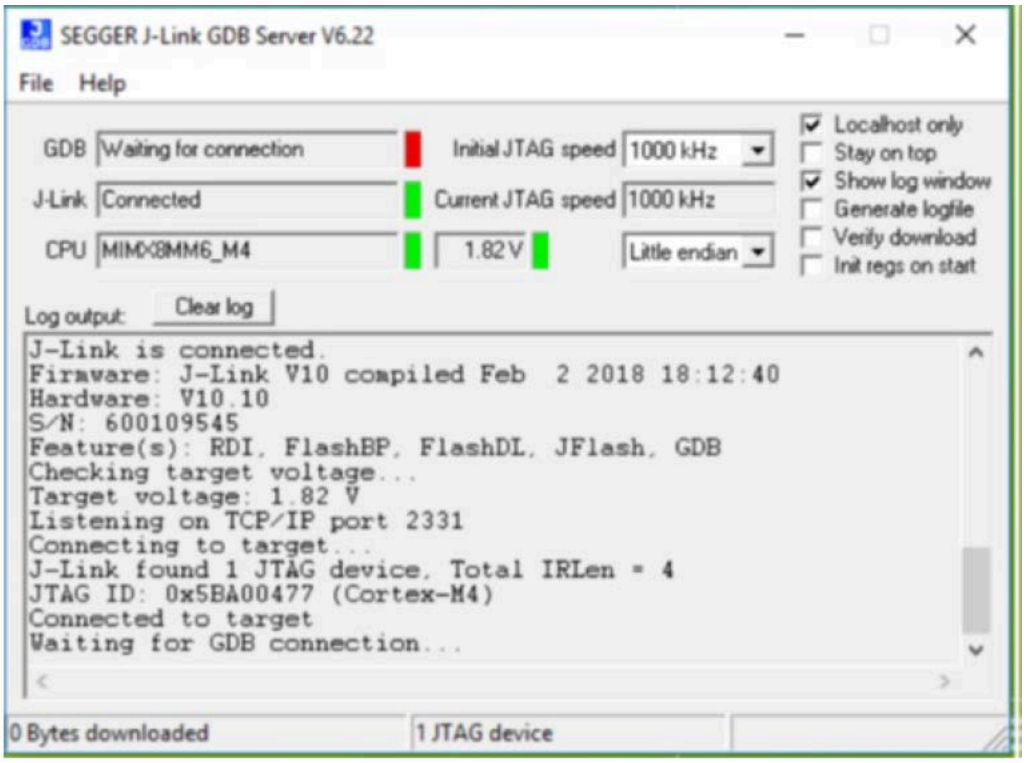

SEGGER J-Link GDB Server V6.98b Config

Connection to J-Link: USB [✓] ; Serial No. [ ] ; TCP/IP
Target device: MIMX8MM6_M4
Little Endian
Target interface: JTAG
Speed: Auto Selection [✓] ; Adaptive clocking ; Fixed 4000 kHz
Misc. settings: Init registers [✓] ; Localhost only [ ]
Command line option:

```
-select USB -device MIMX8MM6_M4 -endian little -if
JTAG -speed auto -ir -noLocalhostOnly
```

Figure 130. J-Link GDB Server application window

5. After GDB server begins to run, the screen looks as in Figure 131:



GDB: Waiting for connection
J-Link: Connected
CPU: MIMX8MM6_M4
1.82 V
Initial JTAG speed: 1000 kHz
Current JTAG speed: 1000 kHz
Little endian
[✓] Localhost only
[ ] Stay on top
[✓] Show log window
[ ] Generate logfile
[ ] Verify download
[ ] Init regs on start

Log output:

```
J-Link is connected.
Firmware: J-Link V10 compiled Feb  2 2018 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-M4)
Connected to target
Waiting for GDB connection...
```

0 Bytes downloaded | 1 JTAG device

Figure 131. J-Link connected and GDB server running

6. Start the GDB client
   - Ensure that the `arm-none-eabi-gdb` is installed
   - 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)
```

7. Connect to the GDB server and load the binary by running the following commands:

```
a. target remote <GDB Server IP>:2331
b. monitor reset <p>
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)
```

8. The application now downloads and stops at the reset vector. Start the demo application by executing the `monitor go` command:

```
gdb) monitor go
```

6.1.4.2.5 Running SOEM demo images by using U-Boot (i.MX 8M Mini)

- **Running on Cortex-M**
  The steps to write SOEM demo image file to TCM or DRAM using the Real-time Edge image on i.MX 8M Mini LPDDR4 EVK are same as followed for i.MX 8M Plus LPDDR4 EVK board. Refer Section 6.1.4.1.5.
- **Running FreeRTOS on Cortex-A**

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/soem-digital-io-freertos/soem_digital_io_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0x93C00000
```

Or

```

=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/soem-servo-freertos/soem_servo_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0x93C00000
```

Or

```

=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/soem-servo-rt1180-freertos/soem_servo_rt1180_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0x93C00000
```

Running Zephyr on Cortex-A

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/soem-digital-io-zephyr/soem_digital_io_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0x93C00000
```

Or

```
=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/soem-servo-
zephyr/soem_servo_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0x93C00000
```

Or

```

=> ext4load mmc 1:2 0x93C00000 /examples/heterogeneous-multicore/soem-servo-
rt1180-zephyr/soem_servo_rt1180_ca53.bin;
=> dcache flush; icache flush;
=> cpu 2 release 0x93C00000
```

6.1.4.3 SOEM for i.MX 93 EVK platform

6.1.4.3.1 Setting up the hardware environment

6.1.4.3.1.1 digital_io example

For **digital_io** example, the below hardware are required:

- i.MX 93 EVK platform
- EK1100 - EtherCAT Coupler
- EL2008 - EtherCAT Terminal, 8-channel digital output, 24V
- DCEL1018 - EtherCAT Terminal, 8-channel digital input, 24 V DC
- 24 V DC Power Supply

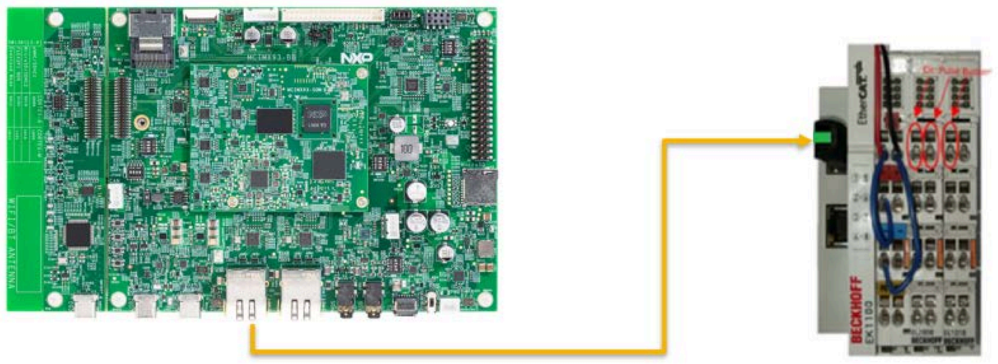


Figure 132. digital_io example for i.MX 93-EVK board

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.

6.1.4.3.1.2 servo_motor andservo_motor_rt1180 examples

For **servo_motor** and **servo_motor_rt1180** example, the below hardware are required:

- i.MX 93 EVK platform
- Inovance SV680 / NXP XSERVO-MTR-PSG Board
- Network cable

There are two ways to build the example:

1. Inovance SV680 as EtherCAT SubDevice. Running SOEM Servo Motor example on i.MX 93 EVK to control Inovance SV680.

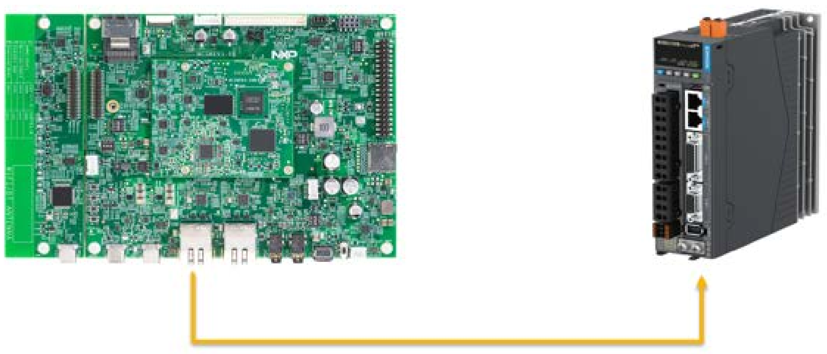


Figure 133. Inovance SV680 as EtherCAT SubDevice

   Or

2. Running `motion_control` example on NXP XSERVO-MTR-PSG Board as EtherCAT SubDevice. Running SOEM servo_motor_rt1180 example on i.MX 93 EVK to control motor on NXP XSERVO-MTR-PSG Board, which is RT1180 Multi-Motor Control Board.

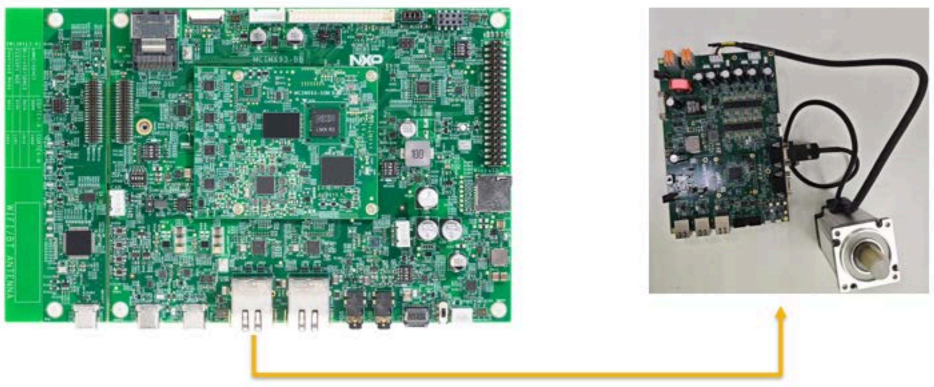


Figure 134. Running SOEM servo_motor_rt1180 example on i.MX 93 EVK board (using NXP XSERVO-MTR-PSG Board as EtherCAT SubDevice)

Note: *Before running the servo_motor example, you must download the motor control source code for NXP XSERVO-MTR-PSG Board from github: https://github.com/nxp-appcodehub/rd-motion-control-slave-servo-mimxrt1180*

6.1.4.3.2 Building the demo images of Cortex-M

For **servo_motor** and **servo_motor_rt1180** examples, the demo images "`soem_servo_motor`" and "`soem_servo_motor_rt1180`" are compiled along with the i.MX 93 EVK target image. These images are installed in the "/example" directory of the target rootfs. For JTAG download, these images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx93evk/examples/`" on building host.

Note: *"soem_servo_motor" are for "Inovance SV680" case, "soem_servo_motor_rt1180" are for "NXP XSERVO-MTR-PSG Board" case.*

```
cd <image-build-dir>/
tree tmp/deploy/images/imx93evk/examples/mcux-sdk/soem-servo-motor-bm
|── soem_servo_motor_bm_cm33.elf
|── soem_servo_motor_bm_cm33.bin
```

```
cd <image-build-dir>/
tree tmp/deploy/images/imx93evk/examples/mcux-sdk/soem-servo-motor-rt1180-bm
|── soem_servo_motor_rt1180_bm_cm33.elf
|── soem_servo_motor_rt1180_bm_cm33.bin
```

Also, the demo images can be compiled using the below command on i.MX 93 EVK build directory:

```
bitbake soem-servo-motor-bm
or
bitbake soem-servo-motor-rt1180-bm
```

6.1.4.3.3 Building the demo images of Cortex-A

6.1.4.3.3.1 digital_io example

For **digital_io** example, the demo images "soem_digital_io_ca55" are compiled with the i.MX 93 EVK target image compiling, and they are installed into the directory "/example" of the target rootfs. Images can also be found in the directory "<image-build-dir>/tmp/deploy/images/imx93evk/examples/" on building host.

```
tree tmp/deploy/images/imx93evk/examples/heterogeneous-multicore/soem-digital-
io-freertos
|── soem_digital_io_ca55.elf
|── soem_digital_io_ca55.bin

tree tmp/deploy/images/imx93evk/examples/heterogeneous-multicore/soem-digital-
io-zephyr
|── soem_digital_io_ca55.elf
|── soem_digital_io_ca55.bin
```

Also, the demo images can be compiled using the below command on i.MX 93 EVK build directory:

```
bitbake soem-digital-io
```

6.1.4.3.3.2 servo_motor and servo_motor_rt1180 examples

For **servo_motor** and **servo_motor_rt1180** example, the demo images "`soem_servo_ca55`", “`soem_servo_rt1180_ca55`” are compiled with the i.MX 93 EVK target image compiling, and they are installed into the directory "/example" of the target rootfs. Images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx93evk/examples/`" on building host.

Note: *"soem_servo_ca55" are for “Inovance SV680” case, "soem_servo_rt1180_ca55" are for “NXP XSERVO-MTR-PSG Board” case.*

```
cd <image-build-dir>/
```

```
tree tmp/deploy/images/imx93evk/examples/heterogeneous-multicore/soem-servo-
freertos
|── soem_servo_ca55.elf
|── soem_servo_ca55.bin
tree tmp/deploy/images/imx93evk/examples/heterogeneous-multicore/soem-servo-
zephyr
|── soem_servo_ca55.elf
|── soem_servo_ca55.bin
tree tmp/deploy/images/imx93evk/examples/heterogeneous-multicore/soem-servo-
rt1180-freertos
|── soem_servo_rt1180_ca55.elf
|── soem_servo_rt1180_ca55.bin
tree tmp/deploy/images/imx93evk/examples/heterogeneous-multicore/soem-servo-
rt1180-zephyr
|── soem_servo_rt1180_ca55.elf
|── soem_servo_rt1180_ca55.bin
```

Also, the demo images can be compiled using the below command on i.MX 93 EVK build directory:

```

bitbake soem-servo
or
bitbake soem-servo-rt1180
```

6.1.4.3.4 Running SOEM demo images by using U-Boot

This section describes the steps to write an SOEM demo image file to the 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.
   a. 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-A55 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.
   b. Build and flash the `nxp-image-real-time-edge` image to a SD card and insert the SD card to the target board. Make sure to use the default boot SD slot and check the DIP boot switch configuration.
   c. Open your preferred serial terminal for the serial devices. Set the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity. Then, power on the board.
   d. Power on the board and hit any key to stop autoboot in the terminal window. Then enter the U-Boot Command line mode. You can then write the image and run it from TCM or DRAM with the following commands:
2. **Running on Cortex-M** If the `soem_servo_motor.bin` is made from the `<examples/..>/` target, which means the binary file runs at TCM, use the following commands to boot:

   ```
   => ext4load mmc 1:2 0xd0000000 /examples/mcuxsdk/soem-servo-motor/
   soem_servo_motor_bm_cm33.bin;
   => cp.b 0xd0000000 0x201e0000 20000;
   => bootaux 0x1ffe0000
   ```

   Or

   ```
   => ext4load mmc 1:2 0xd0000000 /examples/mcuxsdk/soem-servo-motor-rt1180/
   soem_servo_motor_rt1180_bm_cm33.bin;
   => cp.b 0xd0000000 0x201e0000 20000;
   => bootaux 0x1ffe0000
   ```

3. **Running FreeRTOS on Cortex-A**

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-
digital-io-freertos/soem_digital_io_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
freertos/soem_servo_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
rt1180-freertos/soem_servo_rt1180_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Running Zephyr on Cortex-A

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-
digital-io-zephyr/soem_digital_io_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
zephyr/soem_servo_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
rt1180-zephyr/soem_servo_rt1180_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

6.1.4.4 SOEM for i.MX 943 EVK platform

6.1.4.4.1 Setup hardware environment

For the **digital_io** example, the below hardware is required:

- i.MX 943-19x19 LPDDR4 EVK/i.MX 943-19x19 LPDDR5 EVK platform
  - EK1100 - EtherCAT Coupler
  - EL2008 - EtherCAT Terminal, 8-channel digital output, 24V DC
  - EL1018 - EtherCAT Terminal, 8-channel digital input, 24 V DC
  - 24 V DC power supply

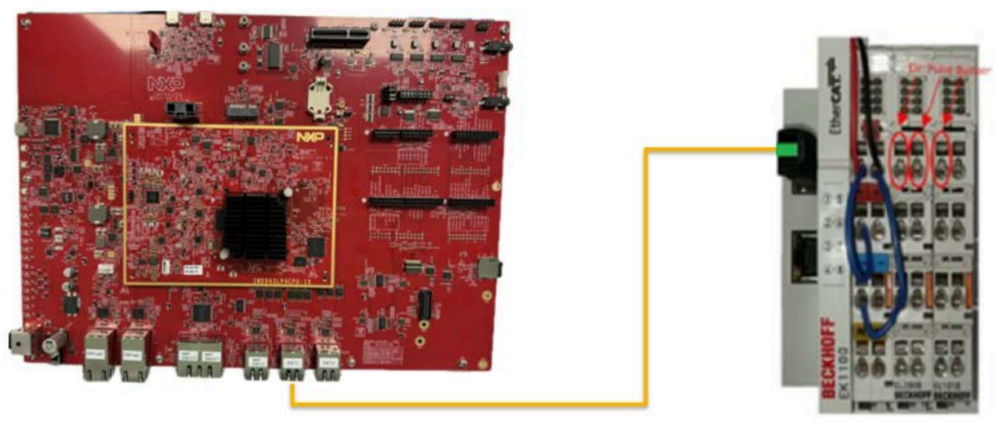


Figure 135. Running the digital_io example on i.MX 943

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.

For **servo_motor** and **servo_motor_rt1180** examples, the below hardware is required:

- i.MX 943-19x19-lpddr4-evk/ i.MX 943-19x19-lpddr5-evk platform
- Inovance SV680 / **NXP XSERVO-MTR-PSG Board**
- Network Cable

There are two types of servos supported, Inovance SV680 and **NXP XSERVO-MTR-PSG Board** as EtherCAT SubDevice.

1. **Inovance SV680 as EtherCAT SubDevice**: Running SOEM Servo Motor example on i.MX 943 EVK to control Inovance SV680.

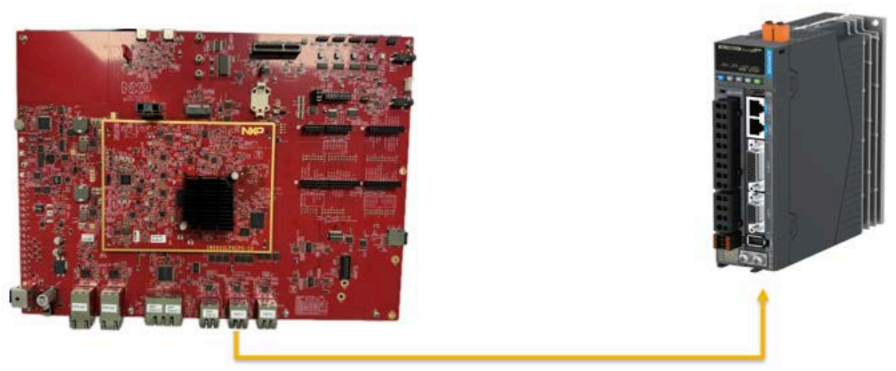


Figure 136. Running SOEM Servo Motor example on i.MX 943 EVK (Inovance SV680 as EtherCAT SubDevice)

2. **Running motion_control example on NXP XSERVO-MTR-PSG Board as EtherCAT SubDevice**. Running SOEM servo_motor_rt1180 example on i.MX 943 EVK to control motor on **NXP XSERVO-MTR-PSG Board**, which is RT1180 Multi-Motor Control Board.

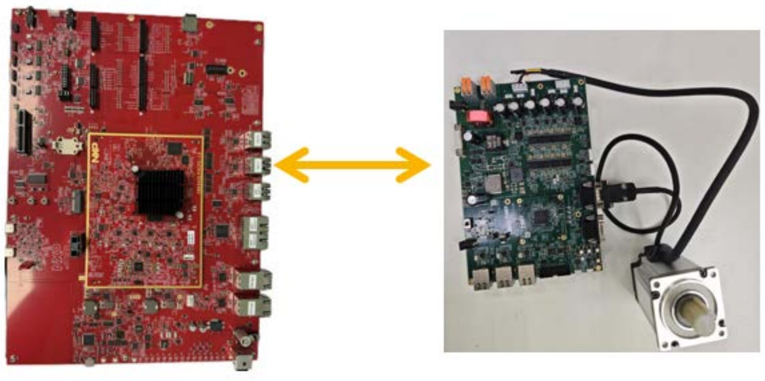


Figure 137. Running servo_motor_rt1180 example on i.MX 943 EVK (NXP XSERVO-MTR-PSG as EtherCAT SubDevice)

Note: *Before running servo_motor example, you must download the MCU-SDK package for NXP XSERVO-MTR-PSG Board from github: https://github.com/nxp-appcodehub/rd-motion-control-slave-servo-mimxrt1180*

6.1.4.4.2 Building the demo images of Cortex-A

6.1.4.4.2.1 digital_io example

For the **digital_io** example, the demo images "soem_digital_io_ca55" are compiled with the i.MX 943 EVK target images and are installed in the "`/example`" directory of the target rootfs. Images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx943-19x19-lpddr5-evk/examples/`" on the build host.

```
tree tmp/deploy/images/imx943-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-digital-io-freertos
|── soem_digital_io_ca55.elf
|── soem_digital_io_ca55.bin

tree tmp/deploy/images/imx943-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-digital-io-zephyr
|── soem_digital_io_ca55.elf
|── soem_digital_io_ca55.bin
```

Also, the demo images can be compiled using the below command on i.MX 943 EVK build directory:

```
bitbake soem-digital-io
```

6.1.4.4.2.2 servo_motor and servo_motor_rt1180 examples

For **servo_motor** and **servo_motor_rt1180** example, the demo images "`soem_servo_ca55`", “`soem_servo_rt1180_ca55`” are compiled with the i.MX 943 EVK target image compiling, and they are installed into the directory "/example" of the target rootfs. Images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx943-19x19-lpddr5-evk/examples/`" on building host.

Note: *"soem_servo_ca55" are for “Inovance SV680” case, "soem_servo_rt1180_ca55" are for “NXP XSERVO-MTR-PSG Board” case.*

```
cd <image-build-dir>/
tree tmp/deploy/images/imx943-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-servo-freertos
|── soem_servo_ca55.elf
|── soem_servo_ca55.bin
tree tmp/deploy/images/imx943-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-servo-zephyr
|── soem_servo_ca55.elf
|── soem_servo_ca55.bin
tree tmp/deploy/images/imx943-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-servo-rt1180-freertos
|── soem_servo_rt1180_ca55.elf
|── soem_servo_rt1180_ca55.bin
tree tmp/deploy/images/imx943-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-servo-rt1180-zephyr
|── soem_servo_rt1180_ca55.elf
|── soem_servo_rt1180_ca55.bin
```

Also, the demo images can be compiled using the below command on the i.MX 943 EVK build directory:

```
bitbake soem-servo
or
bitbake soem-servo-rt1180
```

6.1.4.4.3 Running SOEM demo images by using U-Boot

This section describes the steps to write an SOEM demo image file to the 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.
   a. On Windows OS, open the Device Manager, find USB serial Port in Ports (COM and LPT). Assume that the ports are COM7 COM8 COM9 and COM10. One port is for the debug message from the Cortex-A55 and the other is for the Cortex-M33. 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.
   b. Build and flash the `nxp-image-real-time-edge` image to a SD card and insert the SD card to the target board. Make sure to use the default boot SD slot and check the DIP boot switch configuration.
   c. Open your preferred serial terminal for the serial devices. Set the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity. Then, power on the board.
   d. Power on the board and hit any key to stop autoboot in the terminal window. Then enter the U-Boot Command line mode. You can then write the image and run it from TCM or DRAM with the following commands:
2. **Running FreeRTOS on Cortex-A**

   ```
   => ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-
   digital-io-freertos/soem_digital_io_ca55.bin;
   => dcache flush; icache flush;
   => cpu 1 release 0xD0000000
   ```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
freertos/soem_servo_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
rt1180-freertos/soem_servo_rt1180_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Running Zephyr on Cortex-A

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-
digital-io-zephyr/soem_digital_io_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
zephyr/soem_servo_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
rt1180-zephyr/soem_servo_rt1180_ca55.bin;
=> dcache flush; icache flush;
=> cpu 1 release 0xD0000000
```

6.1.4.5 SOEM for i.MX 95 EVK platform

6.1.4.5.1 Setting up the hardware environment

For the **digital_io** example, the below hardware is required:

- i.MX 95-15x15 LPDDR4x EVK/i.MX 95-19x19 LPDDR5 EVK platform
  - EK1100 - EtherCAT Coupler
  - EL2008 - EtherCAT Terminal, 8-channel digital output, 24V DC
  - EL1018 - EtherCAT Terminal, 8-channel digital input, 24 V DC
  - 24 V DC power supply

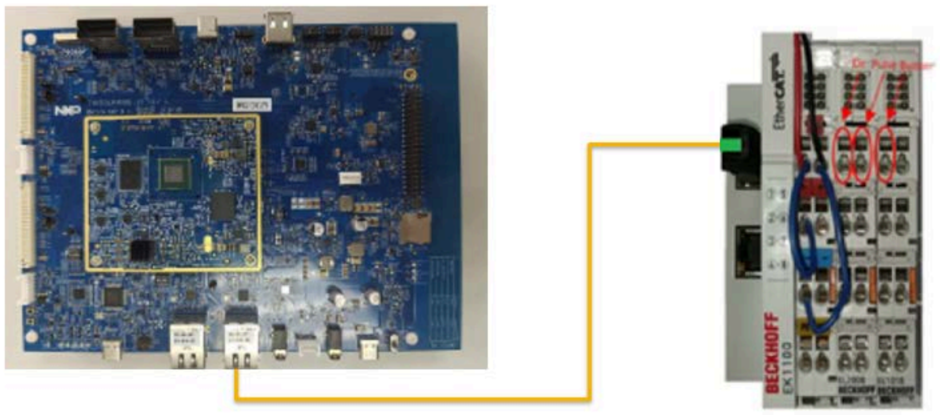


Figure 138. Running the digital_io example on i.MX 95

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.

For **servo_motor** and **servo_motor_rt1180** examples, the below hardware is required:

- i.MX 95-15x15-lpddr4x-evk/ i.MX 95-19x19-lpddr5-evk platform
- Inovance SV680 / **NXP XSERVO-MTR-PSG Board**
- Network Cable

There are two types of servos supported, Inovance SV680 and **NXP XSERVO-MTR-PSG Board** as EtherCAT SubDevice.

1. **Inovance SV680 as EtherCAT SubDevice**: Running SOEM Servo Motor example on i.MX 95 EVK to control Inovance SV680.

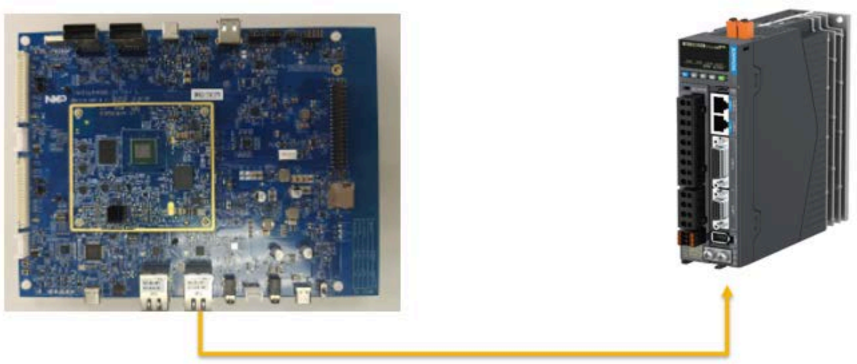


Figure 139. Running SOEM Servo Motor example on i.MX 95 EVK (Inovance SV680 as EtherCAT SubDevice)

2. **Running motion_control example on NXP XSERVO-MTR-PSG Board as EtherCAT SubDevice**. Running SOEM servo_motor_rt1180 example on i.MX 95 EVK to control motor on **NXP XSERVO-MTR-PSG Board**, which is RT1180 Multi-Motor Control Board.

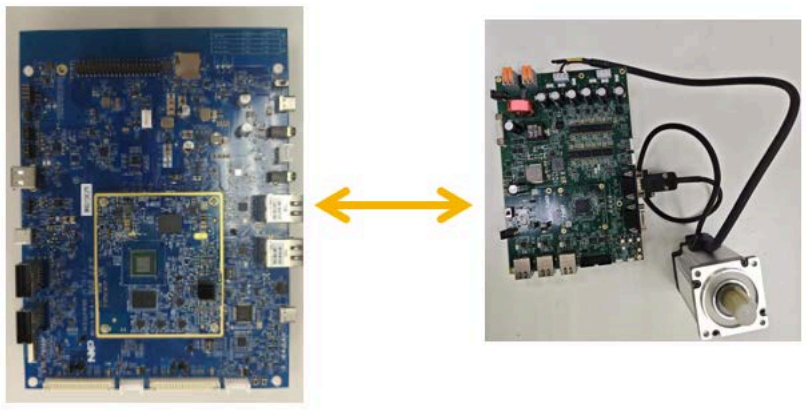


Figure 140. Running servo_motor_rt1180 example on i.MX 95 EVK (NXP XSERVO-MTR-PSG as EtherCAT SubDevice)

Note: *Before running servo_motor example, you must download the MCU-SDK package for NXP XSERVO-MTR-PSG Board from github: https://github.com/nxp-appcodehub/rd-motion-control-slave-servo-mimxrt1180*

6.1.4.5.2 Building the demo images of Cortex-A

6.1.4.5.2.1 digital_io example

For **digital_io** example, the demo images "soem_digital_io_ca55" are compiled with the i.MX 95 EVK target image compiling, and they are installed into the directory "/example" of the target rootfs. Images can also be found in the directory "<image-build-dir>/tmp/deploy/images/imx95-19x19-lpddr5-evk/examples/" on building host.

```
tree tmp/deploy/images/imx95-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-digital-io-freertos
|── soem_digital_io_ca55.elf
|── soem_digital_io_ca55.bin

tree tmp/deploy/images/imx95-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-digital-io-zephyr
|── soem_digital_io_ca55.elf
|── soem_digital_io_ca55.bin
```

Also, the demo images can be compiled using the below command on i.MX 95 EVK build directory:

```
bitbake soem-digital-io
```

6.1.4.5.2.2 servo_motor and servo_motor_rt1180 examples

For **servo_motor** and **servo_motor_rt1180** example, the demo images "`soem_servo_ca55`", “`soem_servo_rt1180_ca55`” are compiled with the i.MX 95 EVK target image compiling, and they are installed into the directory "/example" of the target rootfs. Images can also be found in the directory "`<image-build-dir>/tmp/deploy/images/imx95-19x19-lpddr5-evk/examples/`" on building host.

Note: *"soem_servo_ca55" are for “Inovance SV680” case, "soem_servo_rt1180_ca55" are for “NXP XSERVO-MTR-PSG Board” case.*

```
cd <image-build-dir>/
tree tmp/deploy/images/imx95-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-servo-freertos
|── soem_servo_ca55.elf
|── soem_servo_ca55.bin
tree tmp/deploy/images/imx95-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-servo-zephyr
|── soem_servo_ca55.elf
|── soem_servo_ca55.bin
tree tmp/deploy/images/imx95-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-servo-rt1180-freertos
|── soem_servo_rt1180_ca55.elf
|── soem_servo_rt1180_ca55.bin
tree tmp/deploy/images/imx95-19x19-lpddr5-evk/examples/heterogeneous-multicore/
soem-servo-rt1180-zephyr
|── soem_servo_rt1180_ca55.elf
|── soem_servo_rt1180_ca55.bin
```

Also, the demo images can be compiled using the below command on i.MX 95 EVK build directory:

```
bitbake soem-servo
or
bitbake soem-servo-rt1180
```

6.1.4.5.3 Running SOEM demo images by using U-Boot

This section describes the steps to write an SOEM demo image file to the 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.
   a. On Windows OS, open the Device Manager, find USB serial Port in Ports (COM and LPT). Assume that the ports are COM7 COM8 COM9 and COM10. One port is for the debug message from the Cortex-A55 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.
   b. Build and flash the `nxp-image-real-time-edge` image to a SD card and insert the SD card to the target board. Make sure to use the default boot SD slot and check the DIP boot switch configuration.
   c. Open your preferred serial terminal for the serial devices. Set the speed to 115200 bps, 8 data bits, 1 stop bit (115200, 8N1), no parity. Then, power on the board.
   d. Power on the board and hit any key to stop autoboot in the terminal window. Then enter the U-Boot Command line mode. You can then write the image and run it from TCM or DRAM with the following commands:
2. **Running FreeRTOS on Cortex-A**

   ```
   => ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-
   digital-io-freertos/soem_digital_io_ca55.bin;
   => dcache flush; icache flush;
   => cpu 5 release 0xD0000000
   ```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
freertos/soem_servo_ca55.bin;
=> dcache flush; icache flush;
=> cpu 5 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
rt1180-freertos/soem_servo_rt1180_ca55.bin;
=> dcache flush; icache flush;
=> cpu 5 release 0xD0000000
```

Running Zephyr on Cortex-A

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-
digital-io-zephyr/soem_digital_io_ca55.bin;
=> dcache flush; icache flush;
=> cpu 5 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
zephyr/soem_servo_ca55.bin;
=> dcache flush; icache flush;
=> cpu 5 release 0xD0000000
```

Or

```
=> ext4load mmc 1:2 0xD0000000 /examples/heterogeneous-multicore/soem-servo-
rt1180-zephyr/soem_servo_rt1180_ca55.bin;
=> dcache flush; icache flush;
=> cpu 5 release 0xD0000000
```

6.1.5 CODESYS EtherCAT MainDevice

6.1.5.1 Overview

Based on Real-time Edge yocto project, a new yocto distro named "`nxp-real-time-edge-plc`" is added, which is specific to the PLC use case. This section describes how to build a Real-time Edge PLC image and how to setup a CODESYS test project as an example to drive motors.

- For more information about i.MX Yocto project, refer to: i.MX Yocto Project User's Guide (https://www.nxp.com/docs/en/user-guide/UG10164.pdf)
- For additional information, refer to *Real Time Edge Yocto User Guide (RTEDGEYOCTOUG)* available on the URL: REALTIME EDGE Documentation.

6.1.5.2 Features

Features are as follows:

- **Optimized native driver**

Typical industrial software, for example CODESYS or SOEM EtherCAT MainDevice stack, work on user space and communicate using Linux standard network interface. To reduce the latency when Ethernet raw packets pass from user space through Linux standard network, the network driver is optimized to avoid memory reallocation and copying, task rescheduling. The latency of the critical path is reduced for packet transmitting and receiving.

- **Light root filesystem**

Lighter root filesystem saves CPU cycles that could be used by the PLC user program.

- **Supported platforms**
  - imx6ull14x14evk
  - imx8mm-lpddr4-evk
  - imx8mp-lpddr4-evk
  - imx93evk
  - imx95-15x15-lpddr4x-evk
  - imx95-19x19-lpddr5-evk
  - imx943-19x19-lpddr4-evk
  - imx943-19x19-lpddr5-evk

6.1.5.3 Building the image

This section provides detailed information along with the procedure for building an image.

6.1.5.3.1 Build configurations

A new yocto distro named "nxp-real-time-edge-plc" is added for the PLC use case, and below platforms are supported:

- imx6ull14x14evk
- imx8mm-lpddr4-evk
- imx8mp-lpddr4-evk
- imx93evk
- imx95-15x15-lpddr4x-evk
- imx95-19x19-lpddr5-evk
- imx943-19x19-lpddr4-evk
- imx943-19x19-lpddr5-evk

Real-time Edge provides the script real-time-edge-setup-env.sh to simply the setup for both i.MX and Layerscape boards. To use the script, the name of the specific machine to be built for and the desired distro must be specified. The script sets up a directory and the configuration files for the specified machine and distro.

The syntax for the `real-time-edge-setup-env.sh` script is shown below:

```
$ DISTRO=nxp-real-time-edge-plc MACHINE=<machine name> source real-time-edge-
setup-env.sh -b <build dir>
```

where:

- **MACHINE=<machine name>** is the name of the supported platforms.
- **-b <build dir>** specifies the name of the build directory created by the `real-time-edge-setup-env.sh` script.

After the script runs, the working directory is the one just created by the script, specified with the `-b` option. A `conf` folder is created containing the files `bblayers.conf` and `local.conf`.

The `local.conf` file contains the machine and distro specifications. An example is shown below:

```
MACHINE ??= 'imx8mp-lpddr4-evk'
DISTRO ?= 'nxp-real-time-edge-plc'
ACCEPT_FSL_EULA = "1"
```

The MACHINE configuration can be changed by editing this file, if necessary.

6.1.5.3.2 Build scenarios

The following are build setup scenarios for various configurations. Set up the manifest and populate the Yocto Project layer sources using the commands below:

```
$ mkdir yocto-real-time-edge
$ cd yocto-real-time-edge
$ repo init -u https://github.com/nxp-real-time-edge-sw/yoctoreal-time-edge.git/
yocto-real-time-edge.git \
-b real-time-edge-walnascar \
-m real-time-edge-3.3.0.xml
$ repo sync
```

The following sections give some specific examples.

- **Real-time Edge PLC image on i.MX 6ULL EVK**

  ```
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx6ull14x14evk source real-time-edge-
  setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

- **Real-time Edge PLC image on i.MX 8M Mini EVK**

  ```
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx8mm-lpddr4-evk source real-time-
  edge-setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

- **Real-time Edge PLC image on i.MX 8M Plus EVK**

  ```
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx8mp-lpddr4-evk source real-time-
  edge-setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

- **Real-time Edge PLC image on i.MX 93 EVK**

  ```
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx93evk source real-time-edge-setup-
  env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

- **Real-time Edge PLC image on i.MX 95 EVK**

  ```
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx95-15x15-lpddr4x-evk source real-
  time-edge-setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

- **Real-time Edge PLC image on i.MX 943 EVK**

  ```
  $ DISTRO=nxp-real-time-edge-plc MACHINE=imx943-19x19-lpddr4-evk source real-
  time-edge-setup-env.sh -b build-imx-real-time-edge-plc
  $ bitbake nxp-image-real-time-edge-plc
  ```

6.1.5.3.3 Image deployment

All filesystem images are deployed to the `<build directory>/tmp/deploy/images` directory. Each image building creates a U-Boot image, a kernel image, and an image type based on the `IMAGE_FSTYPES` variable defined in the machine configuration file. Most machine configurations provide an SD card image (.wic) and a rootfs image (.tar). The SD card image contains a partitioned image (with U-Boot, kernel, rootfs, and other such files) suitable for booting the corresponding hardware.

Copy image to SD card:

The SD card image file, (`<image_name>.wic`) contains a partitioned image (with U-Boot, kernel, rootfs, and other files) suitable for booting the corresponding hardware. Run the following commands to copy the image to an SD card:

```
$ zstd -d <image_name>.wic.zst
$ sudo dd if=<image_name>.wic of=/dev/sd<disk> bs=1M conv=fsync
```

6.1.5.4 Running CODESYS on the boards

6.1.5.4.1 Running CODESYS on i.MX 6ULL

Execute the following command to update the dtb file and boot into kernel.

```
uboot ==> setenv fdt_file imx6ull-14x14-evk-ecat.dtb
uboot ==> setenv tee no
uboot ==> run bootcmd
```

Set the CPU frequency governor to "*performance*" for **CPU** core to use the highest frequency.

```
# echo performance > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

Enable the EtherCAT port.

```
# ethercat_port=`ls /sys/bus/platform/devices/2188000.ethernet/net` && echo
 $ethercat_port
# ifconfig $ethercat_port up
```

Assign IP address for the Ethernet port automatically if there is a DHCP server in the network.

```
# ethernet_port=`ls /sys/bus/platform/devices/20b4000.ethernet/net` && echo
 $ethernet_port
# udhcpc -i $ethernet_port
```

Else assign IP address manually.

```
# ethernet_port=`ls /sys/bus/platform/devices/20b4000.ethernet/net` && echo
 $ethernet_port
# ifconfig $ethernet_port 192.168.1.xx
```

Use **scp** command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

Note:

- *Please refer to the next section to obtain `codesyscontrol.bin`.*
- *CODESYS logs can be viewed in the `CODESYS.log` file.*

6.1.5.4.2 Running CODESYS on i.MX 8M Mini and i.MX 8M Plus

Execute the following command to isolate CPU1 (by default, the “EtherCAT_Task” thread created by the Realtime of CODESYS is bound into CPU1).

```
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=
${console}  root=${mmcroot} isolcpus=1'
```

Execute the following command to update the dtb file and boot into kernel.

```
// on i.MX 8M Mini
uboot ==> setenv fdtfile imx8mm-evk-ecat.dtb
// on i.MX 8M Plus
uboot ==> setenv fdtfile imx8mp-evk-ecat.dtb

uboot ==> run bootcmd
```

Set the CPU frequency governor to “*performance*” for **CPU1** core to use the highest frequency.

```
echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

Stop the “**udhcpc**” service which interface is connected to the EtherCAT servos.

```
# ethercat_net=`ls /sys/bus/platform/devices/30be0000.ethernet/net` && echo
 $ethercat_net
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i
 $ethercat_net”
# kill -9 PID
```

Enable the EtherCAT port.

```
# ethercat_port=`ls /sys/bus/platform/devices/30be0000.ethernet/net` && echo
 $ethercat_port
# ifconfig $ethercat_port up
```

Assign IP address for the Ethernet port automatically if there is a DHCP server in the network.

```
// assume the name of the USB network port is eth1
# udhcpc -i eth1
```

Else assign IP address manually.

```
// assume the name of the USB network port is eth1
# ifconfig eth1 192.168.1.xx
```

Use **scp** command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

Note:

- *Please refer to the next section to obtain `codesyscontrol.bin`.*
- *CODESYS logs can be viewed in the `CODESYS.log` file.*

6.1.5.4.3 Running CODESYS on i.MX 93

Execute the following command to isolate CPU1 (by default, the “EtherCAT_Task” thread created by the Realtime of CODESYS is bound into CPU1).

```
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=
${console}  root=${mmcroot} isolcpus=1'
```

Execute the following command to update the dtb file and boot into kernel.

```
uboot ==> setenv fdtfile imx93-11x11-evk-ecat.dtb
uboot ==> run bootcmd
```

Stop the “**udhcpc**” service which interface is connected to the EtherCAT servos.

```
# ethercat_net=`ls /sys/bus/platform/devices/42890000.ethernet/net` && echo
 $ethercat_net
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i
 $ethercat_net”
# kill -9 PID
```

Enable the EtherCAT port.

```
# ethercat_port=`ls /sys/bus/platform/devices/42890000.ethernet/net` && echo
 $ethercat_port
# ifconfig $ethercat_port up
```

Assign IP address for the Ethernet port automatcally if there is a DHCP server in the network.

```
# ethernet_port=`ls /sys/bus/platform/devices/428a0000.ethernet/net` && echo
 $ethernet_port
# udhcpc -i $ethernet_port
```

Else assign IP address manually.

```
# ethernet_port=`ls /sys/bus/platform/devices/428a0000.ethernet/net` && echo
 $ethernet_port
# ifconfig $ethernet_port 192.168.1.xx
```

Execute the following command to disable the "CPU1" idle state1.

```
# echo "1" > /sys/devices/system/cpu/cpu1/cpuidle/state1/disable
```

Execute the following command to disable the DDR auto clock gating.

```
# echo 0 > /sys/devices/platform/imx93-lpm/auto_clk_gating
```

Use **scp** command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

Note:

- *Please refer to the next section to obtain `codesyscontrol.bin`.*
- *CODESYS logs can be viewed in the `CODESYS.log` file.*

6.1.5.4.4 Running CODESYS on i.MX 95

Execute the following command to isolate CPU1 (by default, the “EtherCAT_Task” thread created by the Realtime of CODESYS is bound into CPU1).

```
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=
${console}  root=${mmcroot} isolcpus=1'
uboot ==> run bootcmd
```

Stop the “**udhcpc**” service which interface is connected to the EtherCAT servos.

```
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i eth0”
# kill -9 PID
```

Enable the EtherCAT port.

```
# imx95-15x15-lpddr4x-evk(J8):
# echo -n 0001:00:00.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
# echo -n fsl_ecat_enetc4 > /sys/bus/pci/devices/0001:00:00.0/driver_override
# echo -n 0001:00:00.0 > /sys/bus/pci/drivers/fsl_ecat_enetc4/bind
# ifconfig eth0 up
# imx95-19x19-lpddr5-evk(J11):
# echo -n 0002:00:00.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
# echo -n fsl_ecat_enetc4 > /sys/bus/pci/devices/0002:00:00.0/driver_override
# echo -n 0002:00:00.0 > /sys/bus/pci/drivers/fsl_ecat_enetc4/bind
# ifconfig eth0 up
```

Else assign IP address manually(J9).

```
# ifconfig eth1 192.168.1.xx
```

Set the CPU frequency governor to “*performance*” for **CPU1** core to use the highest frequency.

```
# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

Use **scp** command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

Note:

- *Please refer to the next section to obtain `codesyscontrol.bin`.*
- *CODESYS logs can be viewed in the `CODESYS.log` file.*

6.1.5.4.5 Running CODESYS on i.MX 943

Execute the following command to isolate CPU1 (by default, the “EtherCAT_Task” thread created by the Realtime of CODESYS is bound into CPU1).

```
uboot ==> setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=
${console}  root=${mmcroot} isolcpus=1'
uboot ==> run bootcmd
```

Stop the “**udhcpc**” service which interface is connected to the EtherCAT servos.

```
# ps
// find “PID root 3468 S udhcpc -R -b -p /var/run/udhcpc.eth0.pid -i eth0”
# kill -9 PID
```

Enable the EtherCAT port(J26).

```
# echo -n 0001:01:08.0 > /sys/bus/pci/drivers/fsl_enetc4/unbind
# echo -n fsl_ecat_enetc4 > /sys/bus/pci/devices/0001:01:08.0/driver_override
# echo -n 0001:01:08.0 > /sys/bus/pci/drivers/fsl_ecat_enetc4/bind
# ifconfig eth1 up
```

Else assign IP address manually(J27).

```
# ifconfig eth2 192.168.1.xx
```

Set the CPU frequency governor to “*performance*” for **CPU1** core to use the highest frequency.

```
# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
```

Use **scp** command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

Note:

- *Please refer to the next section to obtain `codesyscontrol.bin`.*
- *CODESYS logs can be viewed in the `CODESYS.log` file.*

6.1.5.5 Setting up CODESYS

CODESYS is a powerful commercial PLC software programming tool. It supports IEC61131-3 standard IL, ST, FBD, LD, CFC, SFC, and six kinds of PLC programming languages. Users can choose different language editing subroutines in the same project, function module, and so on.

6.1.5.5.1 Board environment setup

i.MX 6ULL EVK Boards

- **Hardware requirements**
  - Two network cables
  - Mini/micro USB cable
  - One servo motor(DELTA ASDA-B3)
  - Personal Computer on which the CODESYS has been installed
- **Preparing the example**
  - Connect the PC and the network port labeled **J1501 B** on the board.
  - Connect the servo motors to the network port labeled **J1501 A** on the board.
  - Connect a USB cable between the host PC and the OpenSDA USB port on the target board. Then, open a serial terminal with the following settings
    - 115200 baud rate
    - 8 data bits
    - No parity

  - One stop bit
  - No flow control

i.MX 8M Mini EVK Boards

- **Hardware requirements**
  - Two Network cables
  - Mini/micro USB cable
  - Network port converter with type-C interface
    Note: *i.MX 8M Mini has only one network port, so it is needs another network port connected with CODESYS IDE.*
  - One servo motor (DELTA ASDA-B3)
  - Personal Computer on which the CODESYS has been installed.
- **Preparing the example**
  - Connect the servo and the network port of the board.
  - Plug the Network port converter into the USB port (labeled PORT1) of the board, and then connect the Network port converter to the PC using a network cable.
  - Connect a USB cable between the host PC and the OpenSDA USB port on the target board, open a serial terminal with the following settings:
    - 115200 baud rate
    - 8 data bits
    - No parity
    - One stop bit
    - No flow control

The connection diagram for i.MX 8M Mini board is as follows.

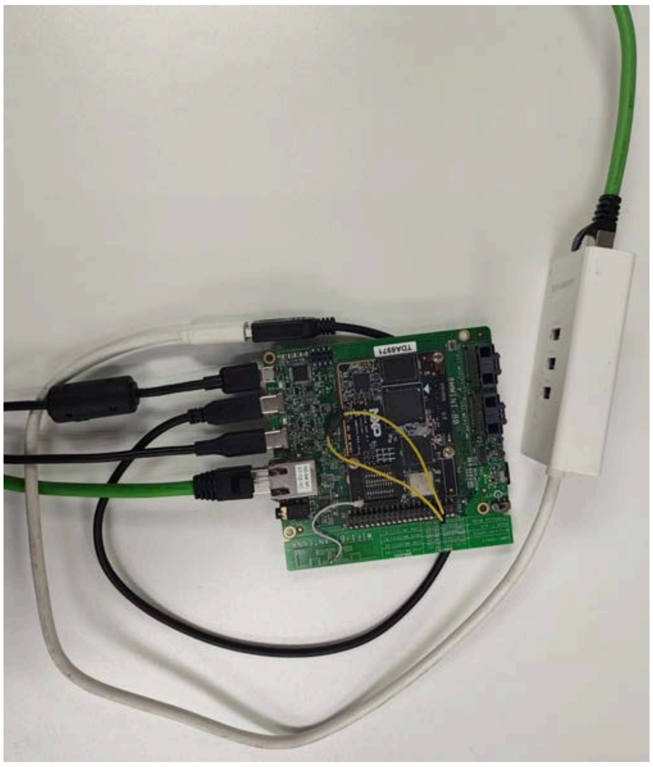


Figure 141. i.MX 8M Mini board connection

i.MX 8M Plus EVK Boards

- **Hardware requirements**
  - Two Network cables
  - Mini/micro USB cable
  - One servo motor (DELTA ASDAB3)
  - Personal Computer on which the CODESYS has been installed.
- **Preparing the example**
  - Connect the PC to the network port labeled **ENET2** on the board.
  - Connect the servo motors to the network port labeled **ENET1** on the board.
  - Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
    - 115200 baud rate
    - 8 data bits
    - No parity
    - One stop bit
    - No flow control

i.MX 93 EVK Boards

- **Hardware requirements**
  - Two Network cables
  - Mini/micro USB cable
  - One servo motor(DELTA ASDAB3)
  - Personal Computer on which the CODESYS has been installed
- **Preparing the example**
  - Connect the PC to the network port labeled **ENET1** on the board.
  - Connect the servo motors to the network port labeled **ENET2** on the board.
  - Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
    - 115200 baud rate
    - 8 data bits
    - No parity
    - One stop bit
    - No flow control

i.MX 95 EVK Boards

- **Hardware requirements**
  - Two Network cables
  - Mini/micro USB cable
  - One servo motor(DELTA ASDAB3)
  - Personal Computer on which the CODESYS has been installed
- **Preparing the example**
  - Connect the PC to the network port labeled **ENET2(J9)** on the board.
  - Connect the servo motors to the network port labeled **ENET1(J8)** on the board.
  - Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
    - 115200 baud rate
    - 8 data bits

  - No parity
  - One stop bit
  - No flow control

i.MX 943 EVK Boards

- **Hardware requirements**
  - Two Network cables
  - Mini/micro USB cable
  - One servo motor(DELTA ASDAB3)
  - Personal Computer on which the CODESYS has been installed
- **Preparing the example**
  - Connect the PC to the network port labeled **ENET2(J27)** on the board.
  - Connect the servo motors to the network port labeled **ENET1(J26)** on the board.
  - Connect a USB cable between the host PC and the OpenSDA USB port on the target board, and open a serial terminal with the following settings:
    - 115200 baud rate
    - 8 data bits
    - No parity
    - One stop bit
    - No flow control

6.1.5.5.2 CODESYS project setup

6.1.5.5.2.1 Downloading CODESYS software and runtime binary

Download and install the "CODESYS Development System V3 3.5.20.20 64 bit (.exe)" from CODESYS offical website to your windows.

Download "CODESYS Control for Linux ARM SL 4.11.0.0" runtime package from CODESYS offical website and decompress it using the commands below:

```
# unzip 'CODESYS Control for Linux ARM64 SL 4.11.0.0.package'
# cd Delivery/linuxarm64/
# dpkg-deb -R codesyscontrol_linuxarm64_4.11.0.0_arm64.deb software
# cd software/opt/codesys/bin/
```

The runtime binary `codesyscontrol.bin` is located in the `software/opt/codesys/bin/` directory.

In Windows environment, double-click 'CODESYS Control for Linux ARM64 SL 4.11.0.0.package' to install it onto your CODESYS.

Note:

- **Download 32-bit version for the platform:**

- i.MX 6ULL EVK

- **Download 64-bit version for the platforms:**

- i.MX 8M Mini EVK

- i.MX 8M Plus EVK

- i.MX 93 EVK

- i.MX 95 EVK

- i.MX 943 EVK

6.1.5.5.2.2 Starting CODESYS runtime

Use the **scp** command to copy **codesyscontrol.bin** into the board and start it:

```
# ./codesyscontrol.bin > CODESYS.log &
```

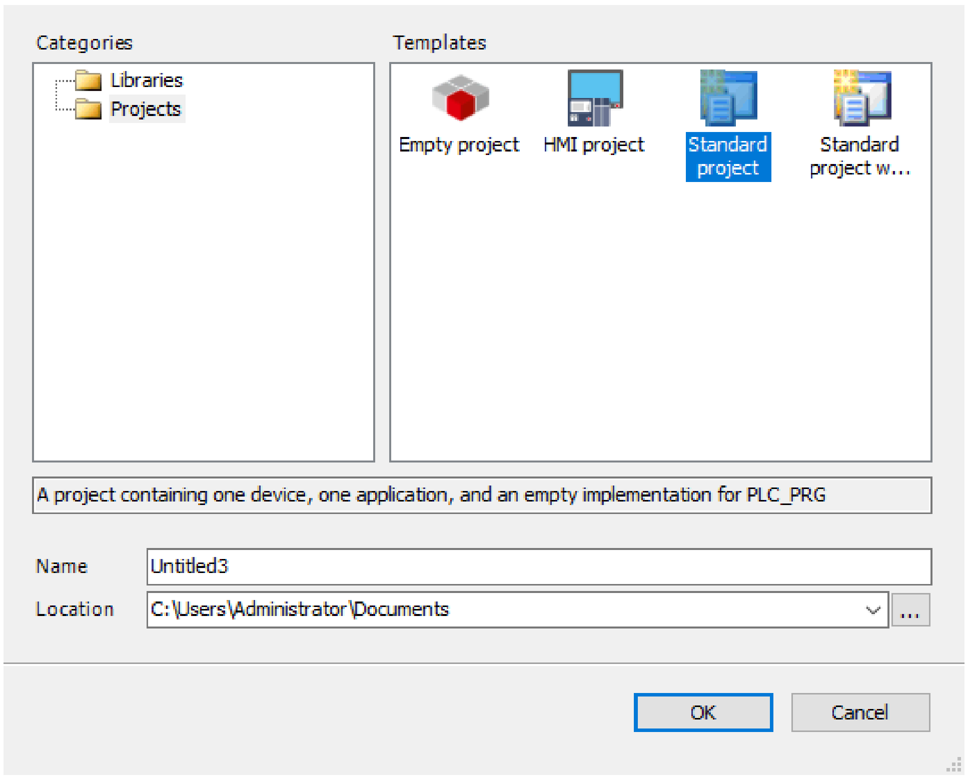
CODESYS logs can be viewed in the CODESYS.log file.

6.1.5.5.2.3 Creating a new CODESYS project

This section describes the steps to create the CODESYS project in the machine on which the CODESYS tool has been installed.

- Open **CODESYS** and build a new project using the steps below:

1. Click “**File**” -> “**New project**”.
2. Select “**Standard project**” as the template.
3. Rename this project.
4. Click “**OK**” to complete it.



A project containing one device, one application, and an empty implementation for PLC_PRG

Name: Untitled3

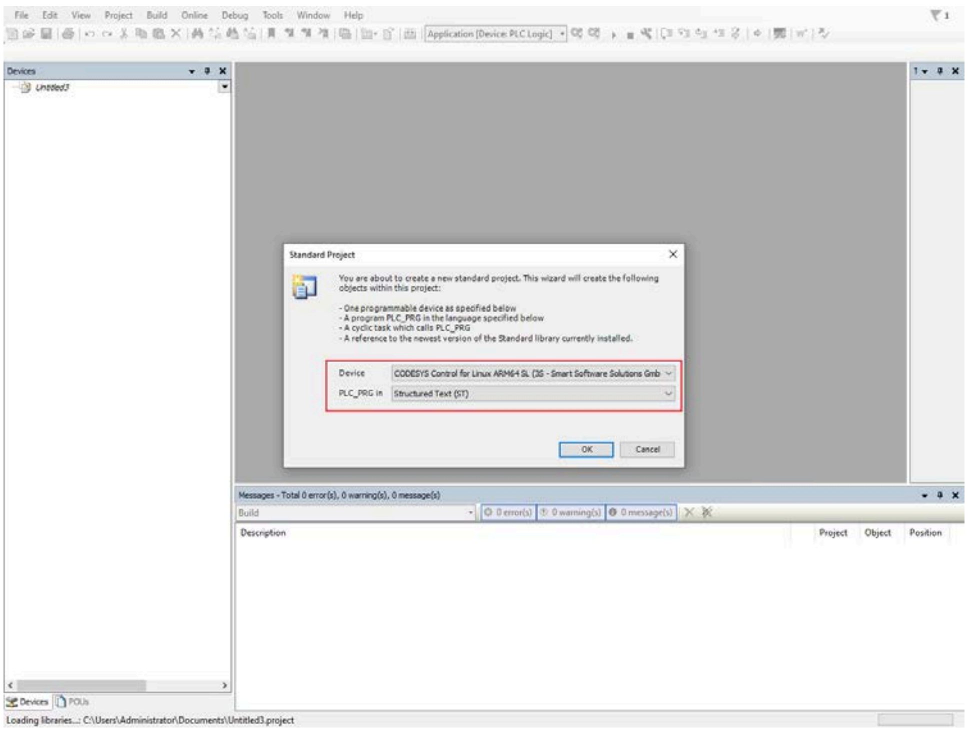
Location: C:\Users\Administrator\Documents

Figure 142. Select Standard project as template

- **Set Device and PLC_PRG**

1. Select “**CODESYS Control for Linux ARM64 SL**” or “**CODESYS Control for Linux ARM SL**” as the Device.
2. Select “**Structured Text(ST)**” as the PLC_PRG.

3. Click “**OK**” to complete it.



Standard Project

You are about to create a new standard project. This wizard will create the following objects within this project:

- One programmable device as specified below
- A program PLC_PRG in the language specified below
- A cyclic task which calls PLC_PRG
- A reference to the newest version of the Standard library currently installed.

Device: CODESYS Control for Linux ARM64 SL (3S - Smart Software Solutions Gmb

PLC_PRG in: Structured Text (ST)

Messages - Total 0 error(s), 0 warning(s), 0 message(s)

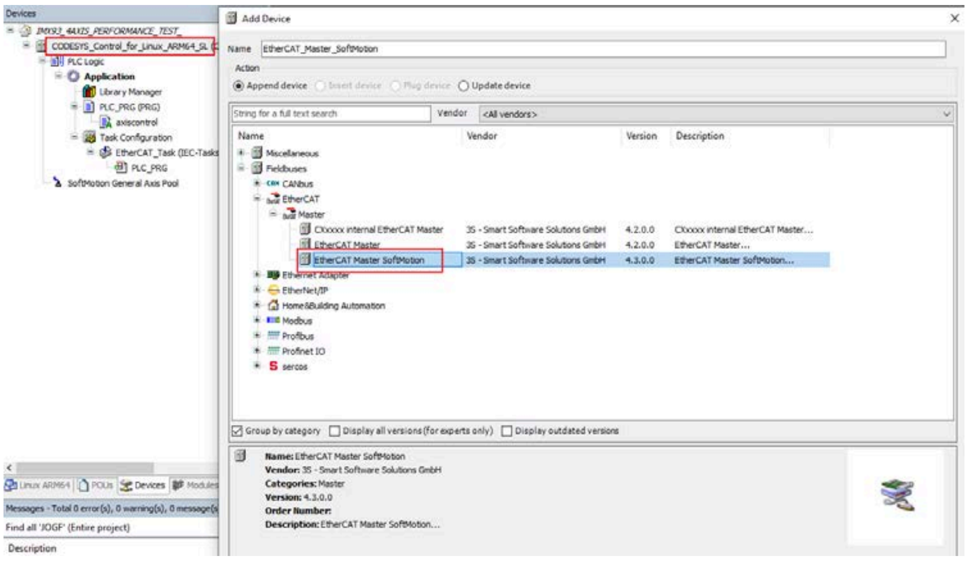
Loading libraries...: C:\Users\Administrator\Documents\Untitled3.project

Figure 143. Selecting the device and PLC_PRG

6.1.5.5.2.4 Adding EtherCAT MainDevice and SubDevice

Use the steps below to add the EtherCAT MainDevice and SubDevice.

1. Set the EtherCAT MainDevice:
    a. Right-click on the device -> “**Add Device**”
    b. Select “**EtherCAT Master SoftMotion**”
    c. Click “**Add Device**” to complete it.



Add Device

Name: EtherCAT_Master_SoftMotion

Action: [✓ Append device] Insert device Plug device Update device

Vendor: <All vendors>

Name	Vendor	Version	Description
CXxxxx internal EtherCAT Master	3S - Smart Software Solutions GmbH	4.2.0.0	CXxxxx internal EtherCAT Master...
EtherCAT Master	3S - Smart Software Solutions GmbH	4.2.0.0	EtherCAT Master...
EtherCAT Master SoftMotion	3S - Smart Software Solutions GmbH	4.3.0.0	EtherCAT Master SoftMotion...

[✓ Group by category] Display all versions (for experts only) Display outdated versions

Name: EtherCAT Master SoftMotion
Vendor: 3S - Smart Software Solutions GmbH
Categories: Master
Version: 4.3.0.0
Order Number:
Description: EtherCAT Master SoftMotion...

Messages - Total 0 error(s), 0 warning(s), 0 message(s)

Figure 144. Selecting EtherCAT MainDevice

2. **Download DELTA ASDA-B3 device description file**https://downloadcenter.deltaww.com/en-US/DownloadCenter
3. **Add DELTA device description file**
   a. Click “**Tools**” -> “**Device Repository**”.
   b. Click “**Install**”.
   c. Select “**Delta ASDA-x3-E rev0.04_10EMC.XML**” file.
   d. Click “**Close**” to complete it.

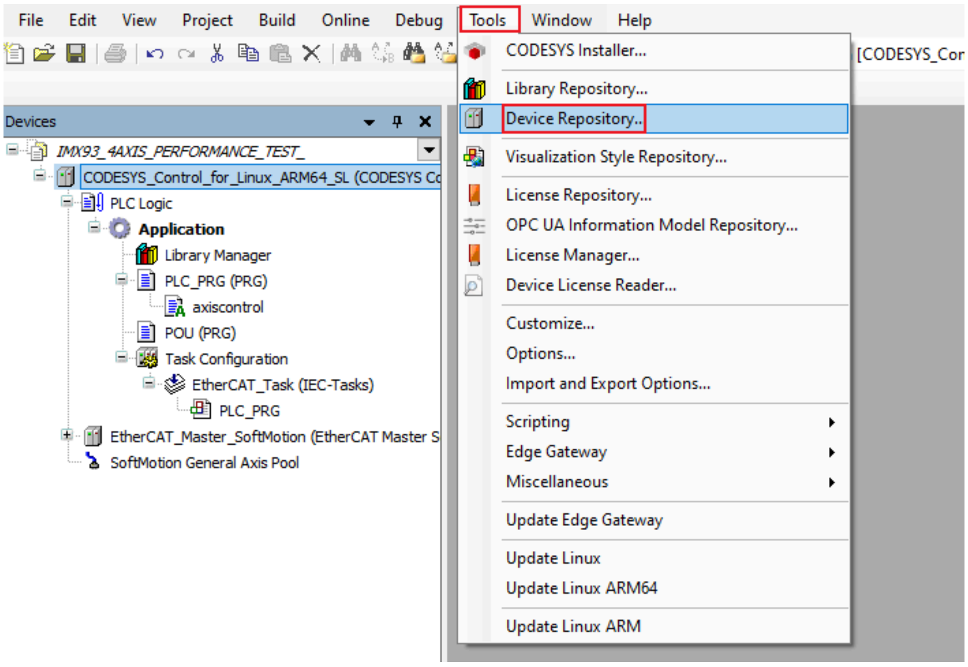
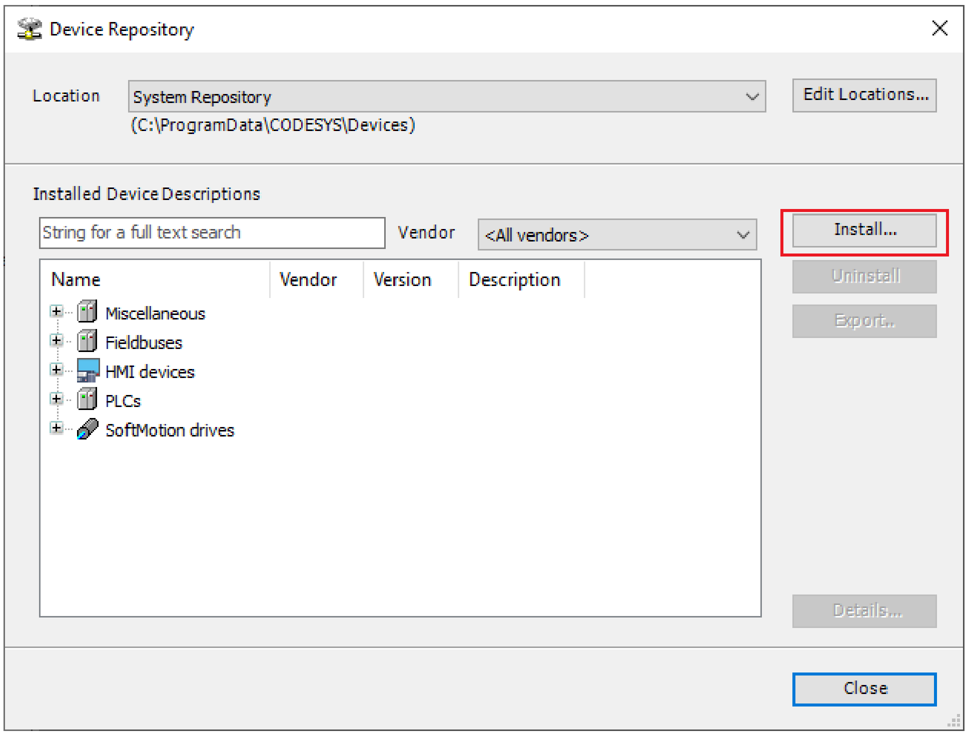


Figure 145. Adding DELTA device description file



Location: System Repository (C:\ProgramData\CODESYS\Devices)

Figure 146. View Installed devices

4. **Add servo device**
   a. Right-click on the “**EtherCAT_Master_SoftMotion**” -> “**Add Device**”.
   b. Select the corresponding servo.
   c. Click “**Add Device**” to complete it.

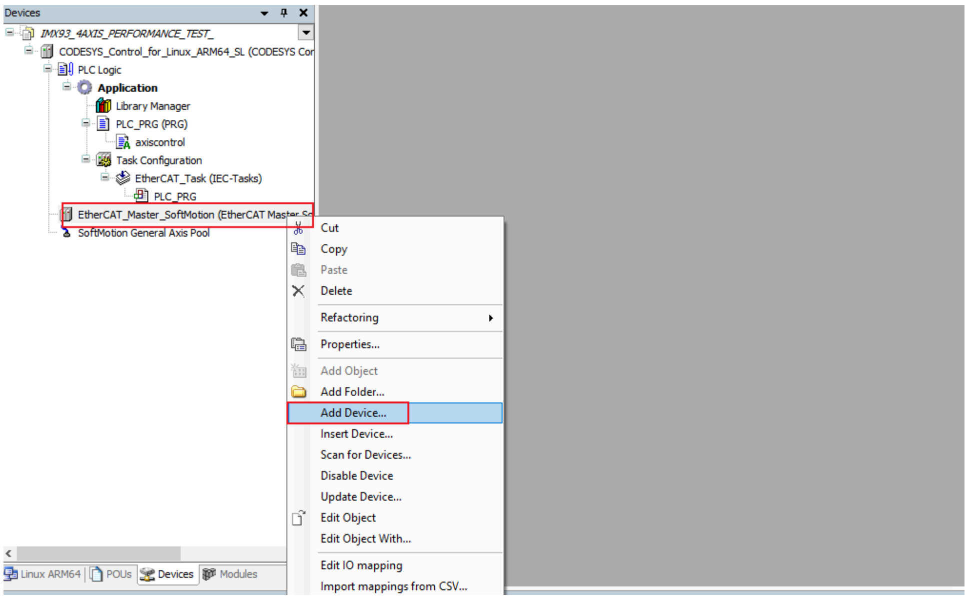
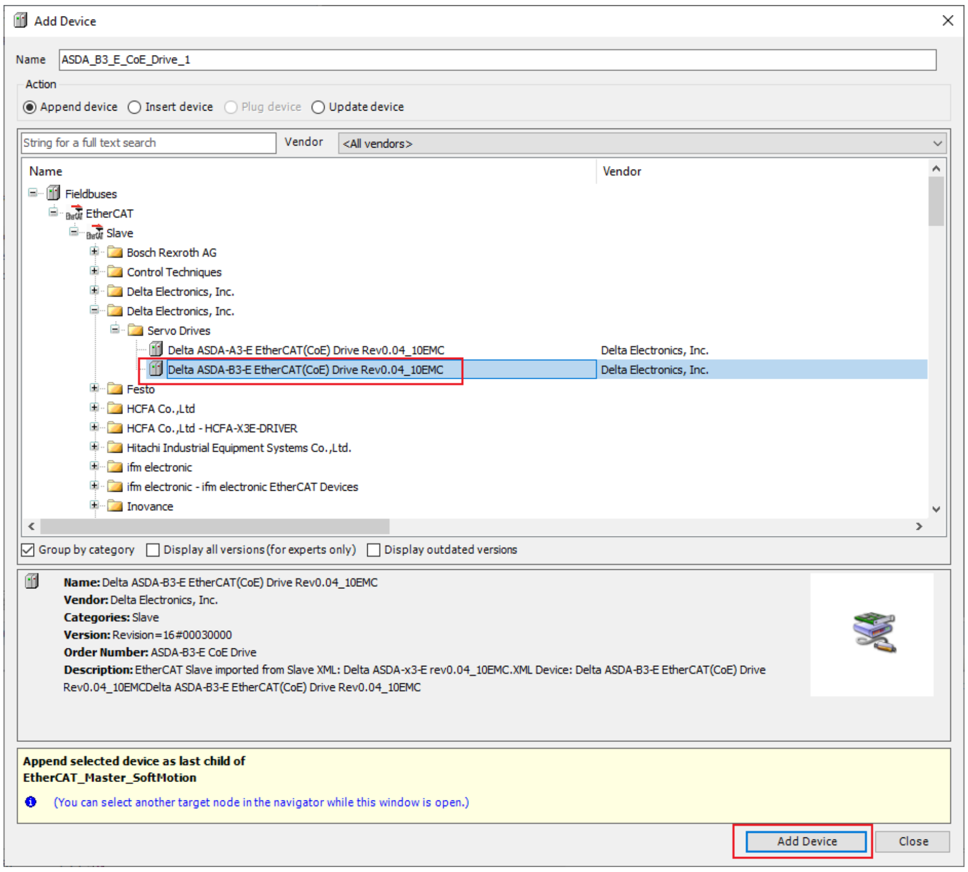


Figure 147. Adding a Device



Add Device

Name: ASDA_B3_E_CoE_Drive_1

Action: (●) Append device ( ) Insert device ( ) Plug device ( ) Update device

String for a full text search | Vendor: <All vendors>

Name	Vendor
Fieldbuses	
EtherCAT	
Slave	
Bosch Rexroth AG	
Control Techniques	
Delta Electronics, Inc.	
Delta Electronics, Inc.	
Servo Drives	
Delta ASDA-A3-E EtherCAT(CoE) Drive Rev0.04_10EMC	Delta Electronics, Inc.
Delta ASDA-B3-E EtherCAT(CoE) Drive Rev0.04_10EMC	Delta Electronics, Inc.
Festo	
HCFA Co.,Ltd	
HCFA Co.,Ltd - HCFA-X3E-DRIVER	
Hitachi Industrial Equipment Systems Co.,Ltd.	
ifm electronic	
ifm electronic - ifm electronic EtherCAT Devices	
Inovance	

[✓] Group by category [ ] Display all versions (for experts only) [ ] Display outdated versions

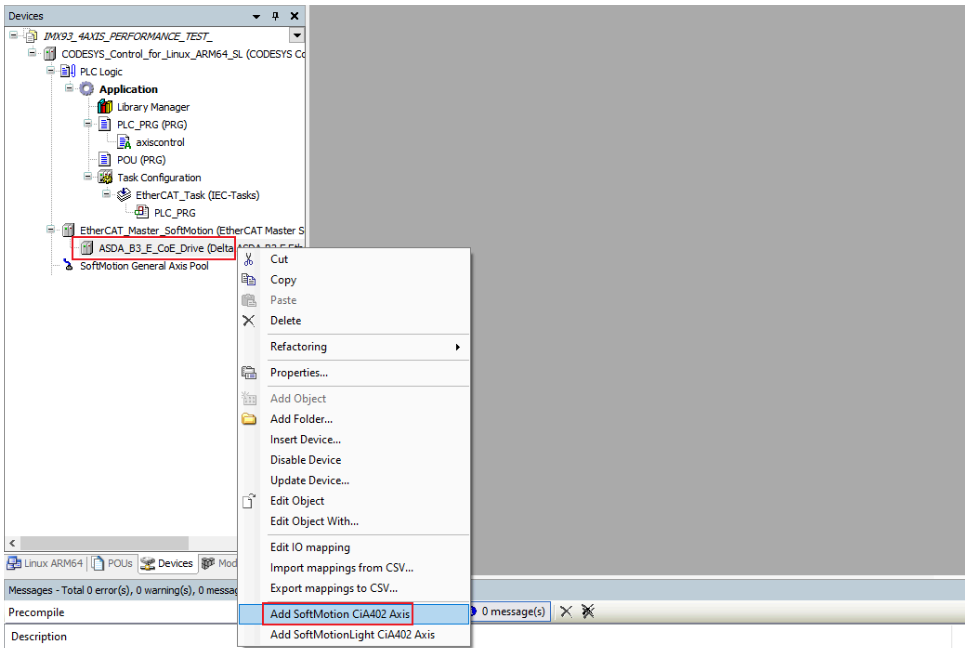
Name: Delta ASDA-B3-E EtherCAT(CoE) Drive Rev0.04_10EMC
Vendor: Delta Electronics, Inc.
Categories: Slave
Version: Revision=16#00030000
Order Number: ASDA-B3-E CoE Drive
Description: EtherCAT Slave imported from Slave XML: Delta ASDA-x3-E rev0.04_10EMC.XML Device: Delta ASDA-B3-E EtherCAT(CoE) Drive Rev0.04_10EMCDelta ASDA-B3-E EtherCAT(CoE) Drive Rev0.04_10EMC

Append selected device as last child of
EtherCAT_Master_SoftMotion

(You can select another target node in the navigator while this window is open.)

Figure 148. Select the device to be added

5. **Add CiA402 Axis device**
   a. Right-click on the servo -> “**Add SoftMotion CiA402 Axis**“.



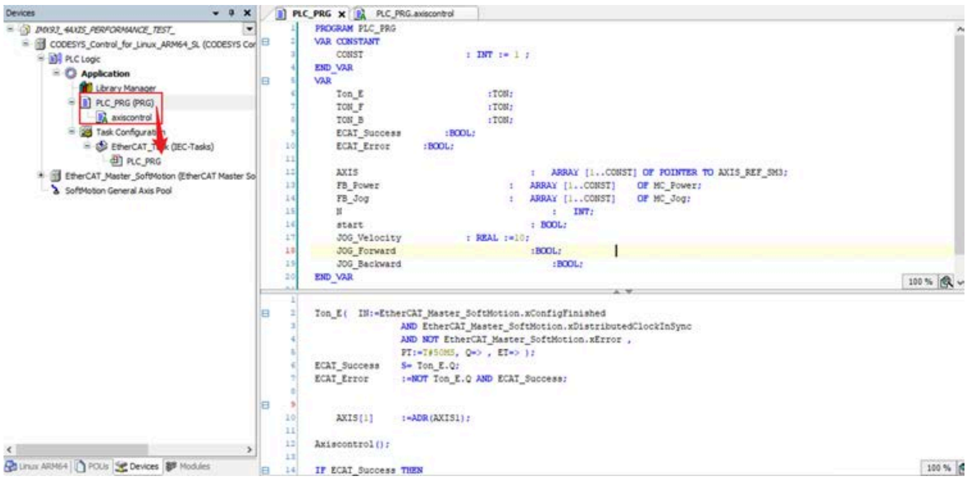
Devices
IMX93_4AXIS_PERFORMANCE_TEST_
CODESYS_Control_for_Linux_ARM64_SL (CODESYS C
PLC Logic
Application
Library Manager
PLC_PRG (PRG)
axiscontrol
POU (PRG)
Task Configuration
EtherCAT_Task (IEC-Tasks)
PLC_PRG
EtherCAT_Master_SoftMotion (EtherCAT Master S
ASDA_B3_E_CoE_Drive (Delta
SoftMotion General Axis Pool
Cut
Copy
Paste
Delete
Refactoring
Properties...
Add Object
Add Folder...
Insert Device...
Disable Device
Update Device...
Edit Object
Edit Object With...
Edit IO mapping
Import mappings from CSV...
Export mappings to CSV...
Add SoftMotion CiA402 Axis
Add SoftMotionLight CiA402 Axis
Linux ARM64
POUs
Devices
Messages - Total 0 error(s), 0 warning(s), 0 messag
Precompile
0 message(s)
Description

Figure 149. Adding CiA402 Axis device

6.1.5.5.2.5 Adding PLC program

The below steps describe how to add the PLC program to the drive motor:

1. Right-click on the “**Application**” -> “**Add Object**” -> “**POU**” and name it “**PLC_PRG**”. (If it already exists, do not add it.) -> Pull the **PLC_PRG** under the **EtherCAT_Task**
2. Right-click on the “**PLC_PRG**” -> “**Add Object**” -> “**action**” and name it “**axiscontrol**”
3. Fill in the following PLC code:


Devices
IMX93_4AXIS_PERFORMANCE_TEST_
CODESYS_Control_for_Linux_ARM64_SL (CODESYS Co
PLC Logic
Application
Library Manager
PLC_PRG (PRG)
axiscontrol
Task Configuration
EtherCAT_Task (IEC-Tasks)
PLC_PRG
EtherCAT_Master_SoftMotion (EtherCAT Master So
SoftMotion General Axis Pool
Linux ARM64
POUs
Devices
Modules
PLC_PRG
PLC_PRG.axiscontrol
100 %

```
PROGRAM PLC_PRG
VAR CONSTANT
    CONST                 : INT := 1 ;
END_VAR
VAR
    Ton_E                    :TON;
    TON_F                    :TON;
    TON_B                    :TON;
    ECAT_Success          :BOOL;
    ECAT_Error        :BOOL;

    AXIS                              :   ARRAY [1..CONST] OF POINTER TO AXIS_REF_SM3;
    FB_Power                      :   ARRAY [1..CONST]    OF MC_Power;
    FB_Jog                        :   ARRAY [1..CONST]    OF MC_Jog;
    N                                   :   INT;
    start                             : BOOL;
    JOG_Velocity          : REAL :=10;
    JOG_Forward                       :BOOL;
    JOG_Backward                         :BOOL;
END_VAR
```

```

Ton_E(  IN:=EtherCAT_Master_SoftMotion.xConfigFinished
                AND EtherCAT_Master_SoftMotion.xDistributedClockInSync
                AND NOT EtherCAT_Master_SoftMotion.xError ,
                PT:=T#50MS, Q=> , ET=> );
ECAT_Success    S= Ton_E.Q;
ECAT_Error      :=NOT Ton_E.Q AND ECAT_Success;


    AXIS[1]     :=ADR(AXIS1);

Axiscontrol();

IF ECAT_Success THEN
```

Figure 150. Filling the PLC code

The PLC_PRG (PRG) variable is as follows:

```
PROGRAM PLC_PRG
VAR CONSTANT
CONST   : INT := 1;  // Modify based on the number of text axis
END_VAR
VAR
 Ton_E  :TON;
 TON_F  :TON;
 TON_B  :TON;
 ECAT_Success :BOOL;
 ECAT_Error   :BOOL;
 AXIS  : ARRAY [1..CONST] OF POINTER TO AXIS_REF_SM3;
 FB_Power     : ARRAY [1..CONST] OF MC_Power;
 FB_Jog       : ARRAY [1..CONST] OF MC_Jog;
 N      : INT;
 start        : BOOL;
 JOG_Velocity : REAL :=10;
 JOG_Forward  :BOOL;
 JOG_Backward :BOOL;
END_VAR
```

The PLC_PRG (PRG) code is as follows:

```

// MainDevice station operation judgment
Ton_E ( IN:=EtherCAT_Master_SoftMotion.xConfigFinished
      AND EtherCAT_Master_SoftMotion.xDistributedClockInSync
      AND NOT EtherCAT_Master_SoftMotion.xError ,
      PT:=T#50MS, Q=> , ET=> );
```

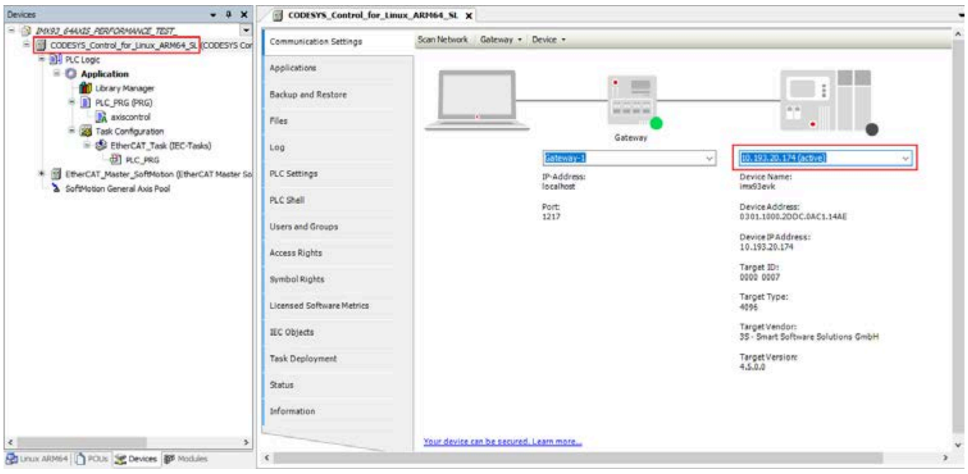
```
ECAT_Success     S= Ton_E.Q;
ECAT_Error  :=NOT Ton_E.Q AND ECAT_Success;

// Axis pointer acquisition
// Add based on the number of text axis

AXIS[1]   :=ADR(SM_Drive_GenericDSP402);
// AXIS[2]   :=ADR(SM_Drive_GenericDSP402_1);

Axiscontrol();

IF ECAT_Success THEN
  start  :=TRUE;
ELSE
  start  :=FALSE;
END_IF
IF JOG_Forward=FALSE AND JOG_Backward =FALSE THEN
 JOG_Backward  :=TRUE;
END_IF
 TON_F (IN:= JOG_Forward, PT:= T#10S, Q=> , ET=> );
 TON_B (IN:= JOG_Backward, PT:= T#10S, Q=> , ET=> );
IF TON_F.Q THEN
 JOG_Forward   :=FALSE;
 JOG_Backward  :=TRUE;
ELSIF TON_B.Q THEN
 JOG_Forward   :=TRUE;
 JOG_Backward  :=FALSE;
END_IF
```

The axis control code is as follows:

```
FOR N :=1 TO CONST BY 1 DO
 FB_Power[N](
  Axis:= AXIS[N]^,
  Enable:= TRUE,
  bRegulatorOn:= start,
  bDriveStart:= TRUE,
  Status=> ,
  bRegulatorRealState=> ,
  bDriveStartRealState=> ,
  Busy=> ,
  Error=> ,
  ErrorID=> );
 FB_Jog[N](
  Axis:= AXIS[N]^,
  JogForward:= FB_Power[N].Status AND JOG_Forward,
  JogBackward:= FB_Power[N].Status AND JOG_Backward,
  Velocity:= JOG_Velocity,
  Acceleration:= JOG_Velocity *10,
  Deceleration:= JOG_Velocity *10,
  Jerk:= ,
  Busy=> ,
  CommandAborted=> ,
  Error=> ,
  ErrorId=> );
END_FOR
```

6.1.5.5.2.6 Running CODESYS Project

1. In the Codesys_Control window, fill in the IP address of the board in the respective textbox. Then, press **Enter** to connect to the board as shown in the figure below:

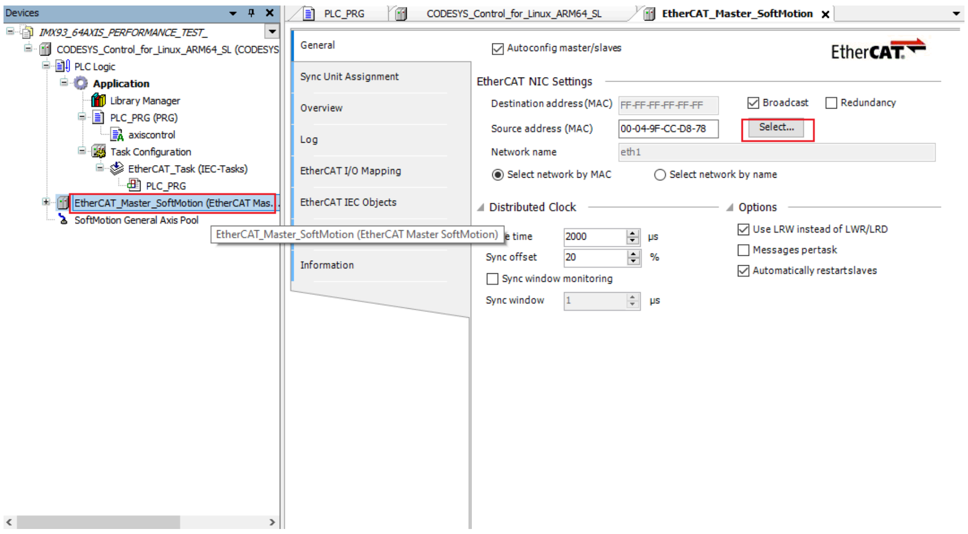


Gateway-1
IP-Address: localhost
Port: 1217

10.193.20.174 (active)
Device Name: imx93evk
Device Address: 0301.1000.2DDC.0AC1.14AE
Device IP Address: 10.193.20.174
Target ID: 0000 0007
Target Type: 4096
Target Vendor: 3S - Smart Software Solutions GmbH
Target Version: 4.5.0.0

Figure 151. Adding IP address of the board

2. Select the network port on the board that is used to communicate with the servo.


Devices
IMX93_64AXIS_PERFORMANCE_TEST_
CODESYS_Control_for_Linux_ARM64_SL (CODESYS
PLC Logic
Application
Library Manager
PLC_PRG (PRG)
axiscontrol
Task Configuration
EtherCAT_Task (IEC-Tasks)
PLC_PRG
EtherCAT_Master_SoftMotion (EtherCAT Mas
SoftMotion General Axis Pool
EtherCAT_Master_SoftMotion (EtherCAT Master SoftMotion)
PLC_PRG
CODESYS_Control_for_Linux_ARM64_SL
EtherCAT_Master_SoftMotion
General
Sync Unit Assignment
Overview
Log
EtherCAT I/O Mapping
EtherCAT IEC Objects
Information
EtherCAT
Select...

[✓] Autoconfig master/slaves

EtherCAT NIC Settings
Destination address (MAC) FF-FF-FF-FF-FF-FF [✓] Broadcast [ ] Redundancy
Source address (MAC) 00-04-9F-CC-D8-78
Network name eth1
[✓ Select network by MAC] Select network by name

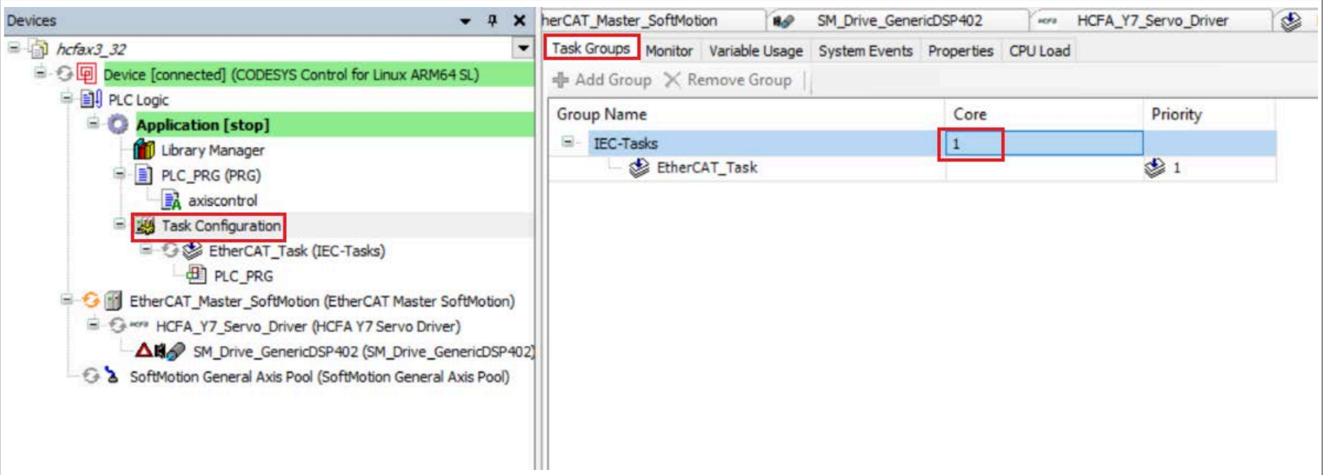
Distributed Clock
Cycle time 2000 µs
Sync offset 20 %
[ ] Sync window monitoring
Sync window 1 µs

Options
[✓] Use LRW instead of LWR/LRD
[ ] Messages per task
[✓] Automatically restart slaves

Figure 152. Selecting the network port

Note: For network port selection, refer to Section 6.1.5.5.1.

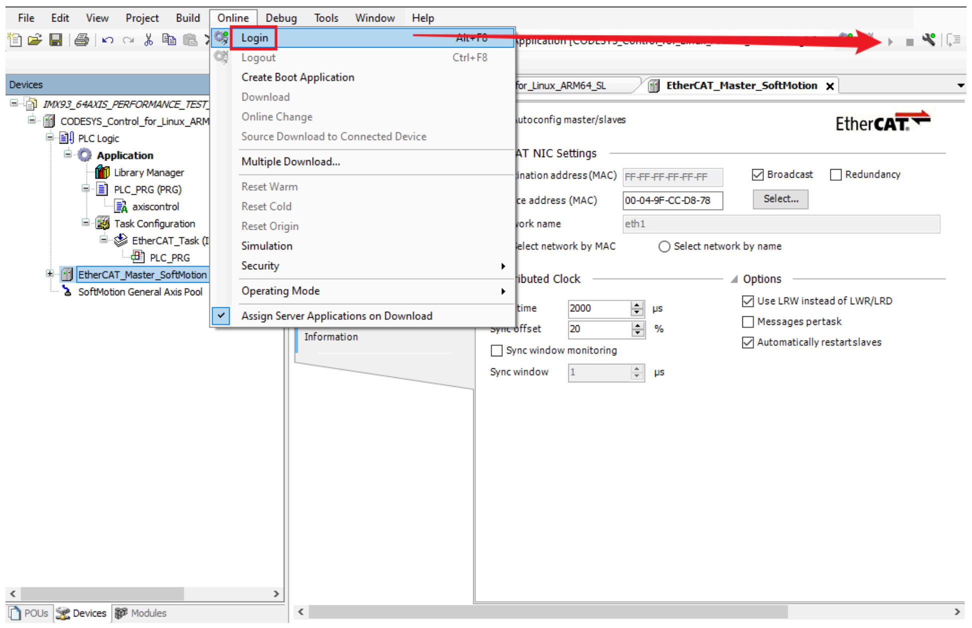
3. Set CPU core isolate


Devices
hcfax3_32
Device [connected] (CODESYS Control for Linux ARM64 SL)
PLC Logic
Application [stop]
Library Manager
PLC_PRG (PRG)
axiscontrol
Task Configuration
EtherCAT_Task (IEC-Tasks)
PLC_PRG
EtherCAT_Master_SoftMotion (EtherCAT Master SoftMotion)
HCFA_Y7_Servo_Driver (HCFA Y7 Servo Driver)
SM_Drive_GenericDSP402 (SM_Drive_GenericDSP402)
SoftMotion General Axis Pool (SoftMotion General Axis Pool)
herCAT_Master_SoftMotion
SM_Drive_GenericDSP402
HCFA_Y7_Servo_Driver
Task Groups
Monitor
Variable Usage
System Events
Properties
CPU Load
Add Group
Remove Group

Group Name	Core	Priority
IEC-Tasks	1	
EtherCAT_Task		1

Figure 153. Codesys CPU core isolate

4. Click **“Login”** and **“Start”** button to run the PLC program:



utoconfig master/slaves
AT NIC Settings
ination address (MAC) FF-FF-FF-FF-FF-FF
Broadcast [✓]
Redundancy
ce address (MAC) 00-04-9F-CC-D8-78
Select...
ork name eth1
elect network by MAC
Select network by name
ibuted Clock
time 2000 µs
sync offset 20 %
Sync window monitoring
Sync window 1 µs
Options
[✓] Use LRW instead of LWR/LRD
Messages pertask
[✓] Automatically restartslaves

Figure 154. Running the PLC program

The motor starts to rotate after all steps above are done.

6.2 FlexCAN and CANopen

The following sections provide an introduction to the FlexCAN standard, details of the CAN bus, the CANopen communication system, details of how to integrate FlexCAN with Real-time Edge, and running a FlexCAN application.

6.2.1 Introduction

The LS1028ARDB board has 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.

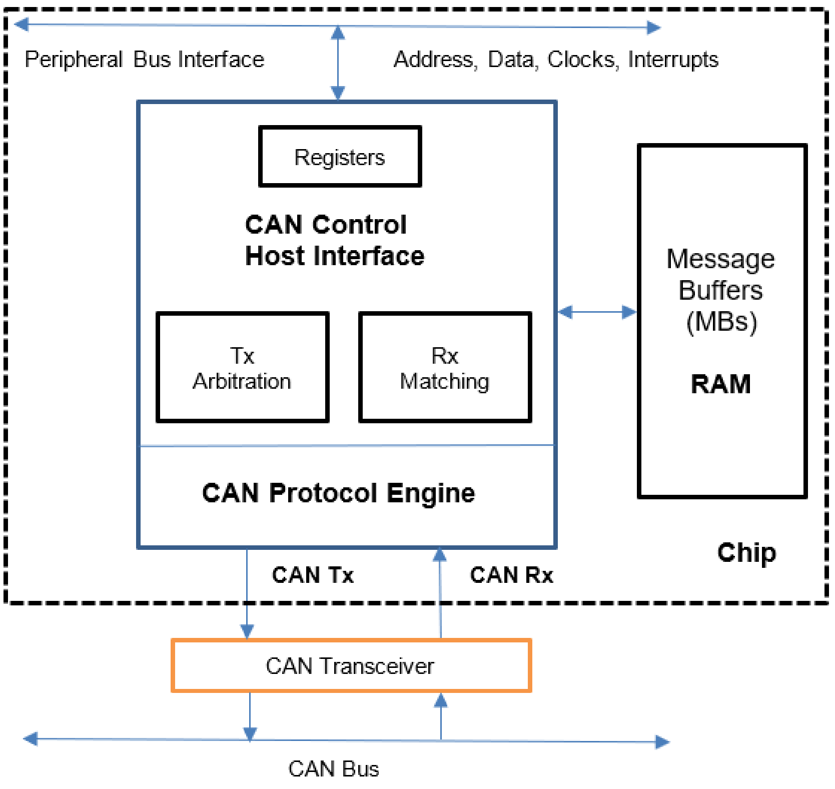


Figure 155. FlexCAN block diagram

6.2.1.1 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.

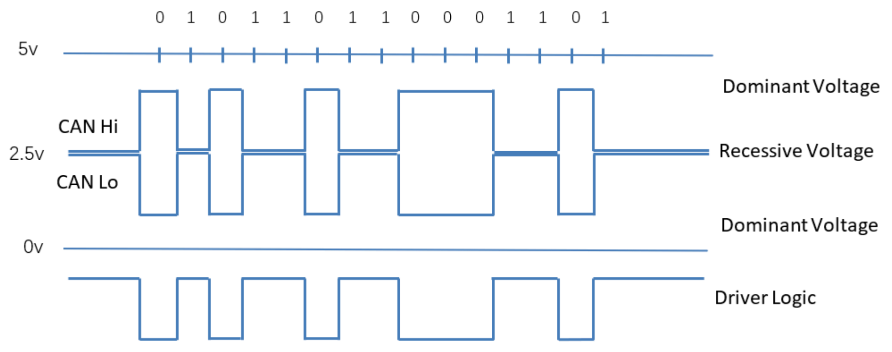


Figure 156. High speed CAN signaling

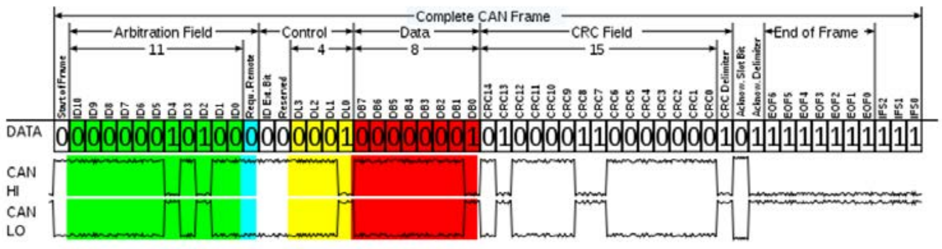


Figure 157. Base frame format

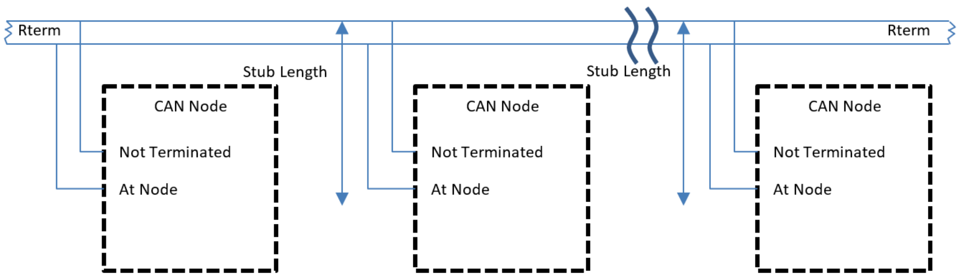


Figure 158. High speed CAN network

6.2.1.2 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.

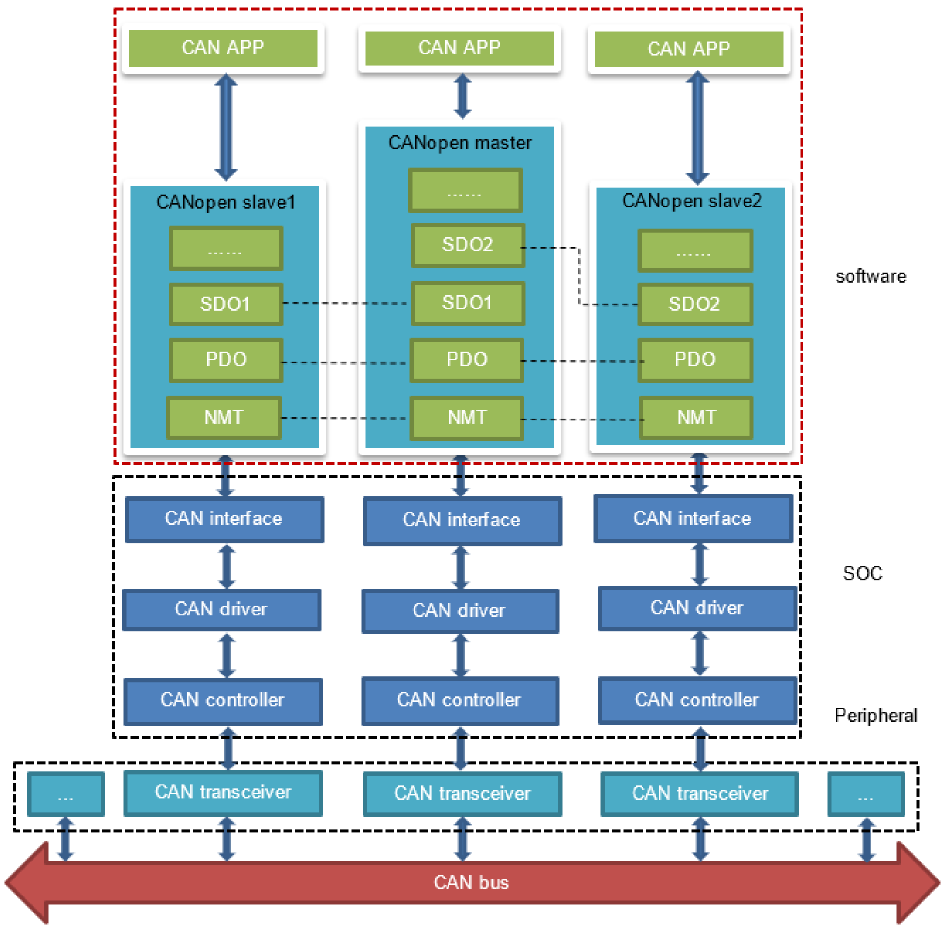


Figure 159. CANopen architecture

6.2.2 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 97. 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

Table 97. CAN Net APIs and their description*...continued*

API name (type)	Description
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 must 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
UNS8 writeNetworkDictCallBack (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS32 count, UNS8 data Type, void *data, SDOCallback_t Callback, UNS8 useBlock Mode)	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

6.2.3 Running a CAN application

The following sections describe the hardware and software preparation steps for running a CAN application.

6.2.3.1 Hardware preparation for LS1028ARDB

For LS1028ARDB, below hardware is required:

- LS1028ARDB board
- Two cables to connect CAN1 and CAN.

The hardware connection diagram is as shown in the following figure.

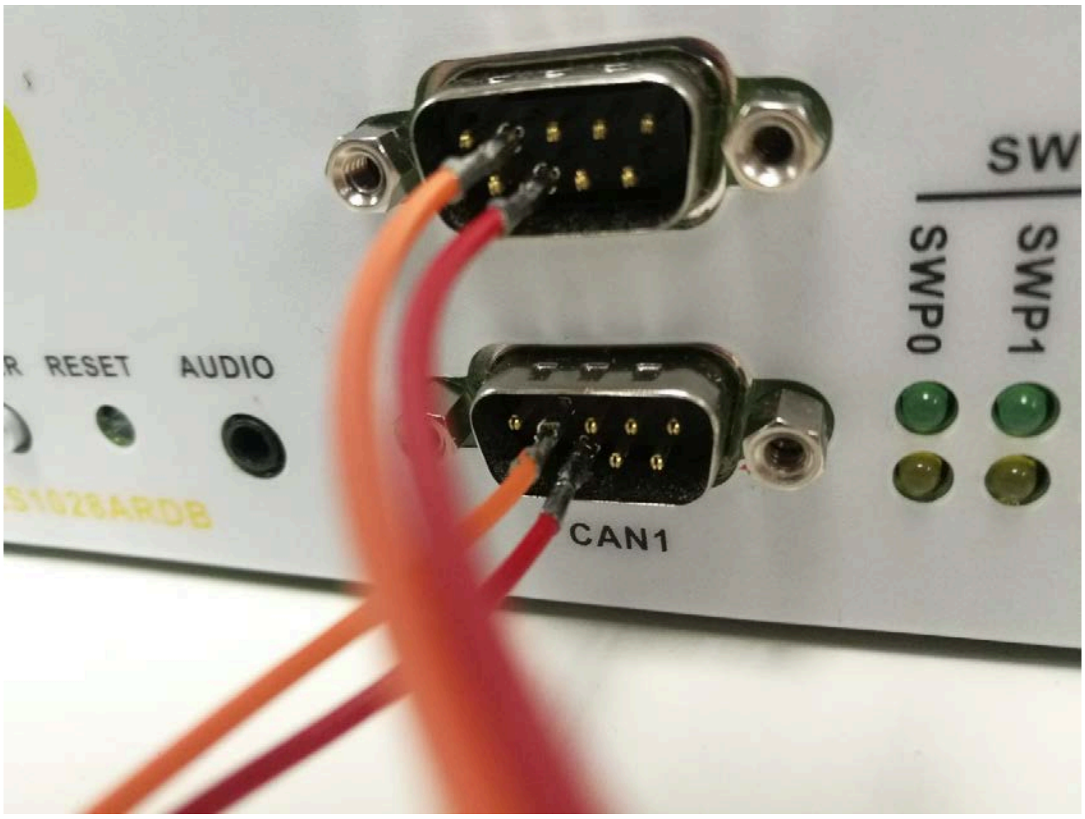


Figure 160. Physical connection for CAN using LS1028ARDB

6.2.3.2 Running the SocketCAN commands

This section describes the steps for running SocketCAN commands that can be performed on LS1028ARDB. These commands are executed on Linux. The standard SocketCAN commands are the following:

1. Open the can0 port.
   ```
   $ ip link set can0 up
   ```
2. Close the can0 port.
   ```
   $ ip link set can0 down
   ```
3. Set the baud rate to 500K for the can0 port
   ```
   $ ip link set can0 type can bitrate 500000
   ```

4. Set can0 port to Loopback mode.

```
$ ip link set can0 type can loopback on
```

5. Send a message through can0. `002` (HEX) is node id, and this value must be 3 characters. `2288DD` (HEX) is a message, and can take a value up to 8 bytes.

```
$ cansend can0 002#2288DD
```

6. Monitor can0 port and wait for receiving data.

```
$ candump can0
```

7. See can0 port details.

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

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

6.2.3.3 Testing CAN bus

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

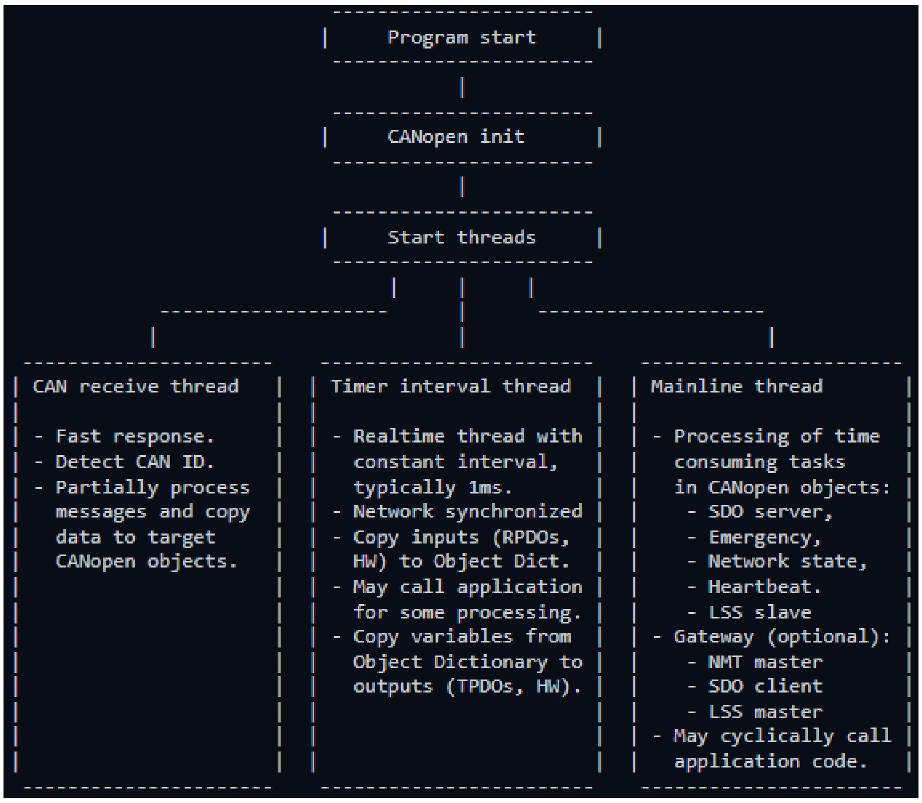
```
[root]# ip link set can0 down
[root]# ip link set can1 down
[root]# ip link set can0 type can loopback off
[root]# ip link set can1 type can loopback off
[root]# ip link set can0 type can bitrate 500000
[root]# ip link set can1 type can bitrate 500000
[root]# ip link set can0 up
[root]# ip link set 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
```

6.2.4 CANopenNode Stack

6.2.4.1 Introduction

CANopenNode is a free and open source CANopen protocol stack. CANopenNode is written in ANSI C in an object-oriented manner. It runs on different microcontrollers as a standalone application or with the RTOS.

Figure 161 shows the flowchart of a typical CANopenNode implementation:



```
                                 -----------------------
                                |     Program start     |
                                 -----------------------
                                            |
                                 -----------------------
                                |     CANopen init      |
                                 -----------------------
                                            |
                                 -----------------------
                                |     Start threads     |
                                 -----------------------
                                      |     |     |
               --------------------         |     --------------------
              |                             |                         |
 ----------------------   -----------------------   -----------------------
| CAN receive thread   | | Timer interval thread | | Mainline thread       |
|                      | |                       | |                       |
| - Fast response.     | | - Realtime thread with| | - Processing of time  |
| - Detect CAN ID.     | |   constant interval,  | |   consuming tasks     |
| - Partially process  | |   typically 1ms.      | |   in CANopen objects: |
|   messages and copy  | | - Network synchronized| |    - SDO server,      |
|   data to target     | | - Copy inputs (RPDOs, | |    - Emergency,       |
|   CANopen objects.   | |   HW) to Object Dict. | |    - Network state,   |
|                      | | - May call application| |    - Heartbeat.       |
|                      | |   for some processing.| |    - LSS slave        |
|                      | | - Copy variables from | | - Gateway (optional): |
|                      | |   Object Dictionary to| |    - NMT master       |
|                      | |   outputs (TPDOs, HW).| |    - SDO client       |
|                      | |                       | |    - LSS master       |
|                      | |                       | | - May cyclically call |
|                      | |                       | |   application code.   |
 ----------------------   -----------------------   -----------------------
```

Figure 161. CANopenNode implementation

All the code of the CANopenNode is non-blocking. Code in source files is collected into objects. Parts of the code can be enabled or disabled. Therefore, only files and parts of the code that are required for the project can be used. See stack configuration in the **301/CO_config.h** file.

In CANopen initialization section all CANopen objects are initialized. CANopen objects are processed cyclically at run time.

The files **CANopen.h** and **CANopen.c** are a joint of all CANopen objects. It seems complex, but they offer flexibility and are suitable for most common configurations of the CANopen objects. CANopen objects can be defined in the global space or allocated dynamically. The Object dictionary (provided by **OD.h/.c** files) can be used by default . Configuration with multiple object dictionaries is also possible by using the **CO_config_t** structure. CANopen.h and **CANopen.c** files can also be only a reference for a more customized implementation of a CANopenNode based device.

The Object Dictionary is a collection of all network accessible variables and offers most flexible usage. OD variables can be initialized by the object dictionary or the application can specify its own read/write access functions for specific OD variables. Groups of OD variables can also be stored to the non-volatile memory, either by using a command or automatically.

6.2.4.2 Running CANopenNode on Cortex-M core

This section describes the board settings for the CANopenNode Stack implementation demo.

6.2.4.2.1 MIMXRT1180-EVK board

The verification of CANopenNode requires two sets of boards. Connect the two base boards together via the CAN3 instance (J35).

Table 98. Board connection using CAN3 (J35)

	Board 1	Board 2
CAN_H	J35-1	J35-1
CAN_L	J35-3	J35-3
GND	J35-2	J35-2

6.2.4.2.2 i.MX 943-EVK board

The verification of CANopenNode requires two sets of boards. You must connect the two base boards together via the CAN1 instance (J17).

Table 99. Board connection using CAN1 (J17)

	Board 1	Board 2
CAN_H	J17-2	J17-2
CAN_L	J17-3	J17-3
GND	J17-4	J17-4

6.2.4.2.3 Running the image

To verify the CANopenNode functionality, you must prepare two images. One of the images implements the capabilities of the CANopen manager node. The other implements the capabilities of CANopen device node.

6.2.4.2.3.1 Pre-built images

You must first run the CANopen device image and then the manager image.

For easy verification, the pre-built images described in this section can be used:

MIMXRT1180-EVK

For MIMXRT1180-EVK, the pre-built images are located at:

meta-rtos-industrial/recipes-kernel/mcux-kernel/industrial-examples_evkmimxrt1180 at walnascar · nxp-real-time-edge-sw/meta-rtos-industrial · GitHub

Table 100. Pre-built images for MIMXRT1180-EVK

Image Type	File name
CANopen Device image	canopen_device_bm_cm33.elf
CANopen Manager image	canopen_manager_bm_cm33.elf

1. Connect the J-Link to the board.

2. To create a GDB server, in PowerShell, enter the command below:

```
JLinkGDBServer -select USB -device MIMXRT1189xxx8_M33 -endian little -if
 SWD -speed 15000 -ir -LocalhostOnly -nologtofile -port 2331 -SWOPort 2332 -
TelnetPort 2333
```

3. To download the image, in PowerShell, enter the below command:

```
arm-none-eabi-gdb -ex "target remote localhost:2331" -ex "monitor reset" -ex
 "monitor halt" -ex "load" image.elf
```

i.MX943-EVK

For i.MX943-EVK, the pre-built images are located at:

meta-rtos-industrial/recipes-kernel/mcux-kernel/industrial-examples_imx943 at walnascar · nxp-real-time-edge-sw/meta-rtos-industrial · GitHub

CANopen Device image

- LPDDR4: canopen_device_bm_cm33_core1_lpddr4_flash.bin
- LPDDR5: canopen_device_bm_cm33_core1_lpddr5_flash.bin

CANopen Manager image

- LPDDR4: canopen_manager_bm_cm33_core1_lpddr4_flash.bin
- LPDDR5: canopen_manager_bm_cm33_core1_lpddr5_flash.bin

Use the steps below to run the image:

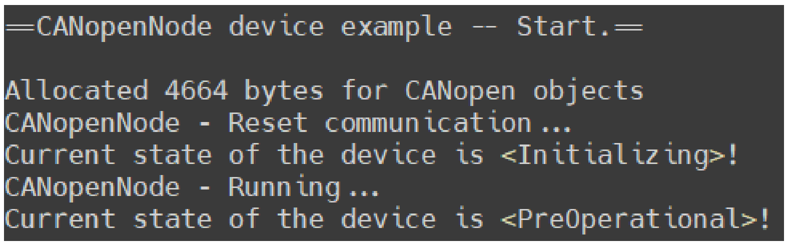
1. Burn `flash.bin` to MicroSD at 32 K(0x8000) offset using the `dd` command.

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

   Note: *__flash.bin__ is the pre-built image. And the __/dev/sdx__ is the SD card device on your PC.*
2. Plug the MicroSD card to the board.
3. Change the boot mode to **SW4[1:4] = x011** for SD boot.
4. Enable MCU UARTs.
   Follow the description on the link **Enable MCU UARTs — MCUXpresso SDK Documentation** to setup the UART.
5. Power on the board.
   Refer to **Getting Started with Package — MCUXpresso SDK Documentation** for details.

6.2.4.2.3.2 UART log

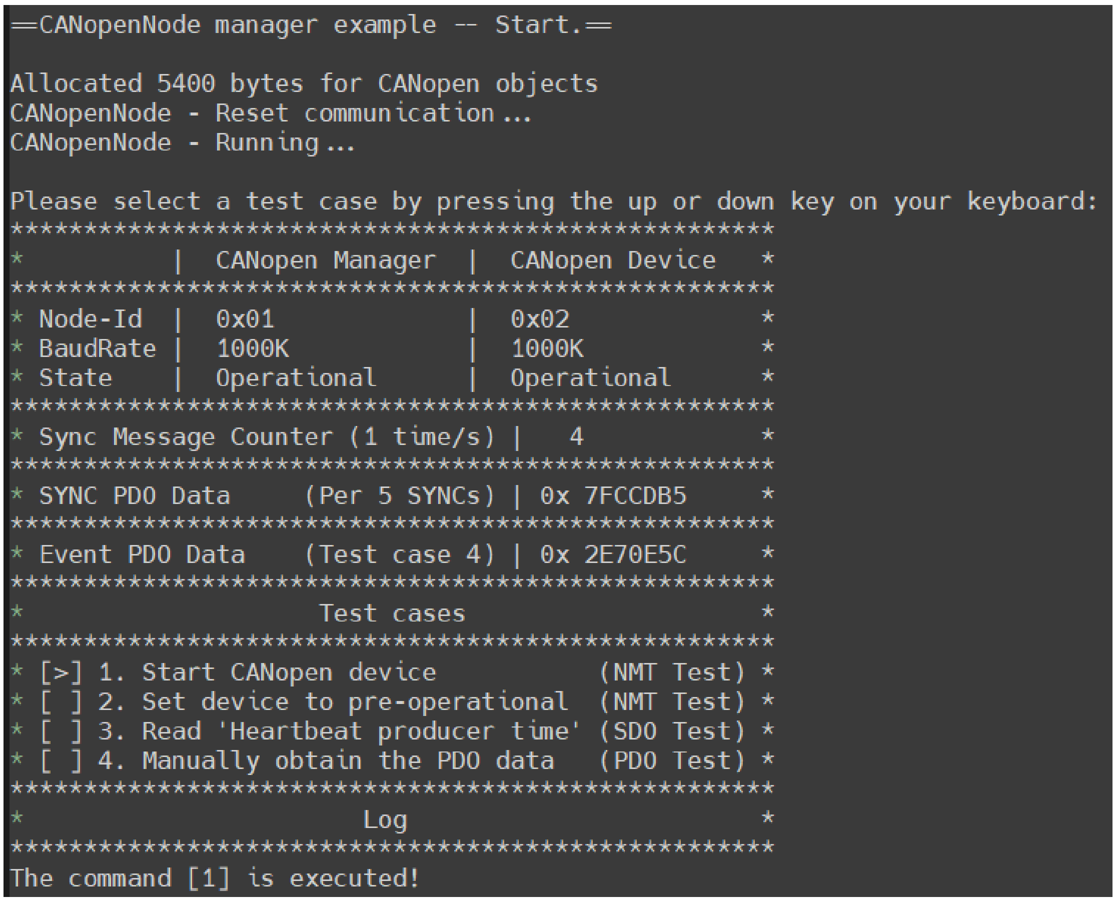
- **CANopenNode device node**
  When the demo runs, the log seen on the terminal is similar to the one below:



```
==CANopenNode device example -- Start.==

Allocated 4664 bytes for CANopen objects
CANopenNode - Reset communication...
Current state of the device is <Initializing>!
CANopenNode - Running...
Current state of the device is <PreOperational>!
```

Figure 162. CANopenNode device node log

- **CANopenNode manager node**
  When the demo runs, the log seen on the terminal is similar to the one shown below:



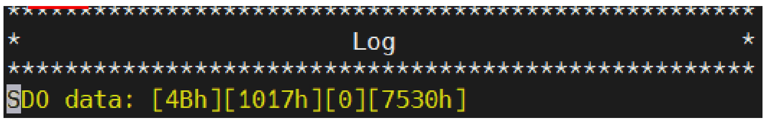
```
==CANopenNode manager example -- Start.==

Allocated 5400 bytes for CANopen objects
CANopenNode - Reset communication...
CANopenNode - Running...

Please select a test case by pressing the up or down key on your keyboard:
****************************************************
*          |  CANopen Manager  |  CANopen Device   *
****************************************************
* Node-Id  |  0x01             |  0x02             *
* BaudRate |  1000K            |  1000K            *
* State    |  Operational      |  Operational      *
****************************************************
* Sync Message Counter (1 time/s) |   4            *
****************************************************
* SYNC PDO Data     (Per 5 SYNCs) | 0x 7FCCDB5     *
****************************************************
* Event PDO Data    (Test case 4) | 0x 2E70E5C     *
****************************************************
*                    Test cases                    *
****************************************************
* [>] 1. Start CANopen device           (NMT Test) *
* [ ] 2. Set device to pre-operational  (NMT Test) *
* [ ] 3. Read 'Heartbeat producer time' (SDO Test) *
* [ ] 4. Manually obtain the PDO data   (PDO Test) *
****************************************************
*                       Log                        *
****************************************************
The command [1] is executed!
```

Figure 163. CANopenNode manager node log

6.2.4.2.3.3 Test cases

You can select a specific test case by pressing **the up and down keys** on the keyboard. When **the Enter key** is pressed, the test case selected by the ">" key is executed.

- **Test case 1**
  When the CANopenNode device node is powered on, the CANopen state is in **PreOperational**. When test case 1 is executed on the CANopenNode manager side, the device node outputs a message, "`Current state of the device is <Operational>`".
- **Test case 2**
  When test case 2 is executed, the state of the CANopen device node is changed to **PreOperational**. The device node outputs a message, "`Current state of the device is <PreOperational>`".
- **Test case 3**
  This test case is used to get the "heartbeat producer cycle" of the object dictionary on the CANopen device node, based on CANopen SDO protocol. When it is executed, the CANopen manager node outputs the below log:



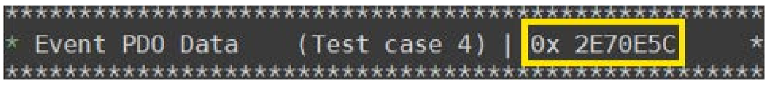
```
****************************************************
*                       Log                        *
****************************************************
SDO data: [4Bh][1017h][0][7530h]
```

Figure 164. CANopen manager node output

1. [4Bh]: SDO header

2. [1017h]: The index of the current object dictionary
3. [0]: The subindex value of the current object dictionary
4. [7530h]: The value of the current object dictionary (Unit ms).

- **Test case 4**
  This test case is used to test CANopen PDO protocol. Each time this test case is executed, the data value in the Figure 165 is updated.



```
*****************************************************************
* Event PDO Data     (Test case 4) | 0x 2E70E5C      *
*****************************************************************
```

Figure 165. CANopen PDO protocol sample test log

  The meaning of this data value is defined by the user.

***Note:** The PDO protocol is enabled only when the CANopen state is "Operational". Therefore, if the status of the CANopenNode device node is not "Operational", **test case 1 must be executed before test case 4**.*

6.2.4.2.4 Running CANopen on Cortex-A core

The CANopen demo running on Cortex-A core is implemented based on CANopenLinux. CANopenLinux is the official project running on Linux devices, which is based on the CANopenNode.

6.2.4.2.4.1 Running the CANopen demo

1. Start the canopen-master service:

```
root@imx943-19x19-lpddr5-evk:~# systemctl start --user canopen-master
```

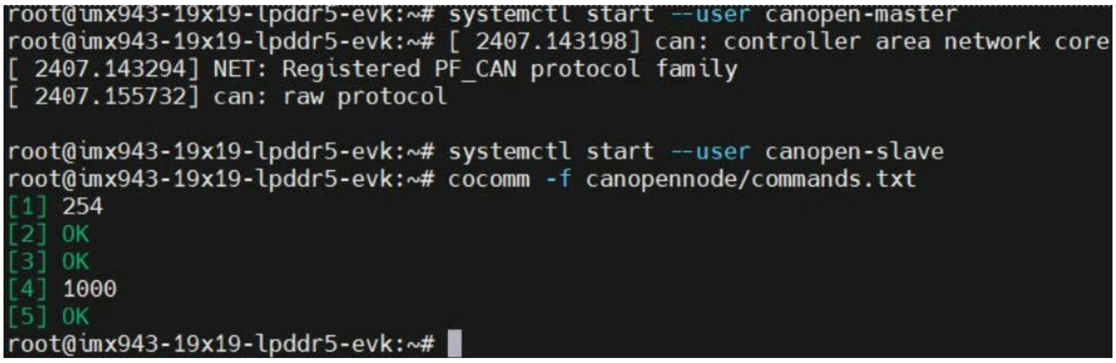
2. Start the canopen-slave service:

```
root@imx943-19x19-lpddr5-evk:~# systemctl start --user canopen-slave
```

3. Pass the list of commands to the canopen-master service:

```
root@imx943-19x19-lpddr5-evk:~# cocomm -f canopennode/commands.txt
```

After the above commands are executed, the below log can be seen on the terminal:



```
root@imx943-19x19-lpddr5-evk:~# systemctl start --user canopen-master
root@imx943-19x19-lpddr5-evk:~# [ 2407.143198] can: controller area network core
[ 2407.143294] NET: Registered PF_CAN protocol family
[ 2407.155732] can: raw protocol

root@imx943-19x19-lpddr5-evk:~# systemctl start --user canopen-slave
root@imx943-19x19-lpddr5-evk:~# cocomm -f canopennode/commands.txt
[1] 254
[2] OK
[3] OK
[4] 1000
[5] OK
root@imx943-19x19-lpddr5-evk:~#
```

Figure 166. CANopen demo running on imx943-19x19-lpddr5-evk

6.2.4.2.4.2 Setting i.MX 943-EVK board

To run the CANopen demo on Cortex-A core, only one i.MX 943-EVK board is required.

The pWin connections are as shown below:

Table 101. Pin connections on i.MX 943-EVK board

Signal Name	From (CAN0)	To (CAN1)
CAN_H	J18-2	J19-2
CAN_L	J18-3	J19-3
GND	J18-4	J19-4

6.2.4.2.4.3 CANopen configurations and commands

The below paths are all on the i.MX 943-EVK board.

1. This is a description of the "`/etc/canopenlinux.conf`" file.

/etc/canopenlinux.conf

```
# CANopenLinux Configuration File
# CAN interface name
CAN_INTERFACE_M=can0
CAN_INTERFACE_S=can1
# CANopen Node ID (1-127, 0 for LSS)
NODE_ID_M=1
NODE_ID_S=4
# CAN bitrate (optional, handled by systemd service)
CAN_BITRATE=250000
# Command interface type (stdio, local socket, tcp socket)
COMMAND_INTERFACE=local-/tmp/CO_command_socket
```

2. This is a description of the "/etc/systemd/user/canopen-master.service" file.

/etc/systemd/user/canopen-master.service

```
[Unit]
Description=CANopenMaster - CANopen stack for the master node
Documentation=https://github.com/CANopenNode/CANopenLinux
After=network.target
Wants=network.target
[Service]
Type=simple
EnvironmentFile=/etc/canopenlinux.conf
ExecStartPre=/bin/sh -c 'ip
ExecStartPre=/bin/sh -c 'ip
ExecStart=/usr/bin/canopend $CAN_INTERFACE_M -i $NODE_ID_M -c
 $COMMAND_INTERFACE
ExecStop=/bin/sh -c 'ip
Restart=always
RestartSec=5
User=root
Group=root
[Install]
WantedBy=multi-user.target
```

3. To configure the CANopen service, modify the parameter values in the configuration file above.
4. Use the command below to implement a custom canopen-master service by compiling a new `canopend` command based on CANopenLinux.

/etc/systemd/user/canopen-slave.service

```
[Unit]
Description=CANopenSlave - CANopen stack for the slave node
```

```
Documentation=https://github.com/CANopenNode/CANopenLinux
After=network.target
Wants=network.target
[Service]
Type=simple
EnvironmentFile=/etc/canopenlinux.conf
ExecStartPre=/bin/sh -c 'ip
ExecStartPre=/bin/sh -c 'ip
ExecStart=/usr/bin/canopend $CAN_INTERFACE_S -i $NODE_ID_S
ExecStop=/bin/sh -c 'ip
Restart=always
RestartSec=5
User=root
Group=root
[Install]
WantedBy=multi-user.target
```

5. We can compile new `canopend` command based on CANopenLinux to implement a custom canopen-slave service.

   `/root/canopennode/commands.txt`

   ```
   [1] 4 read 0x1400 2 u8
   [2] 4 write 0x1400 2 u8 255
   [3] 4 write 0x1017 0 u16 1000
   [4] 4 read 0x1017 0 u16
   [5] 4 start
   ```

   This file defines a set of standard CANopen commands (CiA309-3). The **`cocomm`** command allows to parse this file, which is then passed to the canopen-master service.
   Command format:

   ```
   [Line number] slave-node-id command [index sub-index type [value]]
   ```

   ```
   /usr/bin/canopend
   ```

In the current demo, both the master and slave services are implemented using the same **canopend**. We can compile two different canopend commands based on CANopenLinux. Then, replace the canopend command in the service files.

The **cocomm** is a command line program, which establishes socket connection with the canopen-master service. This command can parse the `commands.txt` file, and pass CANopen commands to the canopen-master service. The commands defined in `commands.txt` can read and write the object dictionary and control the state of slave nodes.

```
/usr/bin/cocomm
```

6.3 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

6.3.1 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 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).

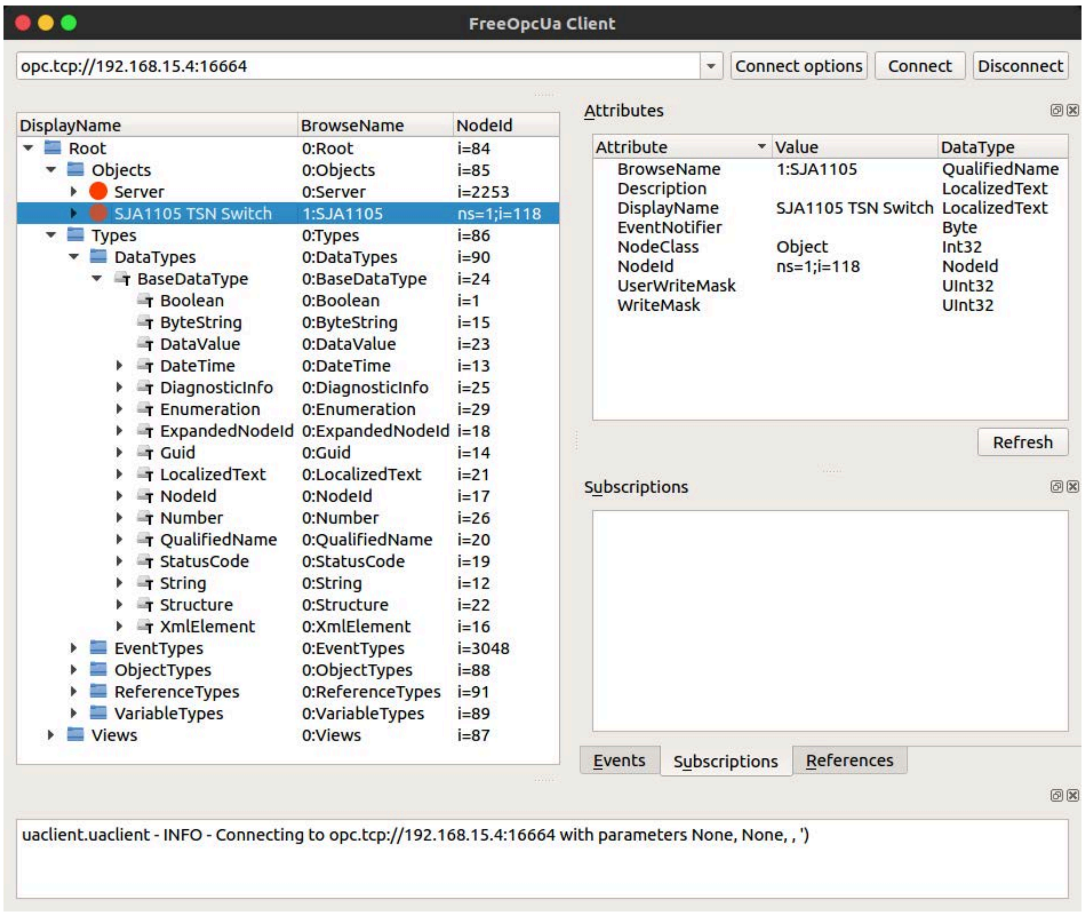
OPC-enabled applications can be composed: an industrial device may run an OPC client and feed the collected data into another physical process, while also exposing the latter by running an OPC server.

6.3.2 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)



DisplayName	BrowseName	NodeId
Root	0:Root	i=84
Objects	0:Objects	i=85
Server	0:Server	i=2253
SJA1105 TSN Switch	1:SJA1105	ns=1;i=118
Types	0:Types	i=86
DataTypes	0:DataTypes	i=90
BaseDataType	0:BaseDataType	i=24
Boolean	0:Boolean	i=1
ByteString	0:ByteString	i=15
DataValue	0:DataValue	i=23
DateTime	0:DateTime	i=13
DiagnosticInfo	0:DiagnosticInfo	i=25
Enumeration	0:Enumeration	i=29
ExpandedNodeId	0:ExpandedNodeId	i=18
Guid	0:Guid	i=14
LocalizedText	0:LocalizedText	i=21
NodeId	0:NodeId	i=17
Number	0:Number	i=26
QualifiedName	0:QualifiedName	i=20
StatusCode	0:StatusCode	i=19
String	0:String	i=12
Structure	0:Structure	i=22
XmlElement	0:XmlElement	i=16
EventTypes	0:EventTypes	i=3048
ObjectTypes	0:ObjectTypes	i=88
ReferenceTypes	0:ReferenceTypes	i=91
VariableTypes	0:VariableTypes	i=89
Views	0:Views	i=87

Attribute	Value	DataType
BrowseName	1:SJA1105	QualifiedName
Description		LocalizedText
DisplayName	SJA1105 TSN Switch	LocalizedText
EventNotifier		Byte
NodeClass	Object	Int32
NodeId	ns=1;i=118	NodeId
UserWriteMask		UInt32
WriteMask		UInt32

```
uaclient.uaclient - INFO - Connecting to opc.tcp://192.168.15.4:16664 with parameters None, None, , ')
```

Figure 167. 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`.

6.3.3 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*.

6.3.4 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*.

6.3.5 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.

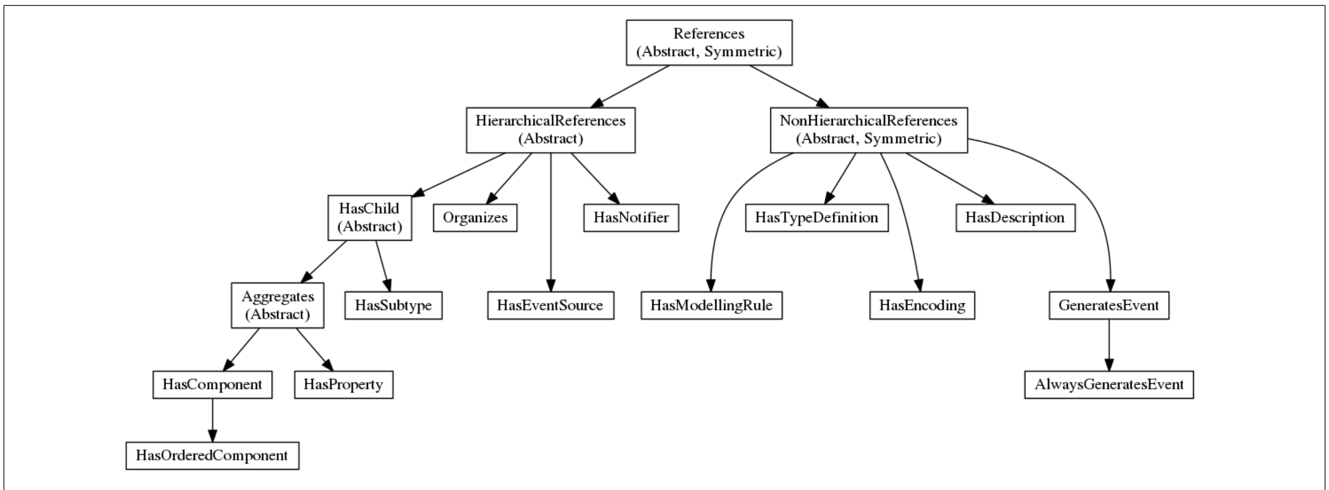
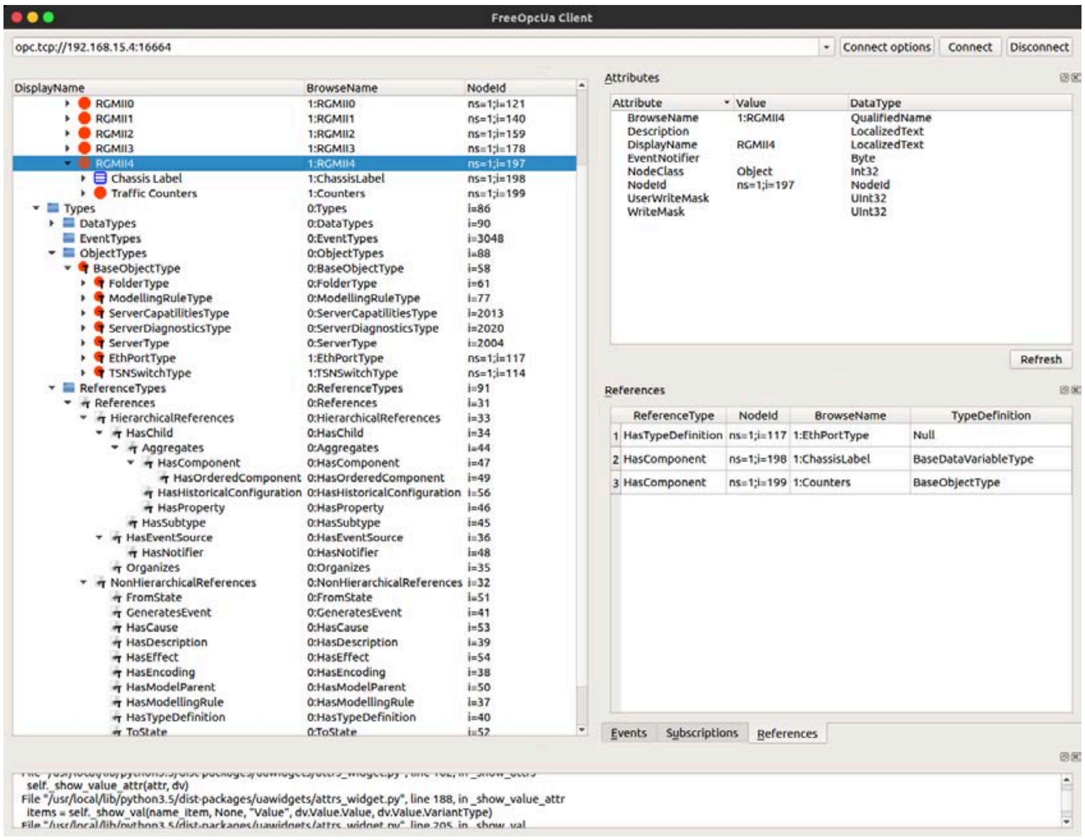


Figure 168. Hierarchy of the standard ReferenceTypes, defined in Part 3 of the OPC UA specification (Image taken from www.open62541.org)

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.



opc.tcp://192.168.15.4:16664

DisplayName	BrowseName	NodeId
RGMII0	1:RGMII0	ns=1;i=121
RGMII1	1:RGMII1	ns=1;i=140
RGMII2	1:RGMII2	ns=1;i=159
RGMII3	1:RGMII3	ns=1;i=178
RGMII4	1:RGMII4	ns=1;i=197
Chassis Label	1:ChassisLabel	ns=1;i=198
Traffic Counters	1:Counters	ns=1;i=199
Types	0:Types	i=86
DataTypes	0:DataTypes	i=90
EventTypes	0:EventTypes	i=3048
ObjectTypes	0:ObjectTypes	i=88
BaseObjectType	0:BaseObjectType	i=58
FolderType	0:FolderType	i=61
ModellingRuleType	0:ModellingRuleType	i=77
ServerCapatilitiesType	0:ServerCapatilitiesType	i=2013
ServerDiagnosticsType	0:ServerDiagnosticsType	i=2020
ServerType	0:ServerType	i=2004
EthPortType	1:EthPortType	ns=1;i=117
TSNSwitchType	1:TSNSwitchType	ns=1;i=114
ReferenceTypes	0:ReferenceTypes	i=91
References	0:References	i=31
HierarchicalReferences	0:HierarchicalReferences	i=33
HasChild	0:HasChild	i=34
Aggregates	0:Aggregates	i=44
HasComponent	0:HasComponent	i=47
HasOrderedComponent	0:HasOrderedComponent	i=49
HasHistoricalConfiguration	0:HasHistoricalConfiguration	i=56
HasProperty	0:HasProperty	i=46
HasSubtype	0:HasSubtype	i=45
HasEventSource	0:HasEventSource	i=36
HasNotifier	0:HasNotifier	i=48
Organizes	0:Organizes	i=35
NonHierarchicalReferences	0:NonHierarchicalReferences	i=32
FromState	0:FromState	i=51
GeneratesEvent	0:GeneratesEvent	i=41
HasCause	0:HasCause	i=53
HasDescription	0:HasDescription	i=39
HasEffect	0:HasEffect	i=54
HasEncoding	0:HasEncoding	i=38
HasModelParent	0:HasModelParent	i=50
HasModellingRule	0:HasModellingRule	i=37
HasTypeDefinition	0:HasTypeDefinition	i=40
ToState	0:ToState	i=52

Attributes

Attribute	Value	DataType
BrowseName	1:RGMII4	QualifiedName
Description		LocalizedText
DisplayName	RGMII4	LocalizedText
EventNotifier		Byte
NodeClass	Object	Int32
NodeId	ns=1;i=197	NodeId
UserWriteMask		UInt32
WriteMask		UInt32

References

	ReferenceType	NodeId	BrowseName	TypeDefinition
1	HasTypeDefinition	ns=1;i=117	1:EthPortType	Null
2	HasComponent	ns=1;i=198	1:ChassisLabel	BaseDataVariableType
3	HasComponent	ns=1;i=199	1:Counters	BaseObjectType

```
  self._show_value_attr(attr, dv)
File "/usr/local/lib/python3.5/dist-packages/uawidgets/attrs_widget.py", line 188, in _show_value_attr
  items = self._show_val(name_item, None, "Value", dv.Value.Value, dv.Value.VariantType)
File "/usr/local/lib/python3.5/dist-packages/uawidgets/attrs_widget.py", line 205, in _show_val
```

Figure 169. 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`.

6.3.6 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 \
```

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

6.3.7 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.

6.3.7.1 OPC UA Pub/Sub introduction

The 14th part of the OPC UA specification defines the OPC UA PubSub communication model. It provides an OPC UA Publish Subscribe model that complements the Client/Server communication model.

In PubSub, the participating OPC UA applications can assume the roles of Publishers and Subscribers. Publishers are the sources of data, while Subscribers consume that data. Communication in PubSub is message-based. Publishers send messages to a Message Oriented Middleware, without knowledge of what, if any, Subscribers there may be. Similarly, Subscribers express interest in specific types of data, and process messages that contain this data, without knowledge of what Publishers there are.

To cover a large number of use cases, OPC UA PubSub supports two largely different Message Oriented Middleware variants. These are:

1. A broker-based form, where the core component of the Message Oriented Middleware is a message Broker. Subscribers and Publishers use standard messaging protocols like AMQP or MQTT to communicate with the Broker.

2. A broker-less form, where the Message Oriented Middleware is the network infrastructure that is able to route datagram-based messages. Subscribers and Publishers use datagram protocols such as UDP or raw Ethernet as the transport protocol. In this form, the data sources (Publishers) and the data consumers (Subscribers) join a multicast group. Any data sent by a source to the group goes to all consumers subscribed to the same group. Joining is trivial in Ethernet (Layer 2): the network broadcasts multicast frames everywhere, leaving it to receivers to decide whether to pick up the frame based on the destination address. The OPC UA PubSub sample applications in this section 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.

6.3.7.2 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.

6.3.7.3 OPC UA PubSub components

The following figure shows the different components of OPC UA PubSub and their relation to each other. The WriterGroup, DataSetWriter, and PublishedDataSet components define the data acquisition for DataSets, message generation, and transmission on the Publisher side. These parameters should be known on the Subscriber side to configure DataSetReaders to filter and process DataSetMessages.

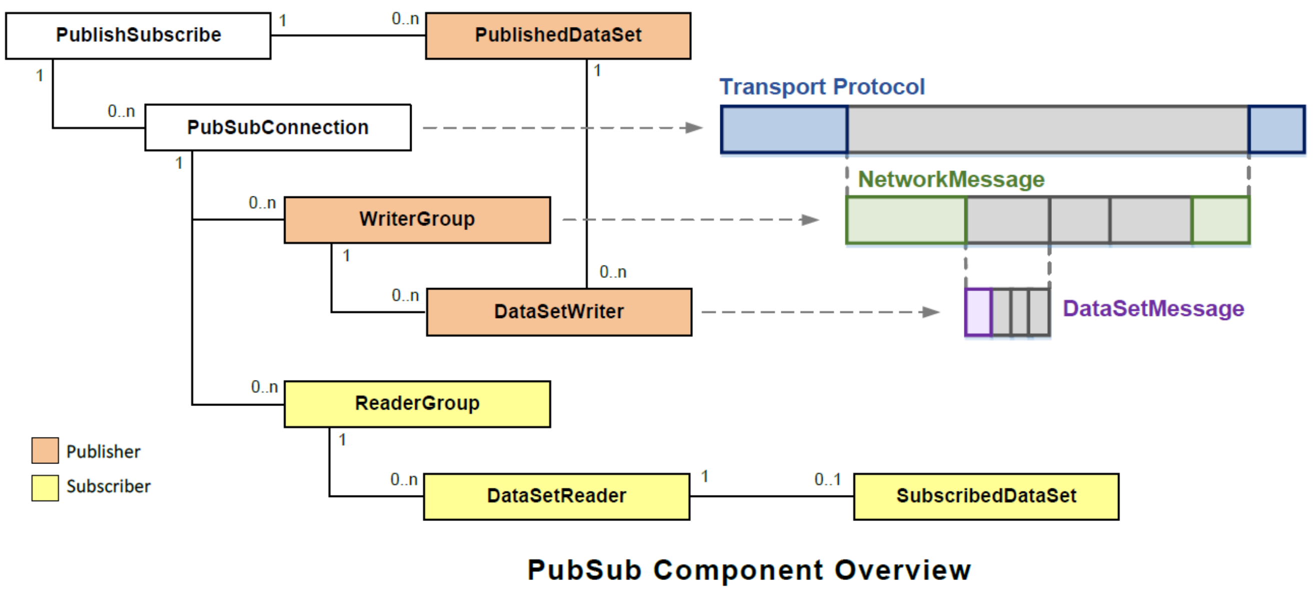


Figure 170. OPC UA PubSub components

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.

6.3.7.4 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. This 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.MX 8M 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.MX 8M Plus. Another DataSetField is the TX HW timestamp of the published packet.

3. A WriteGroup is added with WriterGroup ID hard-coded to 100 and the publishing interval set to 1 second. The Publisher transmits one packet per second cyclically. Each cycle aligns with whole second using Linux system clock ***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 (for example, eth1 for ENET2 on i.MX 8M Plus LPDDR4 EVK). Note that the Subscriber uses the same network address URL as the Publisher.
2. A ReaderGroup is added.
   Note: *The Subscriber also runs cyclically with 1 second cycle time to receive packet. Each cycle aligns with whole second with 500 µs offset to account for the application delay on the publisher and the path delay from publisher to subscriber.*
   Linux system clock ***CLOCK_REALTIME*** is used.
3. A DataSetReader is added and configured with Publisher ID of 2234, WriterGroup ID of 100, and DataSetWriter ID of 62541. Note that all these parameters match the corresponding settings on the Publisher in order to filter the DataSetMessages to be processed by the DataSetReader.
4. A SubscribedDataSet is added with a list of targetVariables. The targetVariables corresponds to the DataSetFields in the PublishedDataSet on the Publisher.
5. Besides the above, the RX HW timestamp of the received packet is taken and the path delay is calculated by subtracting the RX HW timestamp taken on the Subscriber from the TX HW timestamp taken on the Publisher for the same packet. To achieve this, both the Publisher and the Subscriber must have synchronized time. This is achieved by running gPTP.

Both the Publisher and the Subscriber also run a OPC UA server. Users can use a OPC UA client running on a host PC to browse the server's Address Space on either the Publisher or the Subscriber.

6.3.7.5 OPC UA PubSub sample application over TSN

Hardware Requirements:

1. Two or three i.MX 8DXL LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK, i.MX 93 EVK boards, or i.MX 95 EVK boards
2. One LS1028ARDB board

Software Requirements:

1. linuxptp package which provides tools such as `ptp4l`, `phc2sys`, `phc_ctl`, and `hwstamp_ctl`.
2. iproute2 package which provides tools such as 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 `/root/open62541_example`.

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

The following sections use i.MX 8M Plus LPDDR4 EVK board as an example.

6.3.7.5.1 Case #1: two i.MX 8M Plus LPDDR4 EVK connected back-to-back

A simple setup could be made by connecting two i.MX 8M Plus LPDDR4 EVK boards back-to-back via **ENET2** as shown in Figure 171. One i.MX 8M 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 the LAN (that is, the office network). The actual device name in Linux might be different.

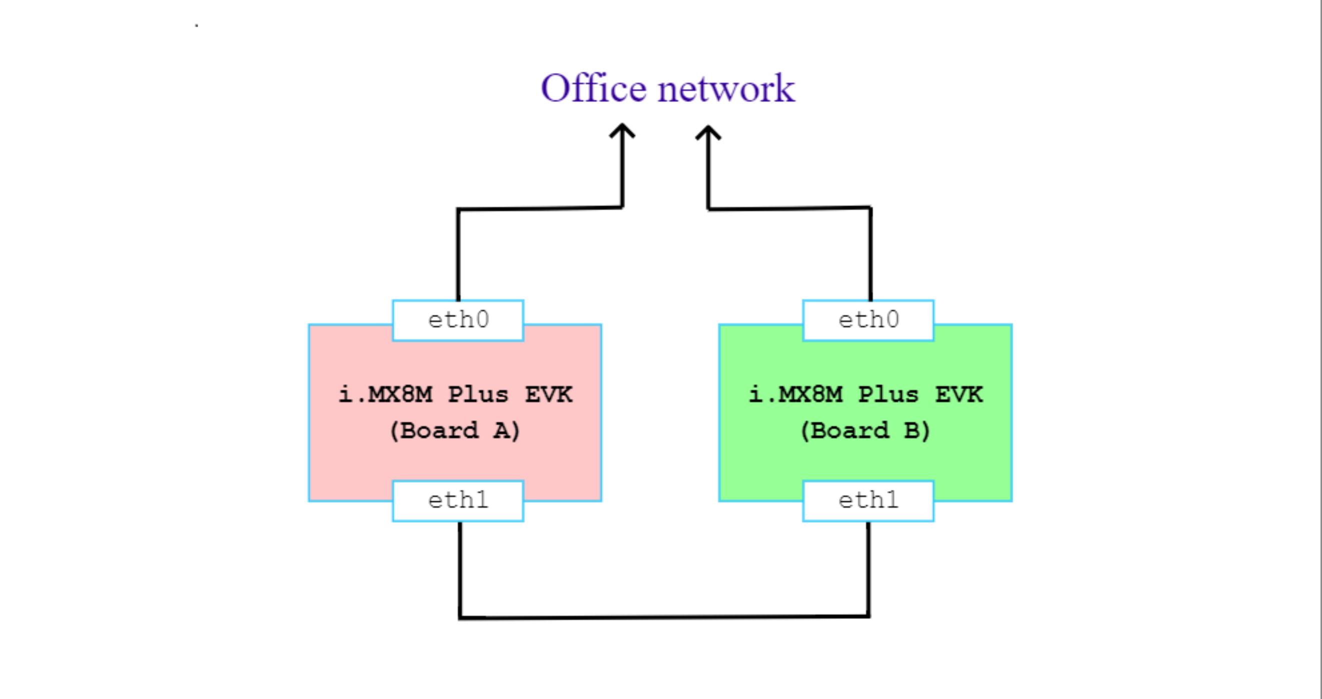


Figure 171. Two i.MX 8M Plus LPDDR4 EVK boards connected back-to-back

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

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

  Using the command *ethtool eth1* must display a message: `Link detected: yes`. Otherwise, check the hardware connection.
- On the Publisher (i.MX 8M 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.MX 8M Plus LPDDR4 EVK - Board B), use `hwstamp_ctl` to change the RX hardware timestamp setting to '`time stamp any incoming packet`' to get the RX hardware timestamp of the packets transmitted by the Publisher.
  On the Publisher (i.MX 8M 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 &
```

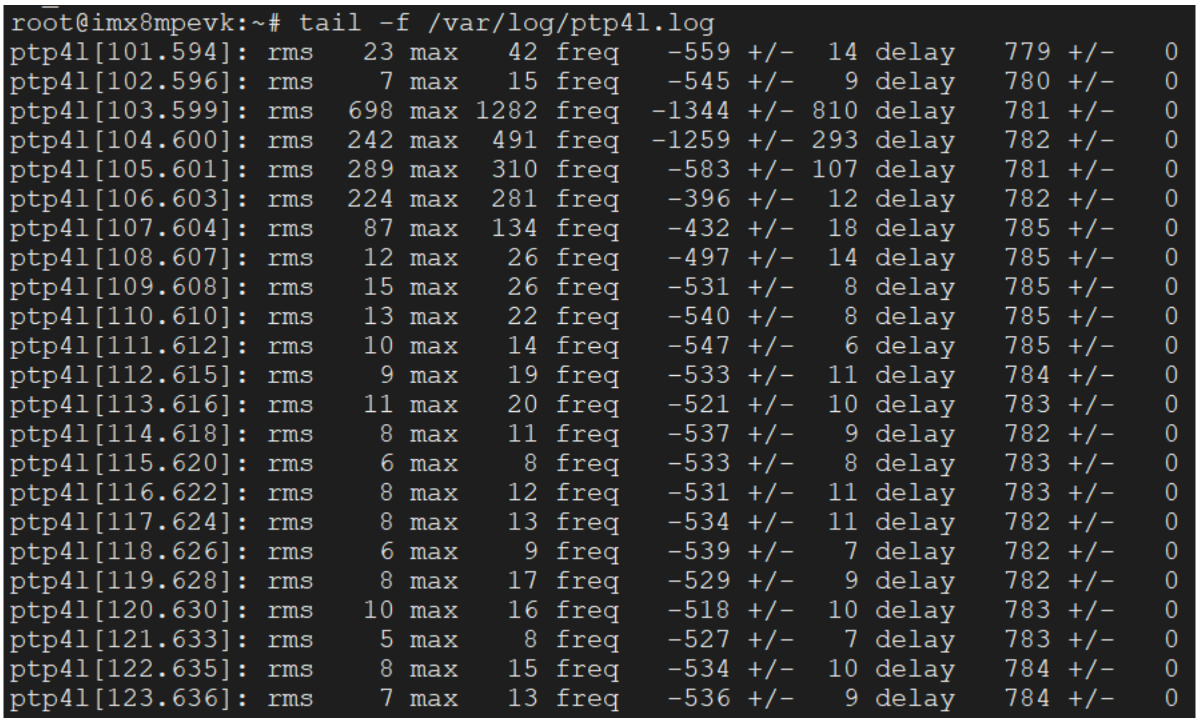
  On the Subscriber (i.MX 8M Plus LPDDR4 EVK - Board B):

```
# 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
```

It is recommended to SSH to both boards to check the logs of ptp4l and phc2sys to continue to execute other commands on the serial console). On both boards, observe the logs of ptp4l and phc2sys to check the time synchronization progress by using the below commands:

```
# tail -f /var/log/ptp4l.log
# tail -f /var/log/phc2sys.log
```

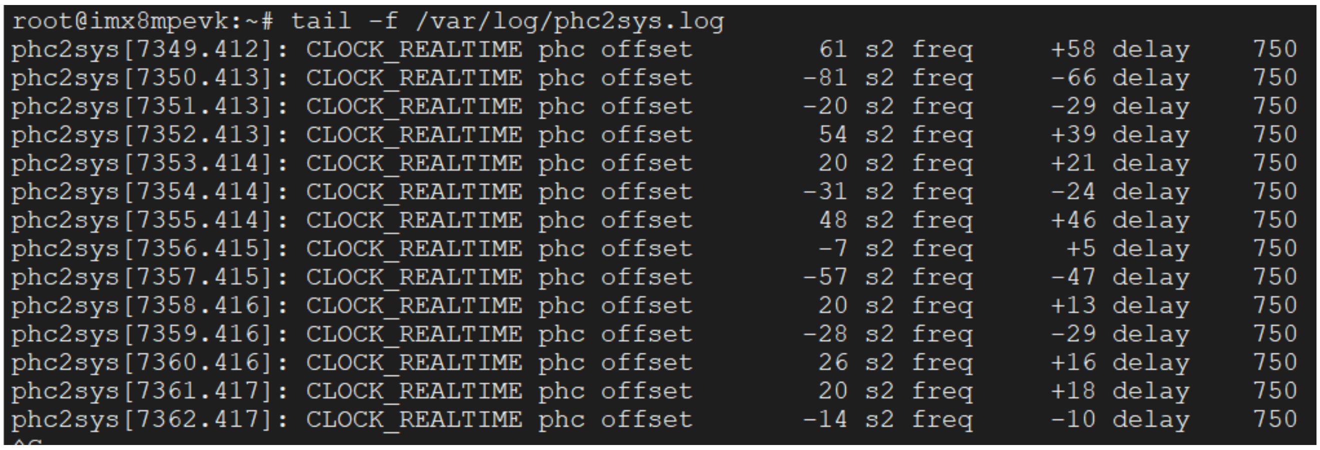
On the Subscriber, the rms value reported by ptp4l shows the root mean square of the time offset between the PHC and the GM clock. If ptp4l consistently reports rms lower than 100 ns, the PHC is synchronized. See the example ptp4l log below:



```
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 172. A sample ptp4l log

On both the Publisher and the Subscriber, the offset information reported by phc2sys shows the time offset between the PHC and the system clock (CLOCK_REALTIME). If phc2sys consistently reports offset lower than 100 ns, the System clock is synchronized. A sample phc2sys log is shown below:



```
root@imx8mpevk:~# tail -f /var/log/phc2sys.log
phc2sys[7349.412]: CLOCK_REALTIME phc offset        61 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
```

Figure 173. 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.MX 8M Plus LPDDR4 EVK - Board A), configure TSN Qbv on ENET2 (that is, eth1) to map SKB priority to traffic class to hardware queue as below, set gate control list to have 2 entries and total cycle time of 1ms (queue 4 has 500 µs for best effort traffic, queue 0 and queue 2 share 500us for OPC UA PubSub and PTP traffic as well as other traffic like ping), also set base time to 1ms so that the schedule is aligned to 1ms. This is just an example configuration for the schedule.

  ```
  SKB priority 0 -> traffic class 0 -> queue 0
  SKB priority 1 -> traffic class 1 -> queue 1
  SKB priority 2 -> traffic class 2 -> queue 2
  SKB priority 3 -> traffic class 3 -> queue 3
  SKB priority 4 -> traffic class 4 -> queue 4
  ```

  ```
  # tc qdisc replace dev eth1 parent root handle 100 taprio num_tc 5 map 0 1 2
   3 4 queues 1@0 1@1 1@2 1@3 1@4 base-time 001000000 sched-entry S 0x10 500000
   sched-entry S 0x05 500000 flags 2
  # tc -g qdisc show dev eth1
  ```

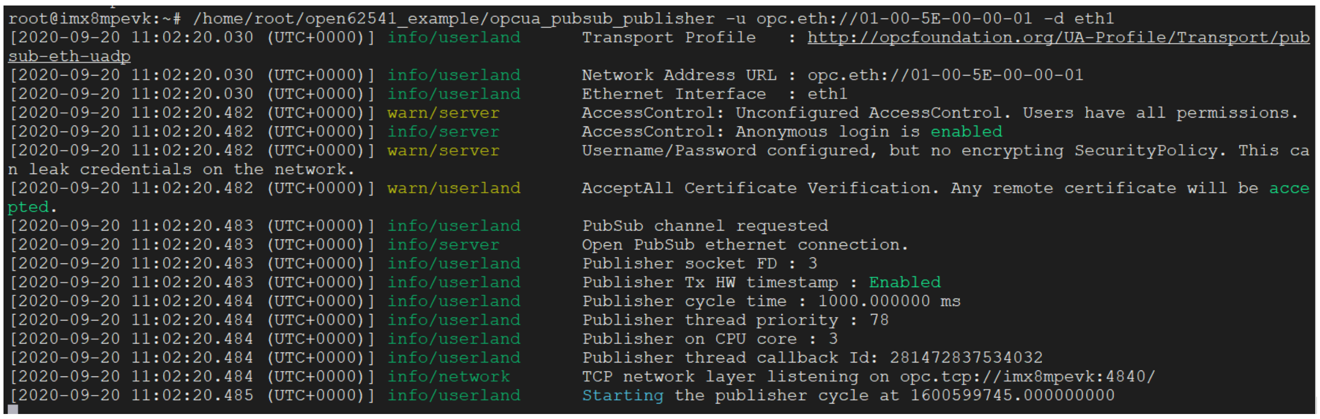
  Together with the tc filter rule configured previously, the above TSN qbv configuration on ENET2 distributes OPC UA PubSub traffic into TX hardware queue 2, PTP traffic into TX hardware queue 0. Also, we send best effort traffic to TX hardware queue 4. Other traffic such as pings can still go into TX 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.MX 8M 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 must be separated by **hyphens (-)**.

  ```
  # /root/open62541_example/opcua_pubsub_subscriber -u
   opc.eth://01-00-5E-00-00-01 -d eth1
  ```

- On the Publisher (i.MX 8M Plus LPDDR4 EVK - Board A), run the OPC UA PubSub Publisher sample application.
  Note that octets in the MAC address must be separated by **hyphens (-)**.

  ```
  # /root/open62541_example/opcua_pubsub_publisher -u opc.eth://01-00-5E-00-00-01
   -d eth1
  ```

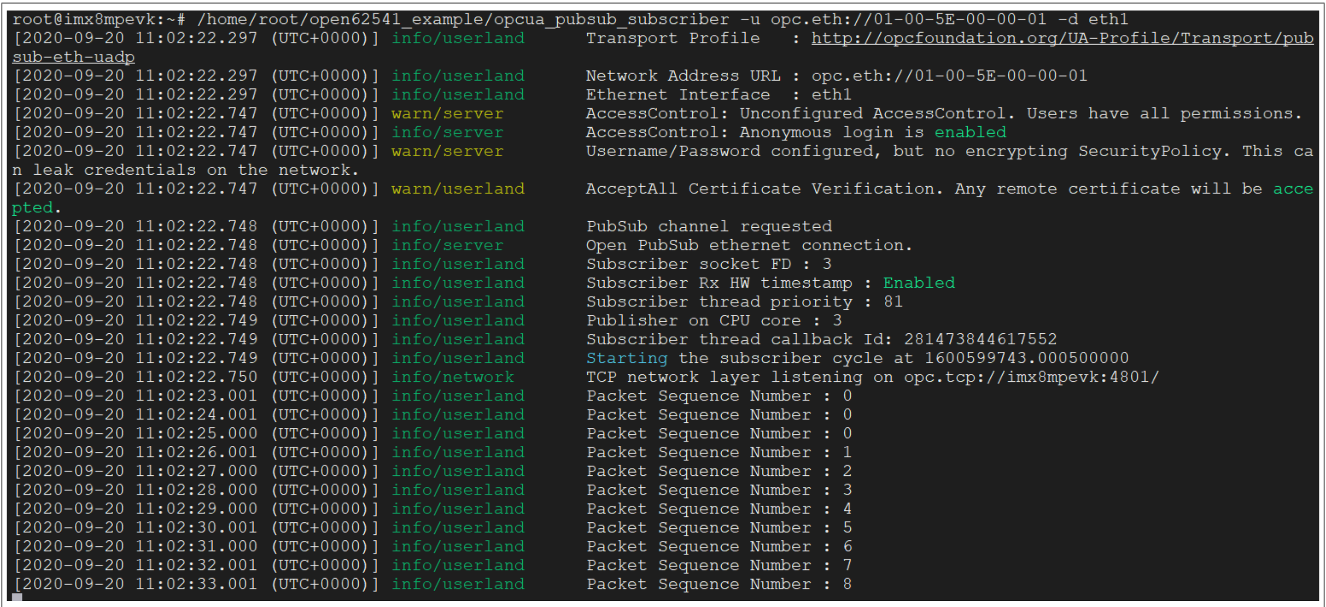
  Example log on the Publisher:



```
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/pub
sub-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 ca
n leak credentials on the network.
[2020-09-20 11:02:20.482 (UTC+0000)] warn/userland      AcceptAll Certificate Verification. Any remote certificate will be acce
pted.
[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 174. Example log on the Publisher

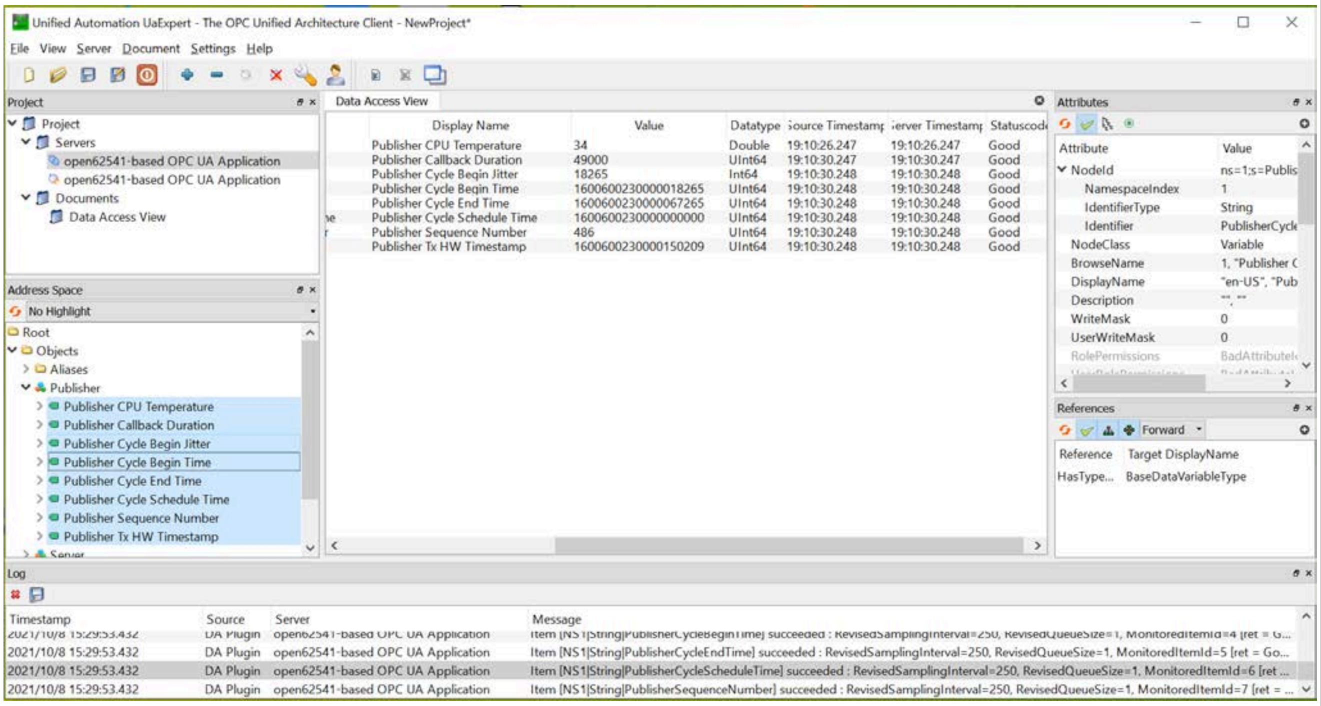
Example log on the Subscriber:



```
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/pub
sub-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 ca
n leak credentials on the network.
[2020-09-20 11:02:22.747 (UTC+0000)] warn/userland      AcceptAll Certificate Verification. Any remote certificate will be acce
pted.
[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 175. 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:



Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*

Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
Publisher CPU Temperature	34	Double	19:10:26.247	19:10:26.247	Good
Publisher Callback Duration	49000	UInt64	19:10:30.247	19:10:30.247	Good
Publisher Cycle Begin Jitter	18265	Int64	19:10:30.248	19:10:30.248	Good
Publisher Cycle Begin Time	1600600230000018265	UInt64	19:10:30.248	19:10:30.248	Good
Publisher Cycle End Time	1600600230000067265	UInt64	19:10:30.248	19:10:30.248	Good
Publisher Cycle Schedule Time	1600600230000000000	UInt64	19:10:30.248	19:10:30.248	Good
Publisher Sequence Number	486	UInt64	19:10:30.248	19:10:30.248	Good
Publisher Tx HW Timestamp	1600600230000150209	UInt64	19:10:30.248	19:10:30.248	Good

Attribute	Value
NodeId	ns=1;s=Publis
NamespaceIndex	1
IdentifierType	String
Identifier	PublisherCycl
NodeClass	Variable
BrowseName	1, "Publisher C
DisplayName	"en-US", "Pub
Description	"", ""
WriteMask	0
UserWriteMask	0
RolePermissions	BadAttributeI

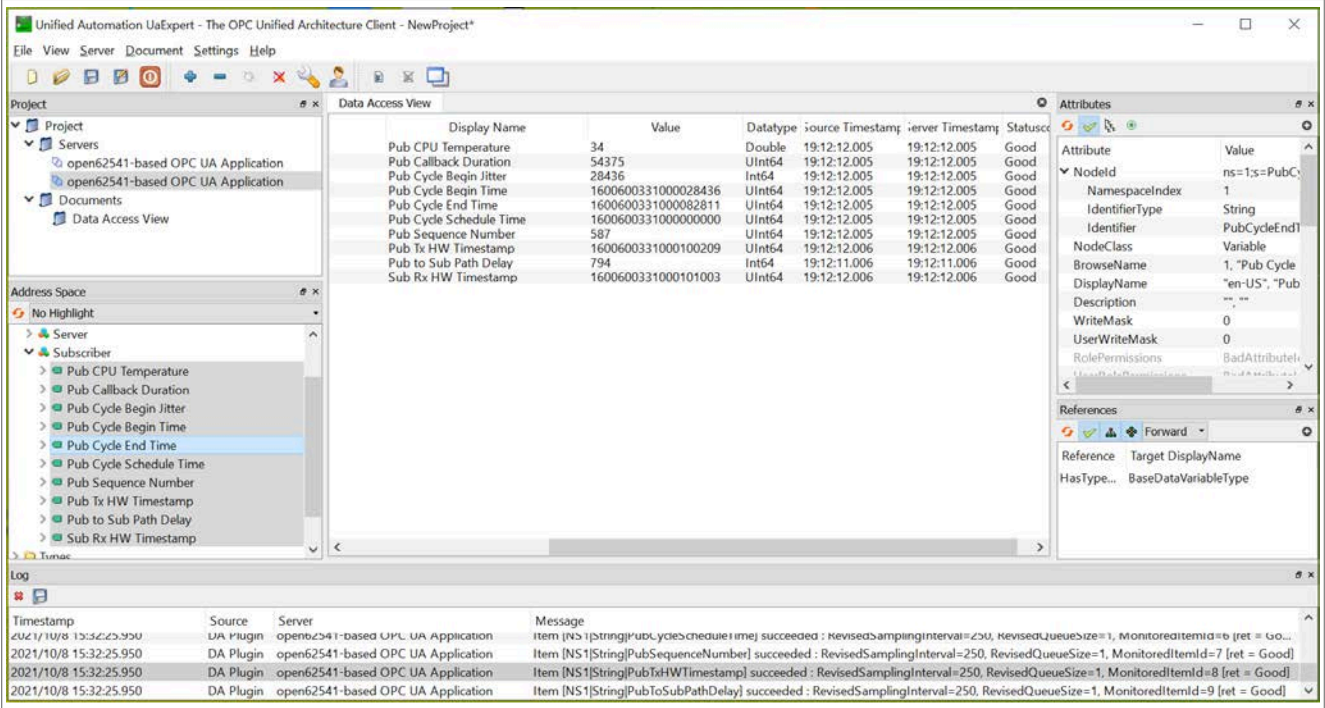
Reference	Target DisplayName
HasType...	BaseDataVariableType

Timestamp	Source	Server	Message
2021/10/8 15:29:53.432	DA Plugin	open62541-based OPC UA Application	Item [NS1|String|PublisherCycleBeginTime] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=4 [ret = G...
2021/10/8 15:29:53.432	DA Plugin	open62541-based OPC UA Application	Item [NS1|String|PublisherCycleEndTime] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=5 [ret = Go...
2021/10/8 15:29:53.432	DA Plugin	open62541-based OPC UA Application	Item [NS1|String|PublisherCycleScheduleTime] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=6 [ret ...
2021/10/8 15:29:53.432	DA Plugin	open62541-based OPC UA Application	Item [NS1|String|PublisherSequenceNumber] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=7 [ret = ...

File View Server Document Settings Help
Project
Project
Servers
open62541-based OPC UA Application
open62541-based OPC UA Application
Documents
Data Access View
Address Space
No Highlight
Root
Objects
Aliases
Publisher
Publisher CPU Temperature
Publisher Callback Duration
Publisher Cycle Begin Jitter
Publisher Cycle Begin Time
Publisher Cycle End Time
Publisher Cycle Schedule Time
Publisher Sequence Number
Publisher Tx HW Timestamp
Server
Data Access View
Attributes
References
Forward
Log

Figure 176. Sample snapshot of UaExpert connected to the Publisher

Example snapshot of UaExpert connected to the Subscriber:



Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
Pub CPU Temperature	34	Double	19:12:12.005	19:12:12.005	Good
Pub Callback Duration	54375	UInt64	19:12:12.005	19:12:12.005	Good
Pub Cycle Begin Jitter	28436	Int64	19:12:12.005	19:12:12.005	Good
Pub Cycle Begin Time	1600600331000028436	UInt64	19:12:12.005	19:12:12.005	Good
Pub Cycle End Time	1600600331000082811	UInt64	19:12:12.005	19:12:12.005	Good
Pub Cycle Schedule Time	1600600331000000000	UInt64	19:12:12.005	19:12:12.005	Good
Pub Sequence Number	587	UInt64	19:12:12.005	19:12:12.005	Good
Pub Tx HW Timestamp	1600600331000100209	UInt64	19:12:12.006	19:12:12.006	Good
Pub to Sub Path Delay	794	Int64	19:12:11.006	19:12:11.006	Good
Sub Rx HW Timestamp	1600600331000101003	UInt64	19:12:12.006	19:12:12.006	Good

```
Timestamp              Source     Server                              Message
2021/10/8 15:32:25.950 DA Plugin  open62541-based OPC UA Application  Item [NS1|String|PubCycleScheduleTime] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=6 [ret = Go...
2021/10/8 15:32:25.950 DA Plugin  open62541-based OPC UA Application  Item [NS1|String|PubSequenceNumber] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=7 [ret = Good]
2021/10/8 15:32:25.950 DA Plugin  open62541-based OPC UA Application  Item [NS1|String|PubTxHWTimestamp] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=8 [ret = Good]
2021/10/8 15:32:25.950 DA Plugin  open62541-based OPC UA Application  Item [NS1|String|PubToSubPathDelay] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=9 [ret = Good]
```

Figure 177. 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.MX 8M Plus LPDDR4 EVK - Board A), we can use pktgen to simulate high load best effort traffic which is sent to queue 4 of ENET2.

```
# /usr/share/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.

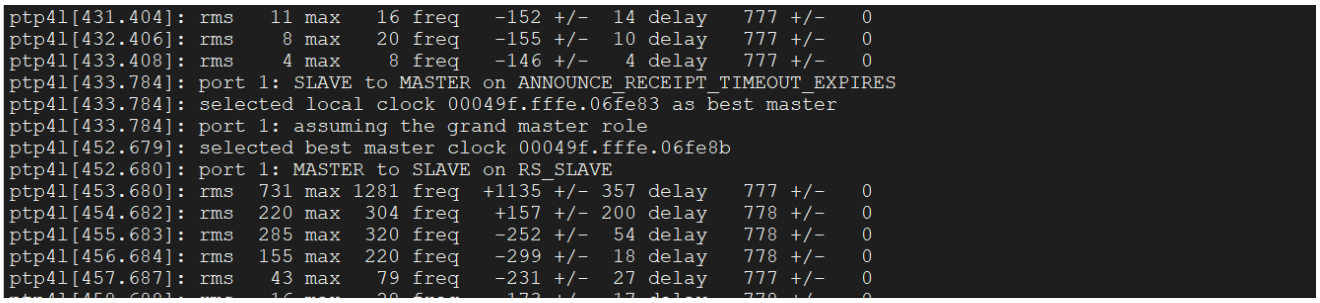
In case TSN Qbv was not configured, after pktgen starts running, various issues might happen. First of all, the ptp4l application hows timeout issue as below.

Example 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
```

Figure 178. Sample error log of ptp4l on the Publisher

Example 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
```

Figure 179. Sample error log of ptp4l on the Subscriber

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: *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 might display a warning message timed out while polling for tx timestamp!..

– The Subscriber application might show that the packet sequence number stops incrementing.

– The path delay from Publisher to Subscriber displayed on UaExpert might show a large number due to packet transmission delay.

When such issues happen, they can only be recovered by restarting both the Publisher and Subscriber applications.

6.3.7.5.2 Case #2: two i.MX 8M Plus LPDDR4 EVK boards connected to LS1028ARDB TSN switch

The setup could use one LS1028ARDB as TSN switch plus two or three i.MX 8M Plus LPDDR4 EVK boards. One i.MX 8M Plus LPDDR4 EVK (Board A) acts as Publisher and others act as Subscribers. Figure 180 shows the block diagram of this setup. The ENET2 interface on each i.MX 8M Plus LPDDR4 EVK is connected to the switch port on LS1028ARDB. Also, the ENET1 interface on each i.MX 8M Plus LPDDR4 EVK is connected to LAN (that is, office network).

Note: *The actual device name might be different in Linux.*

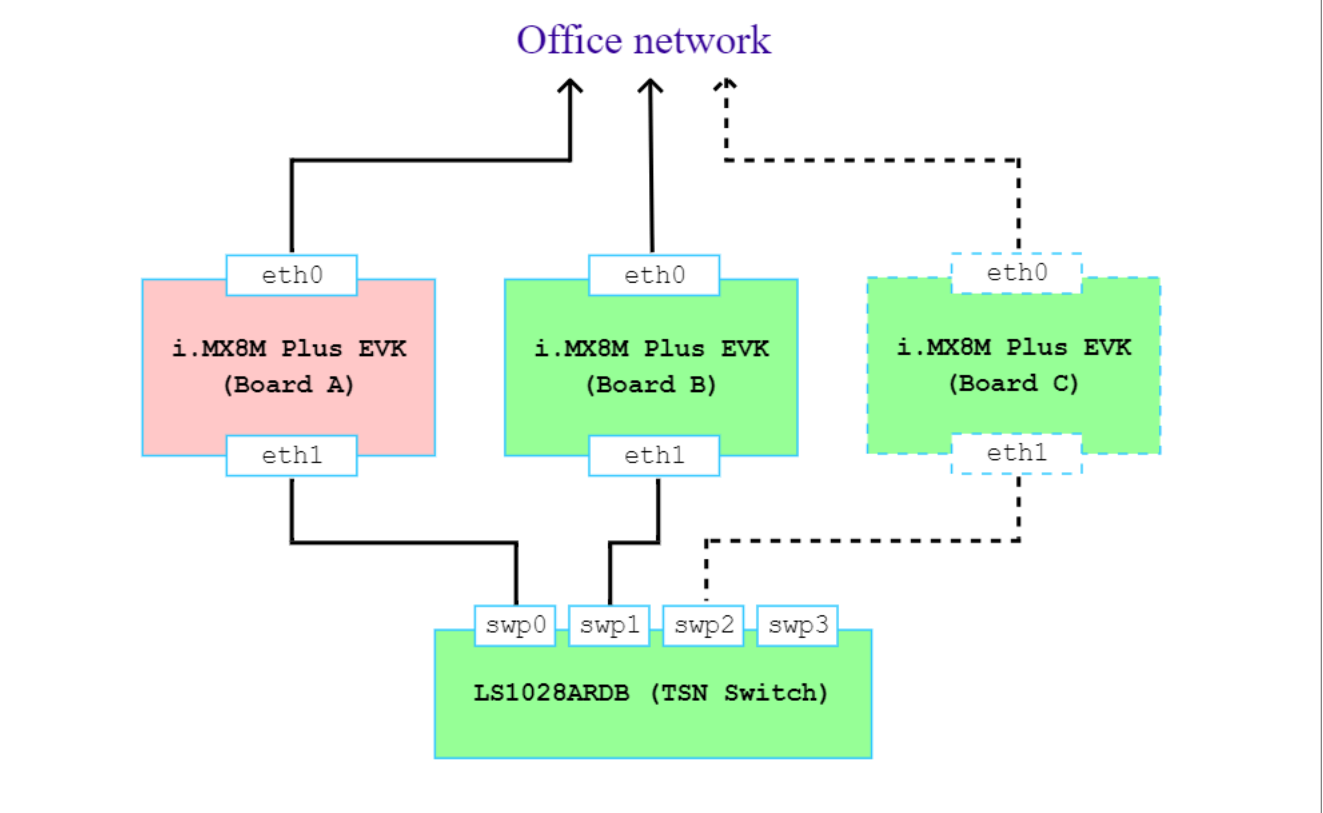


Figure 180. Two i.MX 8M Plus LPDDR4 EVK boards connected to LS1028ARDB TSN switch

The following steps assume that two i.MX 8M Plus LPDDR4 EVK boards are used. Board A acts as Publisher and Board B acts as Subscriber. In this setup, the switch port `swp0` is the ingress port for OPC UA PubSub traffic and best effort traffic. The switch port `swp1` is the egress ports for OPC UA PubSub traffic and best effort traffic.

Since the TSN switch on LS1028ARDB uses the value of VLAN PCP field to map traffic to different TX hardware queue on egress switch port (that is, `swp1`), we add VLAN header to the OPC UA PubSub packet and best effort packet. Note that the PTP packet is untagged without VLAN header.

- On LS1028ARDB, configure Ethernet bridge on TSN switch and enable VLAN filtering.

```
# ip link set eno2 up
# ip link set swp0 up
# ip link set swp1 up
# ip link add name br0 type bridge vlan_filtering 1
# ip link set dev swp0 master br0
# ip link set dev swp1 master br0
# ip link set dev br0 up
# bridge vlan add dev swp0 vid 100
# bridge vlan add dev swp1 vid 100
# bridge vlan show
```

- On both the Publisher (i.MX 8M Plus LPDDR4 EVK - Board A) and the Subscriber (i.MX 8M Plus LPDDR4 EVK - Board B), bring up ENET2 (that is, eth1):

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

- On the Publisher (i.MX 8M Plus LPDDR4 EVK - Board A), add one tc filter to match OPC UA PubSub packet (EtherType 0xb62c) on ENET2 (that is, eth1) and modify SKB priority to 2.

```
# 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.MX 8M 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.MX 8M 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.MX 8M 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, observe the logs of ptp4l and phc2sys to check the time synchronization progress by using the below commands: (To continue to execute other commands on the serial console, performing an SSH to each board to check the logs of ptp4l and phc2sys is recommended).

```
# tail -f /var/log/ptp4l.log
# tail -f /var/log/phc2sys.log
```

  On LS1028ARDB and the Subscriber, the rms value reported by ptp4l shows the root mean square of the time offset between the PHC and the GM clock. If ptp4l consistently reports rms lower than 100 ns, the PHC is synchronized. Refer to the example log of ptp4l in the back-to-back case.
  On each board, the offset information reported by phc2sys shows the time offset between the PHC and the system clock (CLOCK_REALTIME). If phc2sys consistently reports offset lower than 100 ns, the System clock is synchronized. Refer to the example log of phc2sys in the back-to-back case.
  After establishing the time synchronization successfully on each board, users can configure TSN Qbv and run the OPC UA PubSub sample applications using the following steps.
- On the Publisher (i.MX 8M Plus LPDDR4 EVK - Board A), configure TSN Qbv on 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 queue 2 share 500 µs for OPC UA PubSub and PTP traffic as well as other traffic like ping), also set base time to 1 ms so that the schedule is aligned to 1 ms. This is just an example configuration for the schedule.

```
 SKB priority 0 -> traffic class 0 -> queue 0
 SKB priority 1 -> traffic class 1 -> queue 1
 SKB priority 2 -> traffic class 2 -> queue 2
 SKB priority 3 -> traffic class 3 -> queue 3
 SKB priority 4 -> traffic class 4 -> queue 4
```

```
# tc qdisc replace dev eth1 parent root handle 100 taprio num_tc 5 map 0 1 2
 3 4 queues 1@0 1@1 1@2 1@3 1@4 base-time 001000000 sched-entry S 0x10 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. Ensure that queue 4 has 500 µs for best effort traffic, queue 0 and queue 2 share 500 µs for OPC UA PubSub and PTP traffic as well as other traffic such as ping. In addition, set the base time to 1 ms so that the schedule is aligned to 1 ms as 1 ms. Note that the TSN Qbv configuration on LS1028ARDB TSN switch is used to protect the OPC UA PubSub traffic from traffic which may enter the switch from other switch ports. In this use case, it is optional and is used for demonstration purpose only.

```
# tc qdisc replace dev swp1 root taprio num_tc 8 map 0 1 2 3 4 5 6 7 queues
 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 base-time 001000000 sched-entry S 0x10 500000
 sched-entry S 0x05 500000 flags 0x2
# tc -g qdisc show dev swp1
```

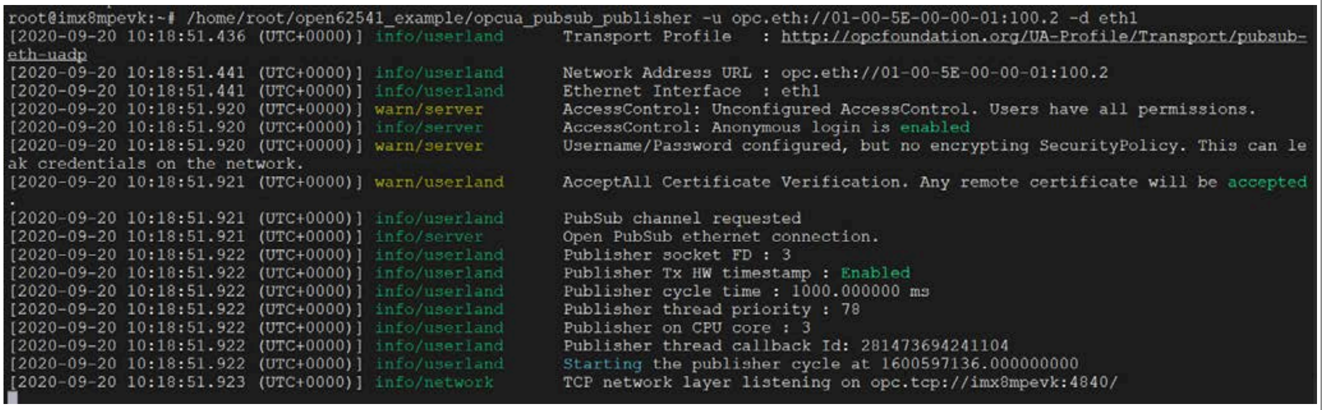
- With the above TSN Qbv configuration on egress switch port swp1,
  – OPC UA PubSub traffic goes into TX hardware queue 2 (Add VLAN header with PCP field set to 2 for OPC UA PubSub packet).
  – The best effort traffic goes into TX hardware queue 4 (Add VLAN header with PCP field set to 4 using pktgen for generating best effort traffic). Note that the PTP traffic is untagged without VLAN header and uses 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.MX 8M Plus LPDDR4 EVK - Board B), run the OPC UA PubSub Subscriber sample application. Run the Subscriber application before the Publisher application so that no packet sent by the Publisher is missed.
  Note that in the URL of below command 100.2 means VLAN ID 100 and PCP value 2 and it is separated from the MAC address using a colon.

```
# /root/open62541_example/opcua_pubsub_subscriber -u
 opc.eth://01-00-5E-00-00-01:100.2 -d eth1
```

- On the Publisher (i.MX 8M 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.

```
# /root/open62541_example/opcua_pubsub_publisher -u
 opc.eth://01-00-5E-00-00-01:100.2 -d eth1
```

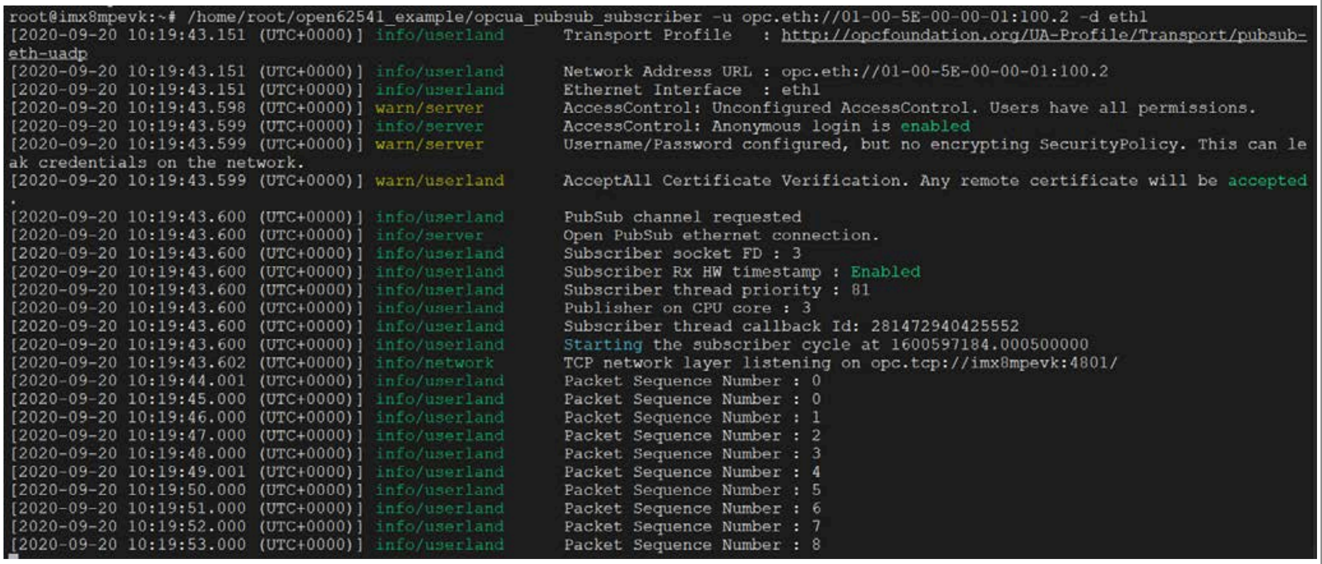
Example log on the Publisher:



```
root@imx8mpevk:~# /home/root/open62541_example/opcua_pubsub_publisher -u opc.eth://01-00-5E-00-00-01:100.2 -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/
```

Figure 181. 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-00-00-01:100.2 -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 182. 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.MX 8M 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 /usr/share/samples/pktgen/pktgen_sample01_simple.sh /usr/share/samples/
pktgen/pktgen_sample01_simple_vlan.sh
# sed -i '/^UDP_MAX=.*/a VLAN_ID=100\nVLAN_P=4' /usr/share/samples/pktgen/
pktgen_sample01_simple_vlan.sh
```

```
# /usr/share/samples/pktgen/pktgen_sample01_simple_vlan.sh -i eth1 -q 4 -s 1000
 -n 0
```

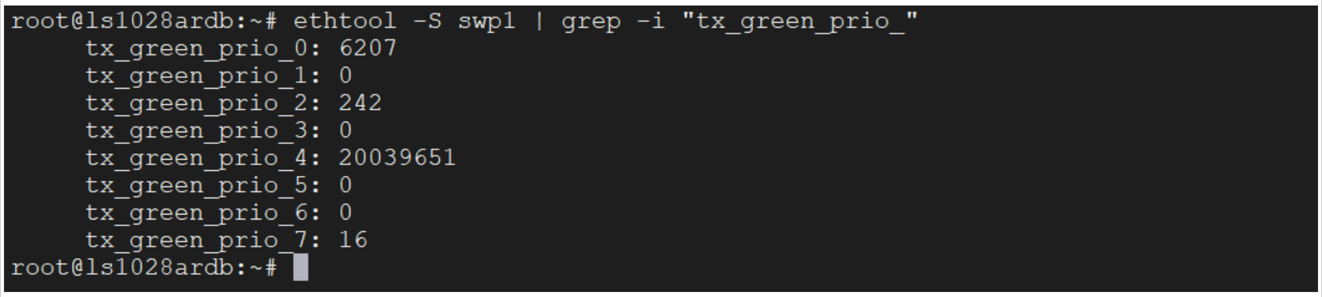
Note: *In order to protect the OPC UA PubSub traffic and PTP traffic, the TSN Qbv must be configured to have different TX hardware queue and time slot from the best effort traffic on both the Publisher and TSN switch. This ensures that users can get consistent output on the console of the Publisher and the Subscriber, and the path delay from Publisher to Subscriber is still around 4 µs.*

In case TSN Qbv was not configured, after pktgen starts running, various issues might occur. Refer to the issues detailed in the back-to-back case.

- On LS1028ARDB, it is possible to check the status of TX packets of swp1 by using the command below:

```
# ethtool -S swp1 | grep -i "tx_green_prio_"
```

Example log below: (`tx_green_prio_0` mainly for PTP traffic, `tx_green_prio_2` mainly for OPC UA PubSub traffic, `tx_green_prio_4` mainly for best effort traffic generated by pktgen).



```
root@ls1028ardb:~# ethtool -S swp1 | grep -i "tx_green_prio_"
     tx_green_prio_0: 6207
     tx_green_prio_1: 0
     tx_green_prio_2: 242
     tx_green_prio_3: 0
     tx_green_prio_4: 20039651
     tx_green_prio_5: 0
     tx_green_prio_6: 0
     tx_green_prio_7: 16
root@ls1028ardb:~#
```

Figure 183. Sample log after checking the stats of TX packets of swp1

6.3.7.6 OPC UA client installation and usage

6.3.7.6.1 UaExpert

The UaExpert is an OPC UA Client developed by Unified Automation. It is free to download. Before downloading, you must 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

The below steps show 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.

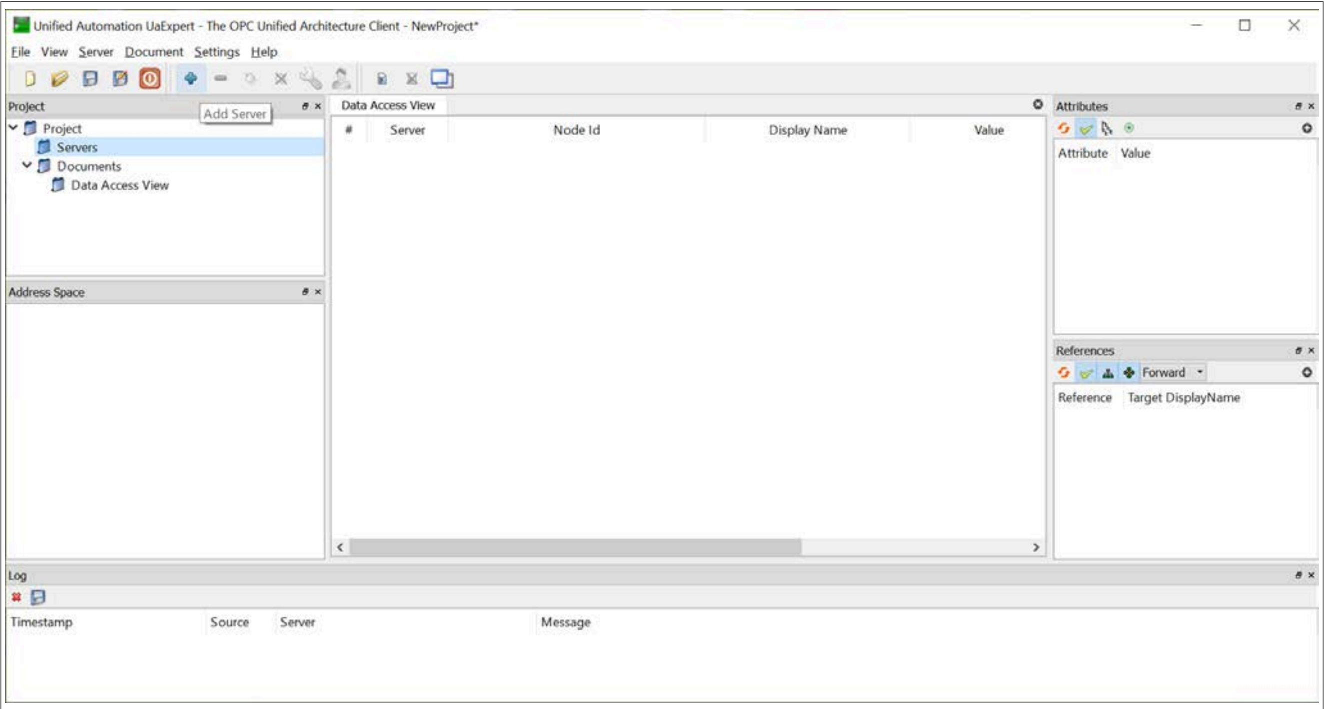
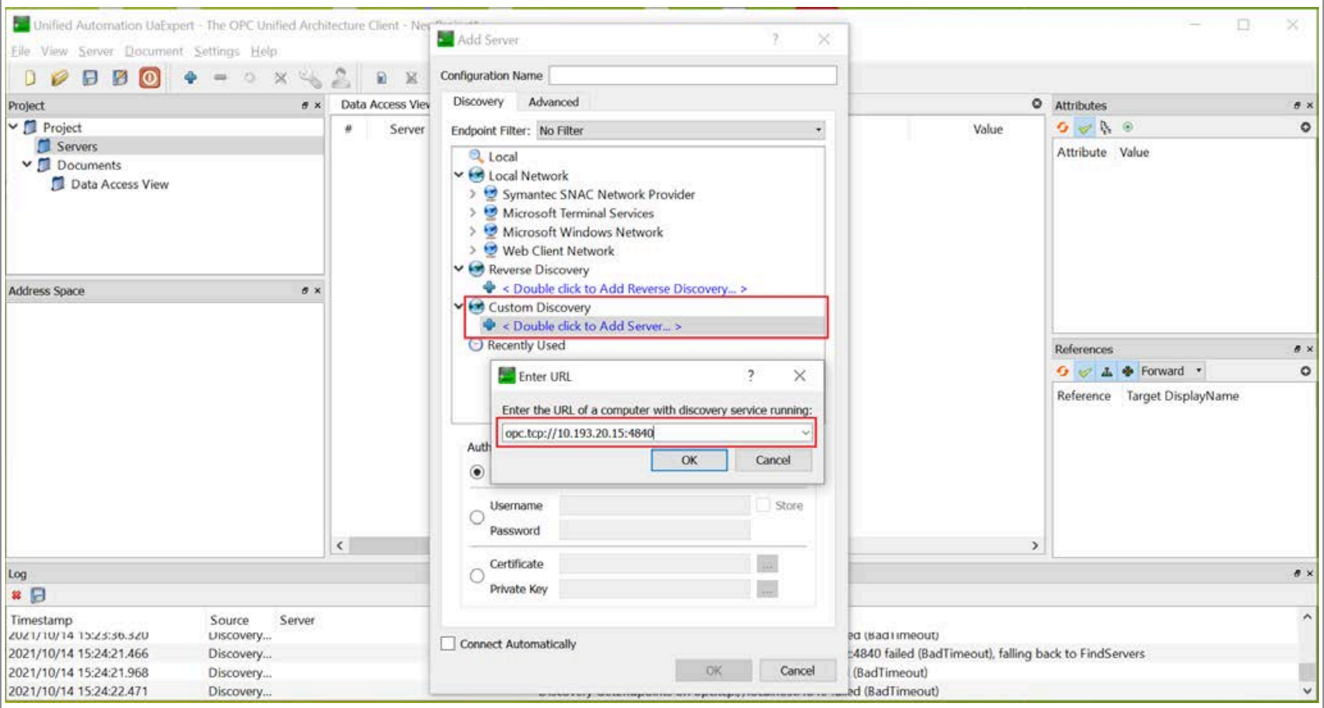

Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*
File View Server Document Settings Help
Add Server
Project
Project
Servers
Documents
Data Access View
Data Access View
# Server Node Id Display Name Value
Attributes
Attribute Value
References
Forward
Reference Target DisplayName
Address Space
Log
Timestamp Source Server Message

Figure 184. UaExpert GUI

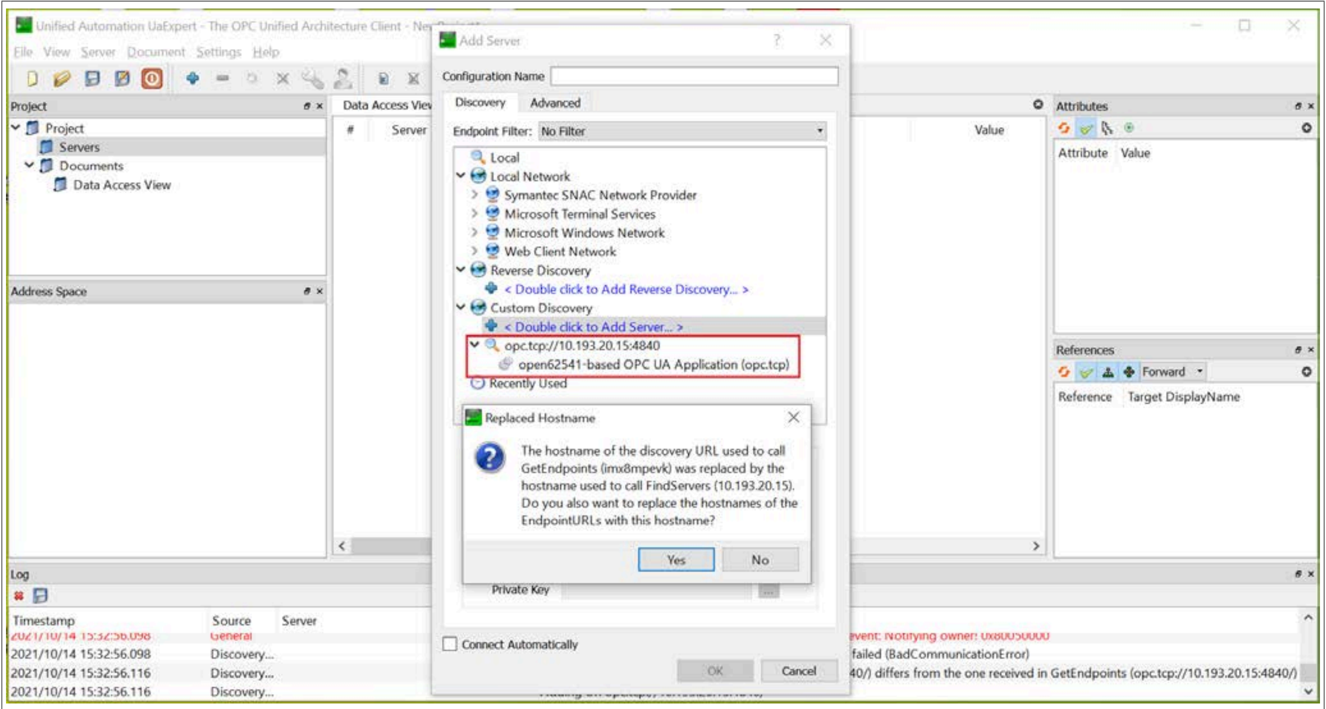
- The '**Add Server**' window pops up. Select Custom Discovery and double click '< ***Double click to Add Server...*** >'. The '**Enter URL**' window pops up. Input IP address and port number of the OPC UA server separated by colon. For example, the complete URL is `opc.tcp://10.193.20.15:4840` in below snapshot. Click **OK**.


Add Server
Configuration Name
Discovery
Advanced
Endpoint Filter: No Filter
Local
Local Network
Symantec SNAC Network Provider
Microsoft Terminal Services
Microsoft Windows Network
Web Client Network
Reverse Discovery
< Double click to Add Reverse Discovery... >
Custom Discovery
< Double click to Add Server... >
Recently Used
Enter URL
Enter the URL of a computer with discovery service running:
opc.tcp://10.193.20.15:4840
OK
Cancel
Username
Password
Store
Certificate
Private Key
Connect Automatically
OK
Cancel

```
Timestamp               Source      Server
2021/10/14 15:23:36.320 Discovery...
2021/10/14 15:24:21.466 Discovery...        4840 failed (BadTimeout), falling back to FindServers
2021/10/14 15:24:21.968 Discovery...        (BadTimeout)
2021/10/14 15:24:22.471 Discovery...        ed (BadTimeout)
```

Figure 185. Adding the OPC UA server

- The new server (that is, *opc.tcp://10.193.20.15:4840*) is listed under Custom Discovery. Click to expand it. Then click to expand '**`open62541-based OPC UA Application (opc.tcp)`**'. A '**Replaced Hostname**' window pops up. Click '**Yes**'.

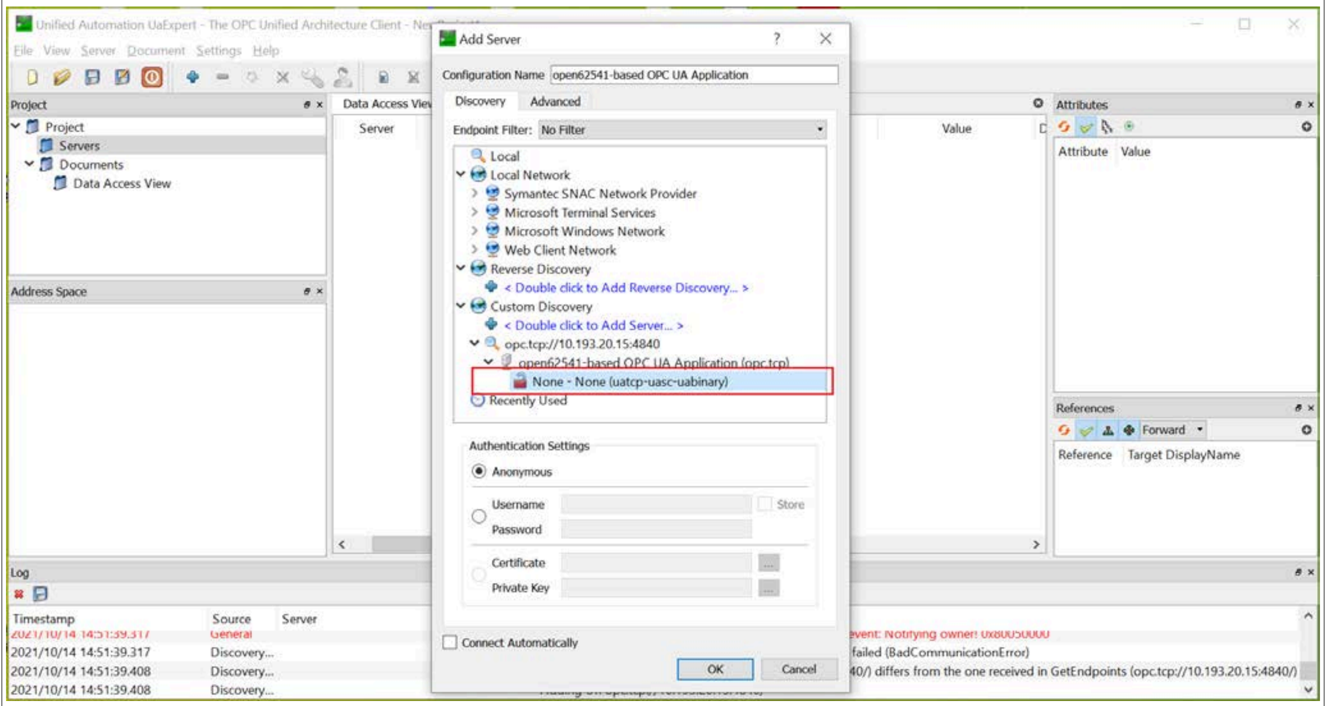


Replaced Hostname

The hostname of the discovery URL used to call GetEndpoints (imx8mpevk) was replaced by the hostname used to call FindServers (10.193.20.15). Do you also want to replace the hostnames of the EndpointURLs with this hostname?

Figure 186. Server listed under Custom Discovery

- Click to select '**None – None (…)**' and click **OK**.

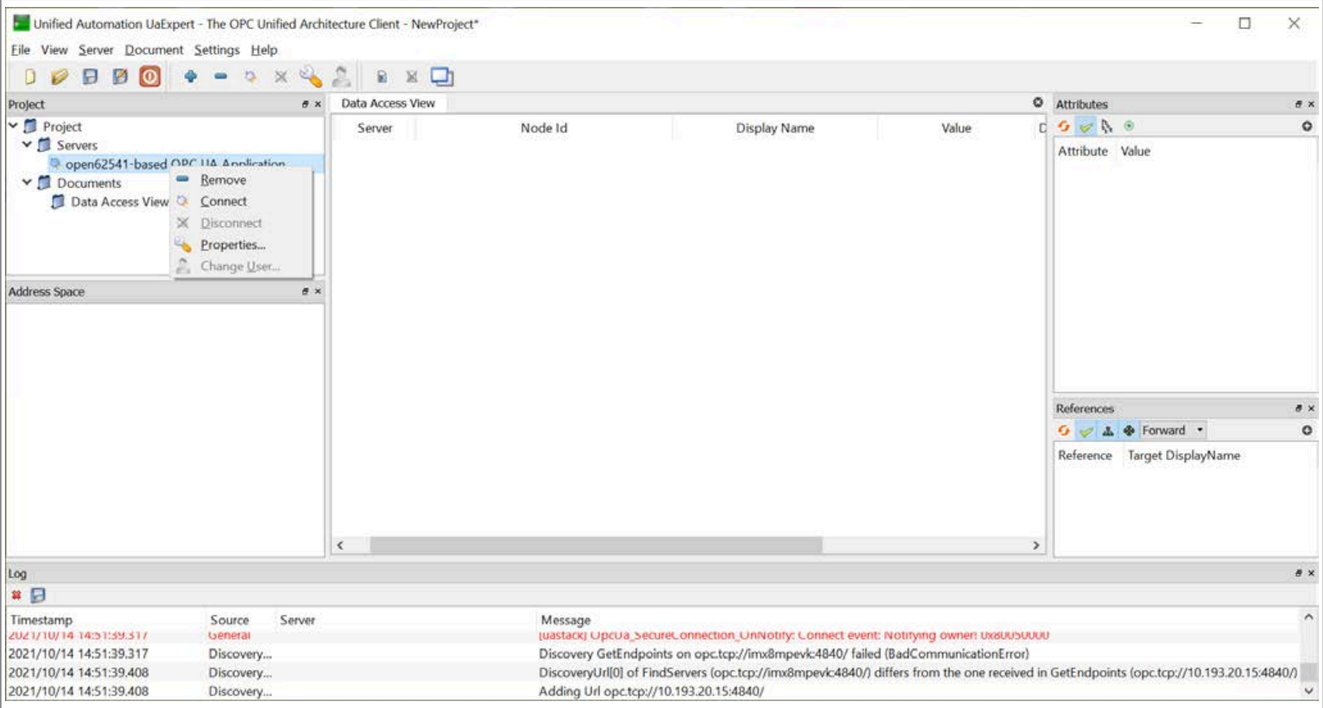


Configuration Name: open62541-based OPC UA Application

None - None (uatcp-uasc-uabinary)

Figure 187. Selecting the hostname

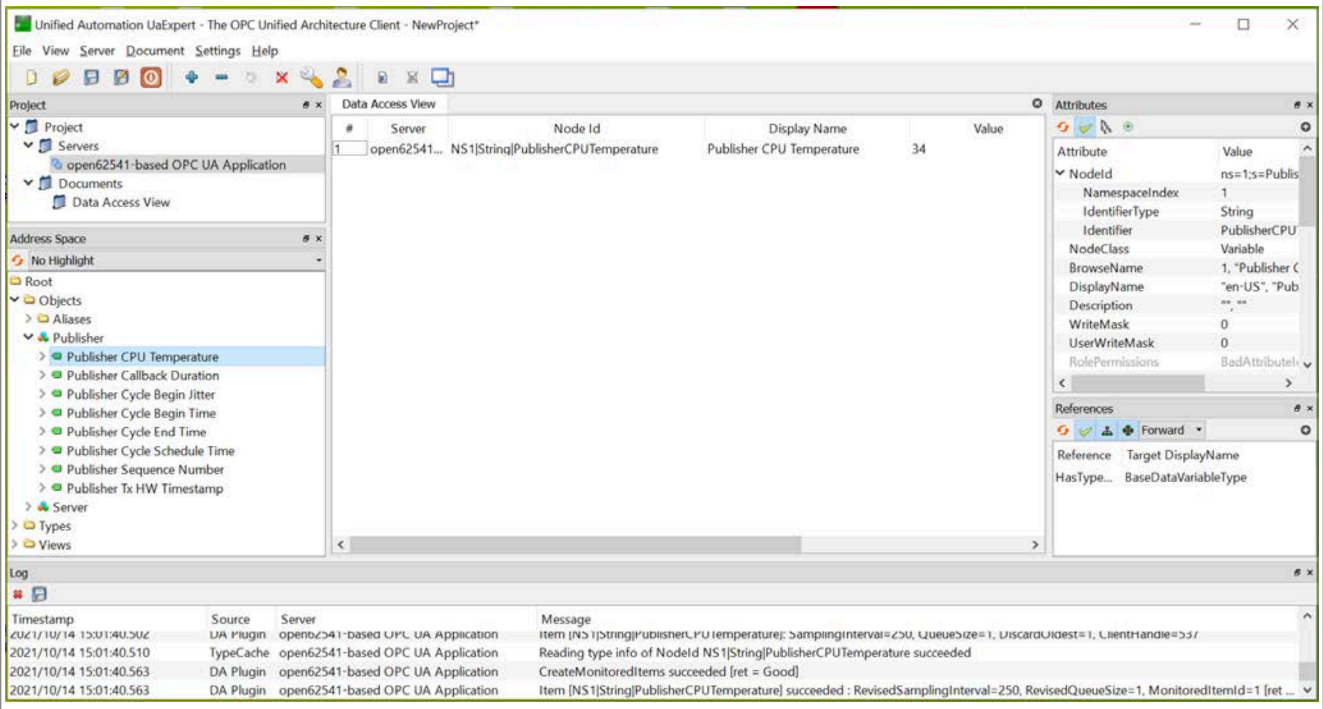
- Right click on the server listed under ‘Servers’ and click ‘Connect’.



```
Timestamp                Source       Server   Message
2021/10/14 14:51:39.317  General               [uastack] OpcUa_SecureConnection_OnNotify: Connect event: Notifying owner! 0x80050000
2021/10/14 14:51:39.317  Discovery...          Discovery GetEndpoints on opc.tcp://imx8mpevk:4840/ failed (BadCommunicationError)
2021/10/14 14:51:39.408  Discovery...          DiscoveryUrl[0] of FindServers (opc.tcp://imx8mpevk:4840/) differs from the one received in GetEndpoints (opc.tcp://10.193.20.15:4840/)
2021/10/14 14:51:39.408  Discovery...          Adding Url opc.tcp://10.193.20.15:4840/
```

Figure 188. 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.


Unified Automation UaExpert - The OPC Unified Architecture Client - NewProject*
File View Server Document Settings Help
Project
Project
Servers
open62541-based OPC UA Application
Documents
Data Access View
Address Space
No Highlight
Root
Objects
Aliases
Publisher
Publisher CPU Temperature
Publisher Callback Duration
Publisher Cycle Begin Jitter
Publisher Cycle Begin Time
Publisher Cycle End Time
Publisher Cycle Schedule Time
Publisher Sequence Number
Publisher Tx HW Timestamp
Server
Types
Views
Data Access View
Attributes
References
Forward
Log

#	Server	Node Id	Display Name	Value
1	open62541...	NS1|String|PublisherCPUTemperature	Publisher CPU Temperature	34

Attribute	Value
NodeId	ns=1;s=Publis
NamespaceIndex	1
IdentifierType	String
Identifier	PublisherCPU
NodeClass	Variable
BrowseName	1, "Publisher C
DisplayName	"en-US", "Pub
Description	"", ""
WriteMask	0
UserWriteMask	0
RolePermissions	BadAttributeI

Reference	Target DisplayName
HasType...	BaseDataVariableType

```
Timestamp                Source     Server                             Message
2021/10/14 15:01:40.502  DA Plugin  open62541-based OPC UA Application  Item [NS1|String|PublisherCPUTemperature]: SamplingInterval=250, QueueSize=1, DiscardOldest=1, ClientHandle=537
2021/10/14 15:01:40.510  TypeCache  open62541-based OPC UA Application  Reading type info of NodeId NS1|String|PublisherCPUTemperature succeeded
2021/10/14 15:01:40.563  DA Plugin  open62541-based OPC UA Application  CreateMonitoredItems succeeded [ret = Good]
2021/10/14 15:01:40.563  DA Plugin  open62541-based OPC UA Application  Item [NS1|String|PublisherCPUTemperature] succeeded : RevisedSamplingInterval=250, RevisedQueueSize=1, MonitoredItemId=1 [ret ...
```

Figure 189. Data Access View to monitor OPC UA objects

6.3.7.6.2 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:

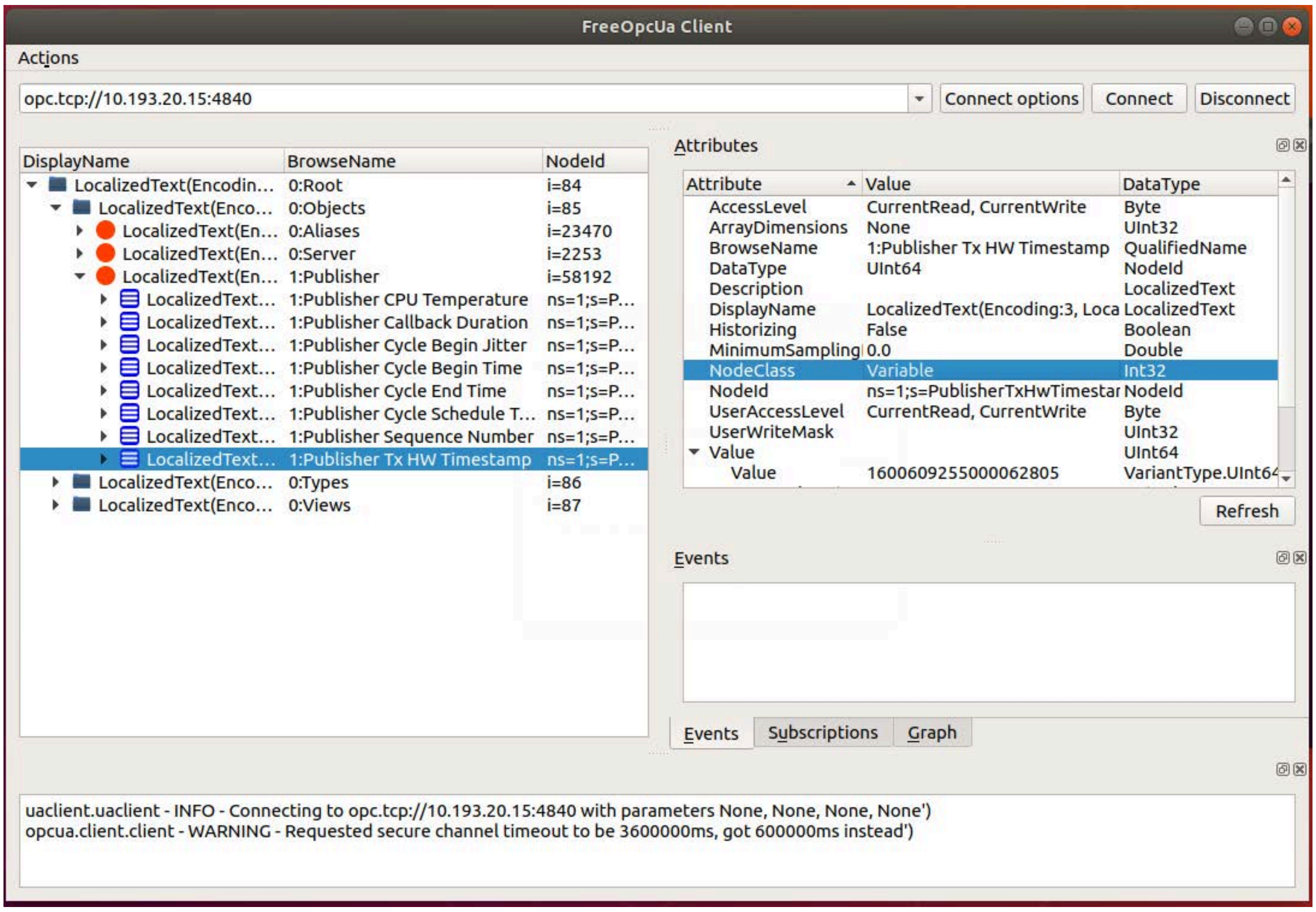
https://github.com/FreeOpcUa/opcua-client-gui

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.



DisplayName	BrowseName	NodeId
LocalizedText(Encodin...	0:Root	i=84
LocalizedText(Enco...	0:Objects	i=85
LocalizedText(En...	0:Aliases	i=23470
LocalizedText(En...	0:Server	i=2253
LocalizedText(En...	1:Publisher	i=58192
LocalizedText...	1:Publisher CPU Temperature	ns=1;s=P...
LocalizedText...	1:Publisher Callback Duration	ns=1;s=P...
LocalizedText...	1:Publisher Cycle Begin Jitter	ns=1;s=P...
LocalizedText...	1:Publisher Cycle Begin Time	ns=1;s=P...
LocalizedText...	1:Publisher Cycle End Time	ns=1;s=P...
LocalizedText...	1:Publisher Cycle Schedule T...	ns=1;s=P...
LocalizedText...	1:Publisher Sequence Number	ns=1;s=P...
LocalizedText...	1:Publisher Tx HW Timestamp	ns=1;s=P...
LocalizedText(Enco...	0:Types	i=86
LocalizedText(Enco...	0:Views	i=87

Attribute	Value	DataType
AccessLevel	CurrentRead, CurrentWrite	Byte
ArrayDimensions	None	UInt32
BrowseName	1:Publisher Tx HW Timestamp	QualifiedName
DataType	UInt64	NodeId
Description		LocalizedText
DisplayName	LocalizedText(Encoding:3, Loca	LocalizedText
Historizing	False	Boolean
MinimumSampling	0.0	Double
NodeClass	Variable	Int32
NodeId	ns=1;s=PublisherTxHwTimestar	NodeId
UserAccessLevel	CurrentRead, CurrentWrite	Byte
UserWriteMask		UInt32
Value		UInt64
Value	1600609255000062805	VariantType.UInt64

```
uaclient.uaclient - INFO - Connecting to opc.tcp://10.193.20.15:4840 with parameters None, None, None, None')
opcua.client.client - WARNING - Requested secure channel timeout to be 3600000ms, got 600000ms instead')
```

Figure 190. FreeOpcUa GUI client Connect options

6.3.7.7 OPC UA Frame Summation

OPC UA FX (Field eXchange) is a manufacturer-independent communication protocol, which extends the core OPC UA functionality for field-level communications. It defines connections using OPC UA PubSub for data exchange, including the usage of TSN. Decreasing the network latency and increasing bandwidth usage is essential in such senarios.

The i.MX 943 processor provides the hardware acceleration feature of Frame Modification on its internal TSN switch. This feature allows insertion of one or more blocks of *DataSetMessage* (an OPC UA PubSub data structure) into a OPC UA PubSub *NetworkMessage* (a container for *DataSetMessage*) on the fly when it is passing the switch. It can benefit the frame summation which combines data sets from various device nodes in one Ethernet frame. This feature not only improves bandwidth usage, but also lower the network latency.

To implement this feature, several tables in the switch must be set up by using the NETC Table Management Protocol (NTMP). The procedure involves configuring the tables below:

- Ingress Port Filter Table (ID 13)
- Ingress Stream Table (ID 31)
- Egress Treatment Table (ID 33)
- Frame Modification Table (ID 40)
- Frame Modification Data Table (ID 44)

Combine them together to implement these operations, filtering out the OPC UA UADP frames, inserting *DataSetMessage* into them. These operations are completely done by the hardware.

6.3.7.7.1 OPC UA summation network setup

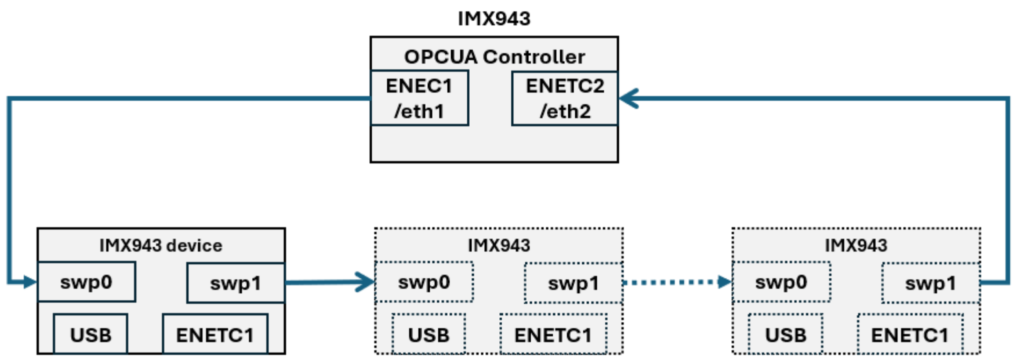


Figure 191. OPC UA summation network setup

In this network, one i.MX943 EVK board works as the OPC UA Controller. The other i.MX943 EVK board works as the OPC UA Device. Connect them as shown in Figure 191. Connect the ENETC1 port of the controller to the swp0 port of the first device, and connect the swp1 port of the first device to the swp0 port of the next device, and so on.

The controller running the application, `opcua_summation_controller`, sends an initial OPC UA frame out periodically on the ENETC1 port. When the frame passes through each device, it inserts one or more DataSet messages. Finally, the frame is received on the ENETC2 port of the controller. These DataSet messages contain data collected on each device. In this way, the controller can quickly collect all data on each device with one OPC UA frame.

The insertion of DataSet messages is done on the fly by the NETC switch of i.MX 943. The application, `opcua_summation_device`, runs on each device. It samples data and packages them in the DataSet messages that are sent to the switch. The switch filters out one OPC UA frame and inserts the stored DataSet messages into it, and forwards it to other specified switch port.

6.3.7.7.2 OPC UA summation software

6.3.7.7.2.1 opcua_summation_controller

The `opcua_summation_controller` program runs on the controller. Its main job is to send the initial OPC UA frame periodically and receive it back and parse the date it contains.

The command path in the filesystem is: `/root/open62541_example/opcua_summation_controller`

```
Usage: /root/open62541_example/opcua_summation_controller <tx interface> <rx
 interface>
  tx interface: interface sending traffic (e.g., eth1)
  rx interface: interface receiving (e.g., eth2)
```

Its source code is in the project, libopen62541:

```
examples/pubsub_realtime/opcua_summation_controller.c
```

6.3.7.7.2.2 opcua_summation_device

The program is running on the device. Its main job is to sample data and package them in DataSet messages and send them to the NETC switch.

The command path in the filesystem is: `/root/open62541_example/opcua_summation_device`

```
Usage: /root/open62541_example/opcua_summation_device <id>
  id: the node id
```

Its source code is in the project,`libopen62541`:

```
examples/pubsub_realtime/opcua_summation_device.c
```

6.3.7.7.2.3 NETC Switch frame modification driver

This Linux kernel driver supports the NETC switch frame modification feature. It is a kernel module by default, `netc_fm.ko`. It is located in the kernel source code:

```
drivers/net/dsa/netc/netc_fm.c
```

The program, `opcua_summation_device`, uses the interface provided by this driver to configure the switch and send DataSet messages to the switch.

6.3.7.7.3 OPC UA controller configuration

One i.MX943-EVK board can function as the controller.

- Install the latest Real-Time Edge image on one of the i.MX943-EVK boards.
- The controller is also the PTP timeTransmitter. Change the priority in the PTP config file, `/etc/ptp4l_cfg/gPTP.cfg`, as shown below:

```
[global]
gmCapable              1
```

```
priority1               1
priority2               1
logAnnounceInterval     0
logSyncInterval         -3
syncReceiptTimeout      3
neighborPropDelayThresh 20000000
min_neighbor_prop_delay -20000000
assume_two_step         1
path_trace_enabled      1
follow_up_info          1
transportSpecific       0x1
ptp_dst_mac             01:80:C2:00:00:0E
network_transport       L2
delay_mechanism         P2P
tx_timestamp_timeout    20
```

- Stop the NTP and SNTP client to avoid unexpected time-shift.

```
systemctl stop ntpd.service
systemctl stop systemd-timesyncd.service
```

- Synchronize the PTP clock of eth1 to CLOCK_REALTIME, and start ptp4l on the eth1 port:

```
phc2sys -s CLOCK_REALTIME -c eth1 -O 0 -m > ./phc2sys.log 2>&1 &
sleep 10
ptp4l -i eth1 -f /etc/ptp4l_cfg/gPTP.cfg -m > ./ptp4l.log 2>&1 &
```

- Execute the program, `opcua_summation_controller`. Use the eth1 port to send the initial OPC UA frames and receive it back on the eth2 port.

```
root@imx943-19x19-lpddr5-evk:~# /root/open62541_example/
opcua_summation_controller eth1 eth2
```

- After the devices are setup, the console output appears as shown below. The message shows that it received one DataSet message of OPC UA. The `Field, Count`, which is filled by one device increases constantly.

```
6: Sent one frame of 60 bytes
  TX HW Timestamp: 11801300567219 ns
  Received 85 bytes
  Application level roundtrip time: 252668 ns (252.668 µs)
  DataSet message count: 1
    DataSetMessage[0]:
      DataSetWriterId: 1001
      Field count: 2
        Field[0]: string
          Name: Node 100
        Field[1]: int32
          Count: 338
7: Sent one frame of 60 bytes
  TX HW Timestamp: 11801400567723 ns
  Received 85 bytes
  Application level roundtrip time: 231459 ns (231.459 µs)
  DataSet message count: 1
    DataSetMessage[0]:
      DataSetWriterId: 1001
      Field count: 2
        Field[0]: string
          Name: Node 100
        Field[1]: int32
        Count: 339
```

6.3.7.7.4 OPC UA Device configuration

Configure other i.MX943-EVK boards, which work as the OPC UA device. If you have multiple boards, follow the steps described in this section on each board.

1. Install the latest image of Real-Time Edge on i.MX943-EVK board.
2. Stop the NTP and SNTP client to avoid unexpected time-shift.

```
systemctl stop ntpd.service
systemctl stop systemd-timesyncd.service
```

3. Configure the NETC switch:

```
ip link set eth0 up
ip link set swp0 up
ip link set swp1 up
ip link set swp2 up
ip link add name br0 type bridge
ip link set dev swp0 master br0
ip link set dev swp1 master br0
ip link set dev swp2 master br0
ip link set dev br0 up
```

4. Start ptp4l to start clock synchronization:

```
ptp4l -i swp0 -f /etc/ptp4l_cfg/gPTP.cfg -m > ./ptp4l.log 2>&1 &
sleep 10
phc2sys -s swp0 -c CLOCK_REALTIME -O 0 -m > ./phc2sys.log 2>&1 &
```

5. Wait for clock to be synchronized. Check the `ptp4l.log` file:

```
# tail ptp4l.log
ptp4l[1201.812]: rms   51 max   70 freq  +6580 +/-  51 delay  1526 +/-   0
ptp4l[1202.814]: rms   54 max   70 freq  +6484 +/-  59 delay  1523 +/-   0
ptp4l[1203.816]: rms   67 max   88 freq  +6607 +/-  65 delay  1526 +/-   0
ptp4l[1204.819]: rms   65 max   84 freq  +6487 +/-  71 delay  1523 +/-   0
ptp4l[1205.822]: rms   60 max   79 freq  +6589 +/-  68 delay  1524 +/-   0
ptp4l[1206.824]: rms   66 max   79 freq  +6477 +/-  72 delay  1524 +/-   0
```

6. Check `phc2sys.log`:

```
# tail phc2sys.log
phc2sys[1300.145]: CLOCK_REALTIME phc offset        19 s2 freq   +6594 delay
   625
phc2sys[1301.145]: CLOCK_REALTIME phc offset       -26 s2 freq   +6555 delay
   625
phc2sys[1302.146]: CLOCK_REALTIME phc offset        15 s2 freq   +6588 delay
   625
phc2sys[1303.146]: CLOCK_REALTIME phc offset       -25 s2 freq   +6553 delay
   625
phc2sys[1304.147]: CLOCK_REALTIME phc offset        24 s2 freq   +6594 delay
   625
```

7. On the serial console, load the kernel module, `netc_fm.ko`:

```
root@imx943-19x19-lpddr5-evk:~# modprobe netc_fm
root@imx943-19x19-lpddr5-evk:~# lsmod | grep netc_fm
netc_fm                 12288  0
```

8. Execute the program, `opcua_summation_device`. (The parameter, 100, is its node ID. Give each device a different node ID.)

```
root@imx943-19x19-lpddr5-evk:~# /root/open62541_example/opcua_summation_device 100
```

```
Start to publish data...
```

9. Check the console output of the controller.

6.3.7.7.5 References

1. *i.MX 943 Applications Processor Reference Manual, Chapter 161 Ethernet and Network Controller (NETC)*:
   a. 161.3.6.1.4.2 Egress frame modification
   b. 161.3.6.8.3 Ingress Port Filter Table
   c. 161.3.6.8.10 Ingress Stream Table
   d. 161.3.6.8.12 Egress Treatment Table
   e. 161.3.6.8.19 Frame Modification Table
   f. 161.3.6.8.20 Frame Modification Data Table
2. *OPC UA specification: UA Part 14: PubSub*

6.4 NETCONF/YANG

This section provides an overview of the NETCONF protocol and Yang (a data modeling language for NETCONF). It describes how to enable the NETCONF feature in the Real-time Edge software. It describes the applications, installation, and configuration steps of the NETCONF protocol. It also provides operation examples, a Web UI demo, and highlights troubleshooting aspects of NETCONF.

6.4.1 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. These features enable 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 102. 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 used to model the configuration and state data used by NETCONF operations, remote procedure calls (RPCs), and server event notifications. The device configuration data is stored in the form of an XML document. The specific nodes in the document and 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 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). 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 such cases, there are mechanisms defined to enable extensions while keeping the support for the standardized core. This whole mechanism is designed in a liberal fashion, therefore, the configuration does not concern the network strictly. 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.

6.4.2 Netopeer2

6.4.2.1 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. It allows developers to control their devices via NETCONF and operators to connect to their NETCONF-enabled devices.

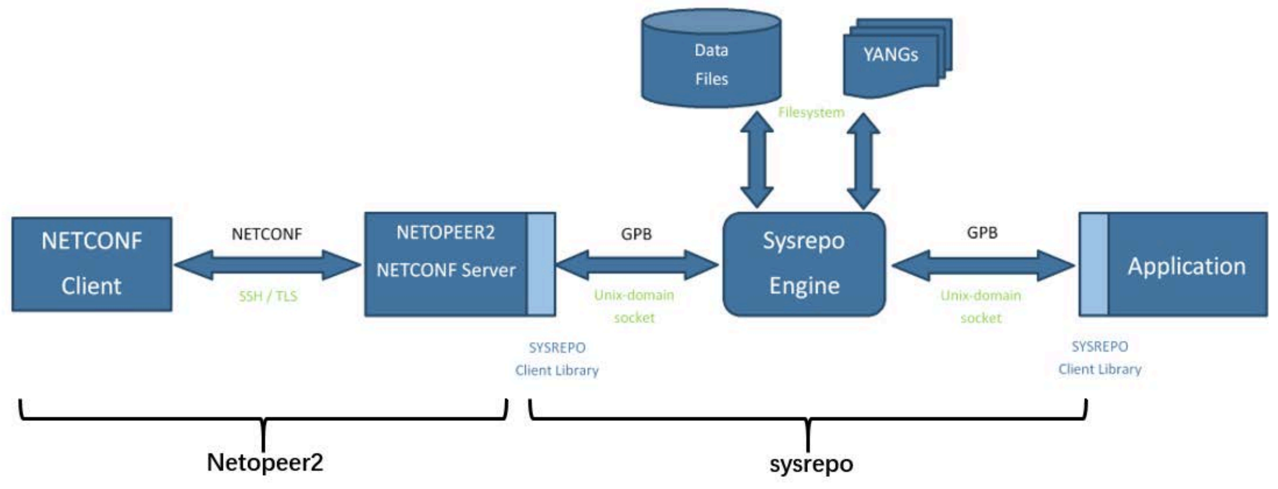


Figure 192. High-level architecture of Netopeer and sysrepo

6.4.2.2 Installing Netopeer2 on Ubuntu22.04

Use the following steps for installing Netopeer2-cli on Ubuntu22.04 operating systems.

1. Install the following packages:

```
$ sudo apt update
$ sudo apt install -y git cmake build-essential bison autoconf \
dh-autoreconf flex libavl-dev libprotobuf-c-dev protobuf-c-compiler \
zlib1g-dev libgcrypt20-dev libssh-dev libev-dev libpcre3-dev
```

2. Install Netopeer2:

```
sudo apt install -y netopeer2 sysrepo-plugind libyang2-tools
```

Attention: *You may encounter an error as shown below:*

```
/usr/share/netopeer2/merge_hostkey.sh: line 30: /dev/null: restricted: cannot
 redirect output
dpkg: error processing package netopeer2 (--configure):
 installed netopeer2 package post-installation script subprocess returned
 error exit status 1
Errors were encountered while processing:
 netopeer2
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

Use the below workaround for the above mentioned error:

```
sudo /usr/share/netopeer2/merge_hostkey.sh
sudo /usr/share/netopeer2/merge_config.sh
sudo apt install -f
```

6.4.2.3 Sysrepo

Sysrepo is a YANG-based configuration and operational state data store for UNIX or Linux applications.

Applications can use sysrepo to store their configuration modeled by the provided YANG model instead of using, for example, flat configuration files. Sysrepo ensures data consistency of the data stored in the datastore and enforce data constraints defined by the YANG model. Applications can currently use the C language API of sysrepo Client Library to access the configuration in the datastore. However, the support for other programming languages is planned for later. Since sysrepo uses Google Protocol Buffers (GPB) as the interface between the datastore and client library. Therefore, writing of a native client library for any programing language that supports GPB is possible.

For information about sysrepo, refer to the URL: https://netopeer.liberouter.org/doc/sysrepo/master/html/.

6.4.2.4 Netopeer2 server

To implement network configuration management based on the NETCONF protocol, Netopeer2 includes a server (`netopeer2-server`). It is based on the new generation of the NETCONF and YANG libraries - libyang and libnetconf2. Netopeer2 uses sysrepo as a NETCONF datastore implementation.

6.4.2.5 Netopeer2 client

Netopeer2 includes a simple command-line NETCONF client, netopeer2-cli, which is built and installed by default. The Netopeer2 client is primarily used for Netopeer2 server testing. It allows the user to establish a NETCONF session with a NETCONF-enabled device to obtain or edit its configuration data.

6.4.2.6 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.

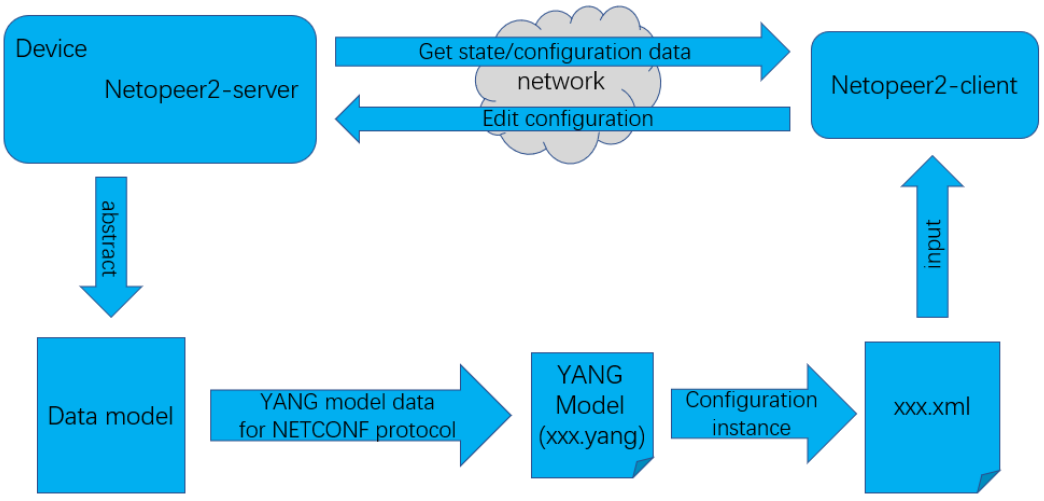


Figure 193. Workflow for netopeer

6.4.3 Configuration

6.4.3.1 Enabling NETCONF feature

This feature is enabled by default in Real-time Edge software.

The below packages are enabled by default in Real-time Edge software:

```
libyang libnetconf2 sysrepo netopeer2-server real-time-edge-sysrepo
```

Some libraries are loaded by sysrepo-plugind to implement the tsn configuration based on **sysrepo**. It is enabled for **LS1028ARDB**, **i.MX 8DXL LPDDR4 EVK, i.MX 8M Plus LPDDR4 EVK**, and **i.MX 93 EVK**.

Note: *For LS1028ARDB board, Qbv, Qbu, Qci, stream identification in CB, IP, MAC, and VLAN are supported.*

6.4.3.2 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.

6.4.3.3 Netopeer2-cli

6.4.3.3.1 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 6.4.3.3.2.
   - **--source** datastore: Specifies the source datastore for the `copy-config` operation. For description of the datastore parameter, refer to Section 6.4.3.3.2.
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 6.4.3.3.2. 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 RFC 6243 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 6.4.3.3.2. 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 6.4.3.3.2. 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 6.4.3.3.2. Note, that the url configuration datastore cannot be unlocked.
15. **verb**
    - Enables/disables verbose messages.
16. **quit**
    - Quits the program.

6.4.3.3.2 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.

6.4.3.4 Sysrepocfg

The **sysrepocfg** command-line tool allows editing, importing, and exporting configuration stored in the Sysrepo datastore. It also allows users to edit the startup or running configuration of a specified module in a preferred text editor. It 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 the Sysrepo client library for any data manipulation rather than directly accessing configuration data files. Therefore, it effectively inherits all main features of Sysrepo, such as YANG-based data validation, transaction, and concurrency support. Most importantly, subscribed applications are notified about the changes made using `sysrepocfg` and can immediately take the new configuration into account.

6.4.3.5 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).

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 /home/user/ietf-interfaces.yang --owner root --group root
 --permissions 600
```

- Changes the ownership and permissions of an existing YANG module:

```
sysrepoctl --change ietf-interfaces --owner root --group root --permissions 644
```

- Enables a feature within a YANG module:

```
sysrepoctl --change ietf-interfaces --enable-feature if-mib
```

- Uninstall a module:

```
sysrepoctl --uninstall ietf-interfaces
```

6.4.3.6 List of yang models

Table 103. Revision of yang models

YANG Model Name	Revision	Enabled Features
ietf-interfaces	2018-02-20	
ietf-ip	2018-02-22	ipv4-non-contiguous-netmasks
ieee802-dot1q-bridge	2023-10-26	
ieee802-dot1q-sched	2023-10-22	scheduled-traffic
ieee802-dot1q-sched-bridge	2023-10-26	
ieee802-dot1q-preemption	2023-10-26	frame-preemption
ieee802-dot1q-preemption-bridge	2023-10-26	frame-preemption
ieee802-dot1q-stream-filters-gates	2023-07-03	closed-gate-state
ieee802-dot1q-stream-filters-gates-bridge	2023-07-03	
ieee802-dot1q-psfp	2023-10-26	psfp
ieee802-dot1q-psfp-bridge	2023-10-26	
ieee802-dot1cb-stream-identification	2021-12-08	
ieee802-dot1cb-stream-identification-types	2021-12-09	
ieee802-dot1cb-frer	2021-12-08	
ieee802-dot1cb-frer-types	2021-12-08	
ieee802-dot1q-pb	2023-10-22	
ieee802-dot1q-qci-augment	2019-05-20	
nxp-bridge-vlan-tc-flower	2020-04-02	
ieee1588-ptp-tt	2023-08-14	path-trace
ieee802-dot1as-gptp	2024-09-11	
ieee802-dot1ab-lldp	2022-03-15	

6.4.3.7 Operation examples

The following figure describes the steps to configure the device via netopeer2:

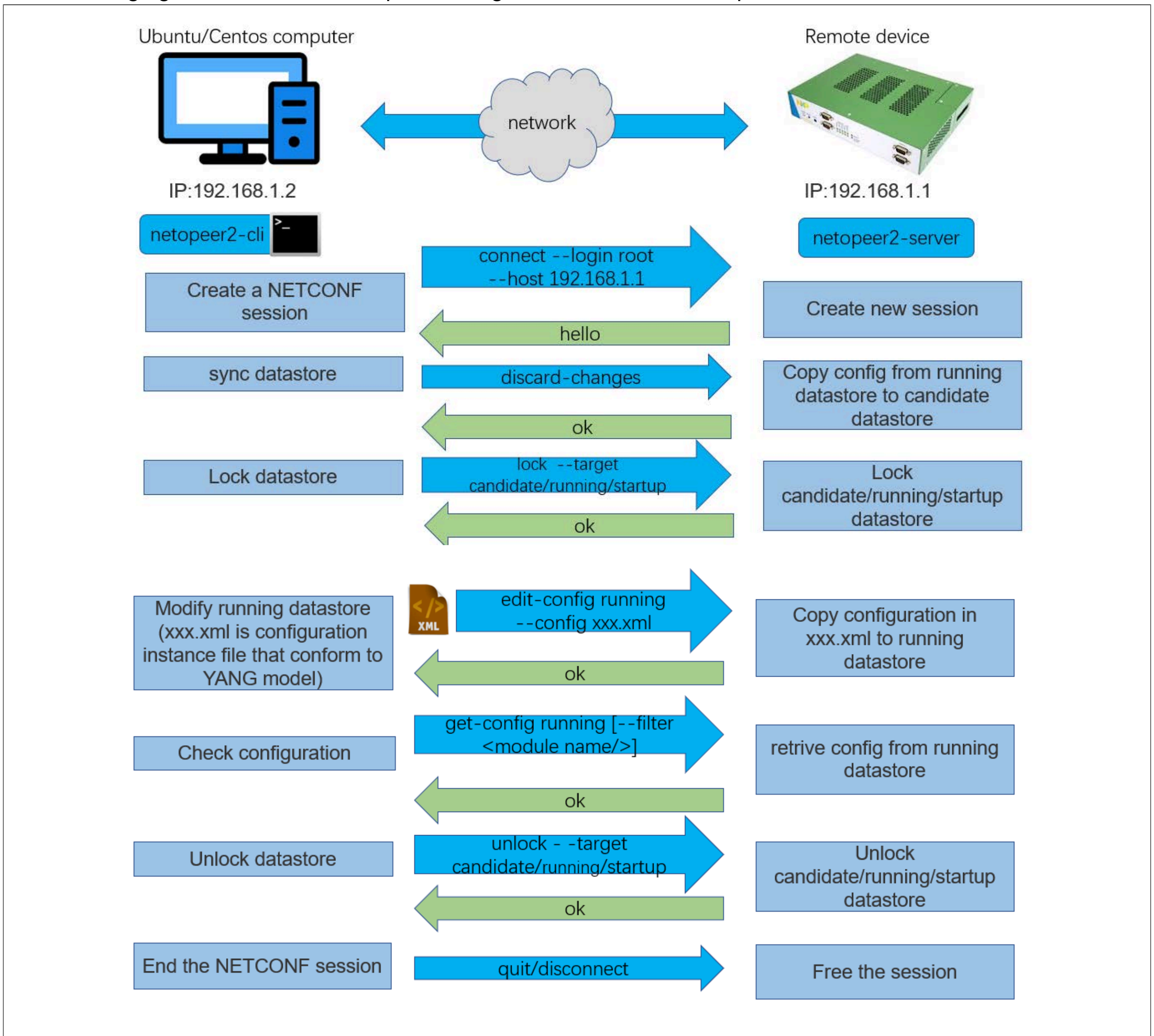

Ubuntu/Centos computer
Remote device
network
IP:192.168.1.2
IP:192.168.1.1
netopeer2-cli
netopeer2-server
Create a NETCONF session
connect --login root --host 192.168.1.1
Create new session
hello
sync datastore
discard-changes
Copy config from running datastore to candidate datastore
ok
Lock datastore
lock --target candidate/running/startup
Lock candidate/running/startup datastore
ok
Modify running datastore (xxx.xml is configuration instance file that conform to YANG model)
XML
edit-config running --config xxx.xml
Copy configuration in xxx.xml to running datastore
ok
Check configuration
get-config running [--filter <module name/>]
retrive config from running datastore
ok
Unlock datastore
unlock - -target candidate/running/startup
Unlock candidate/running/startup datastore
ok
End the NETCONF session
quit/disconnect
Free the session

Figure 194. Steps to configure the device via netopeer2 using the MAC0 interface on LS1028ARDB

The git repository, `real-time-edge-sysrepo`, includes a few *instance files* to configure TSN features on the board:

- Instance files for TSN configuration

Users can configure the TSN functions using these instance files. Before starting, ensure that **sysrepo-plugind**, and **netopeer2-server** are running on the board. Use the following steps to configure the TSN feature.

1. On a Linux command console, execute the `netopeer2-cli` command to start:

```
$ netopeer2-cli
```

2. Connect to the netopeer2-server on the board (this example assumes that the IP address of the board is 10.193.20.53):

```
> connect --login root --host 10.193.20.53
```

3. Get the status data of the server:

```
> get
```

4. Get the configuration data in the running datastore using the command below:

```
> get-config --source running
```

5. Configure the QBV feature. The xml file, `qbv-swp0-enable.xml`, is located on github (https://github.com/nxp-real-time-edge-sw/real-time-edge-sysrepo/tree/master/Instances):

```
> edit-config --target running --config=qbv-swp0-enable.xml
```

6. Check the configuration data of QBV:

```
> get-config --source running --filter-xpath /ietf-interfaces:interfaces/
interface[name='swp0']/ieee802-dot1q-bridge:bridge-port/ieee802-dot1q-
preemption-bridge:frame-preemption-parameters
```

7. Set the LLDP configuration. The xml file, `lldp.xml`, is located on github (https://github.com/nxp-real-time-edge-sw/real-time-edge-sysrepo/tree/master/Instances).

```
edit-config --target running --config=lldp.xml
```

8. Get the LLDP configuration.

```
get-config --source running --filter-xpath /ieee802-dot1ab-lldp:lldp
```

9. Get the operational datastore of LLDP using the command below:

```
get-data --datastore operational --filter-xpath /ieee802-dot1ab-lldp:lldp
```

10. To validate the LLDP configuration, execute the commands below on the console of the board:

```
lldpcli -f json show configuration
lldpcli -f json show chassis details
lldpcli -f json show interfaces
```

11. Set the PTP parameters. It creates a new configuration file in `/etc/ptp4l_cfg/instance0.cfg`. Use this file to start the PTP command, `ptp4l`. The xml file, `ptp.xml`, is located on github (https://github.com/nxp-real-time-edge-sw/real-time-edge-sysrepo/tree/master/Instances).

```
> edit-config --target running --config=ptp.xml
# ptp4l -i hms0p2 -p /dev/ptp2 -f /etc/ptp4l_cfg/instance0.cfg -m
```

12. Get the PTP parameters using the command:

```
get-config --source running --filter-xpath /ieee1588-ptp-tt:ptp
```

13. Copy the configuration data in the **running** datastore to the **startup** datastore:

```
> copy-config --source running --target startup
```

14. Disconnect the netopeer2-server:

```
> disconnect
```

6.4.3.8 Application scenarios on LS1028A

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 must 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 the configuration file, change the 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`.
   Note: *The command only applies to the board with a switch, for example, LS1028ARDB.*
   ```
   $ ip link add name br1 type bridge
   ```
   b. Edit the configuration file, change the 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 the Ethernet interface:
   a. Create bridge named "`br1`" if not existing:
   Note: *Enter this command on the device console*
   ```
   $ 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.

Note: *Enter this command on the device console*

```
# 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 must 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
      Note: *Enter this command on the device console.*

      ```
      # 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.
      Note: *Enter this command on the device console.*

      ```
      # tc filter show dev swp0 ingress
      ```

   d. Disable the configuration.

      ```
      > edit-config --target running --config=switch-qci-fm-gate-disable.xml
      ```

      Note:
      - *The switch must learn the destination-address in the instance file.*
      - *Users must send the `switch-qci-fm-gate-disable.xml` after `switch-qci-fm-gate-enable.xml`.*

7. Configure LS1028ARDB Qbu via ethtool using the steps below.
   Note: *Disable cut through on the target board first by executing the command below:*

   ```
   # tsntool ctset --device swp0 --queue_stat 0x0
   ```

   a. Edit and send configuration file:

      ```
      > edit-config --target running --config=qbu-swp0.xml
      ```

   b. Show the result:

```
# ethtool --show-frame-preemption swp0
```

8. Configure LS1028ARDB VLAN ID and priority filter via tc:
   a. Edit configuration file, change the interface name and action_spec:

```
$ vim ietf-br-vlan-cfg.xml
```

   b. Send the configuration file:

```
> edit-config --target running --config=ietf-br-vlan-cfg.xml
```

9. Configure i.MX 8DXL / i.MX 8M Plus / i.MX 93 Qbv via tc.
   a. Edit and send configuration file:

```
> edit-config --target running --config=qbv-eth1-enable.xml
```

   b. Display the result using the command below:

```
# tc qdisc show dev eth1
```

10. Configure i.MX 8DXL / i.MX 8M Plus / i.MX 93 Qbu via ethtool.
   a. Edit and send configuration file:

```
> edit-config --target running --config=qbu-eth1.xml
```

   b. Display the result using the command below:

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

6.4.4 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.

6.5 Wireless on LS1028A

6.5.1 NFC

NFC click board is a mikroBUS add-on board with a versatile near field communications controller from NXP — the PN7120 NFC devices are used in contactless payment systems, electronic ticketing, smartcards. In retail and advertising, inexpensive NFC tags can be embedded into packaging labels, flyers, or posters.

This board is fully compliant with NFC Forum specifications. This implies that users can use the full potential of NFC and its three distinct operating modes listed below:

1. Card emulation
2. Read/Write
3. P2P

6.5.1.1 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.

6.5.1.2 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.

6.5.1.3 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.*

6.5.1.4 Software preparation

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.

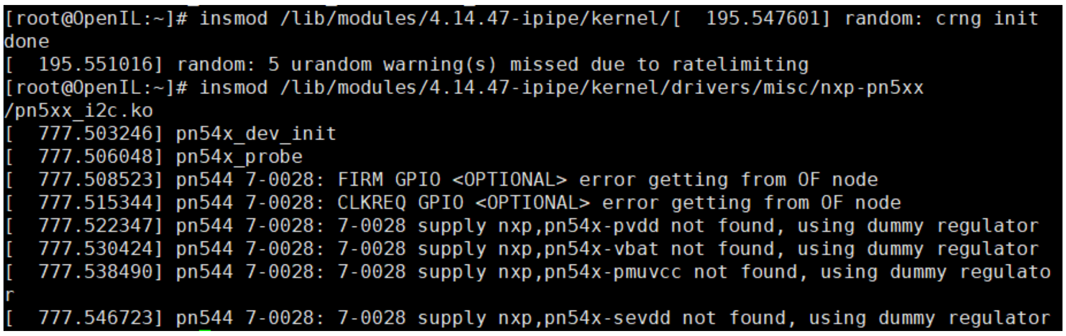
6.5.1.5 Testing the NFC click board

Use the following steps for testing the NFC clickboard:

1. Install the 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.

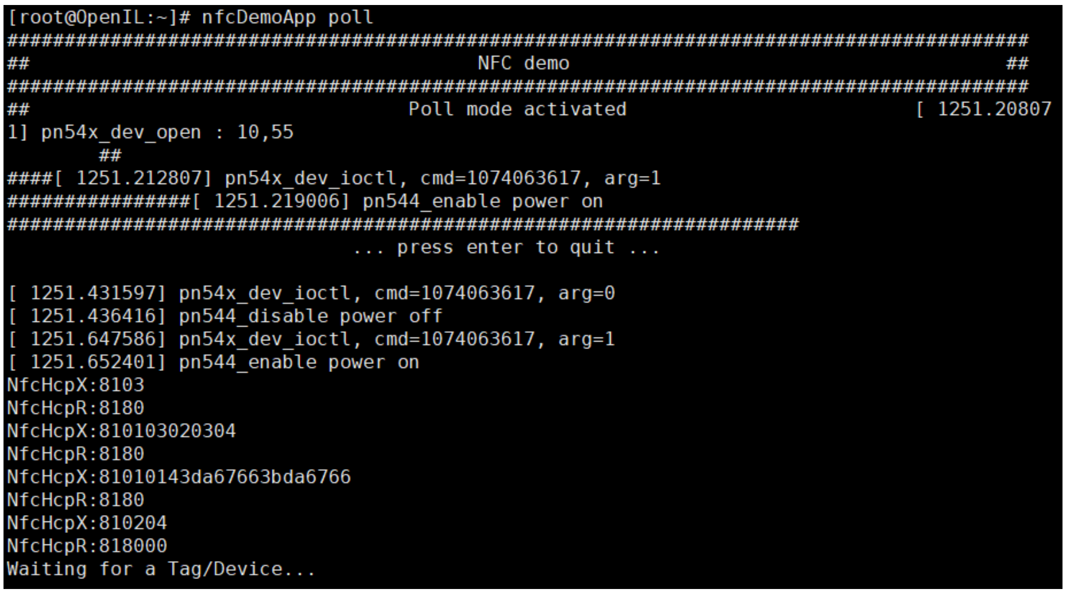


```
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/[  195.547601] random: crng init
done
[  195.551016] random: 5 urandom warning(s) missed due to ratelimiting
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/drivers/misc/nxp-pn5xx
/pn5xx_i2c.ko
[  777.503246] pn54x_dev_init
[  777.506048] 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
```

Figure 195. NFC driver module installation log

3. Run the **nfcDemoApp** application:

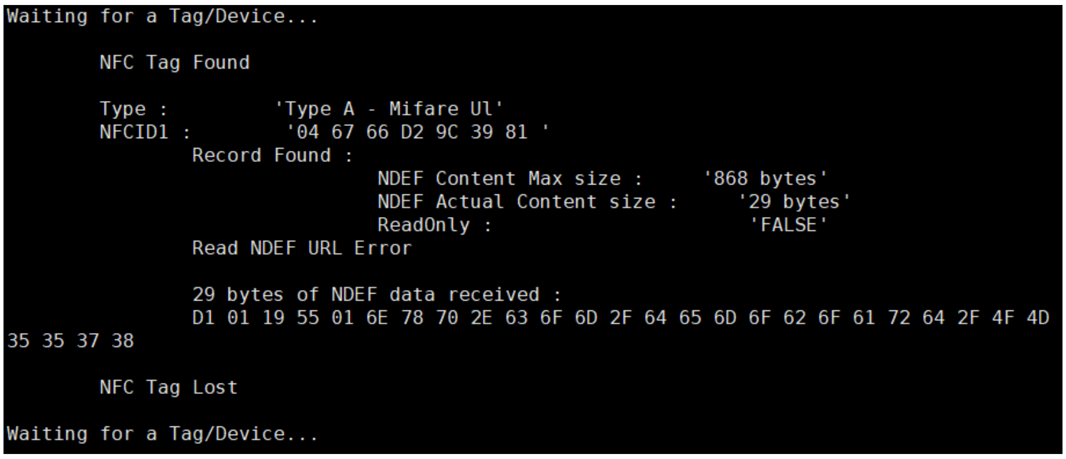
```
[root]# nfcDemoApp poll
```



```
[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...
```

Figure 196. Running the nfcDemoApp application log

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...
```

Figure 197. Log generated after the NFC sample card (tag) is displayed on the NFC click board

If the above information is displayed, it indicates that the card was read successfully.

6.5.2 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).

6.5.2.1 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.

6.5.2.2 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.

6.5.2.3 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

6.5.2.4 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:

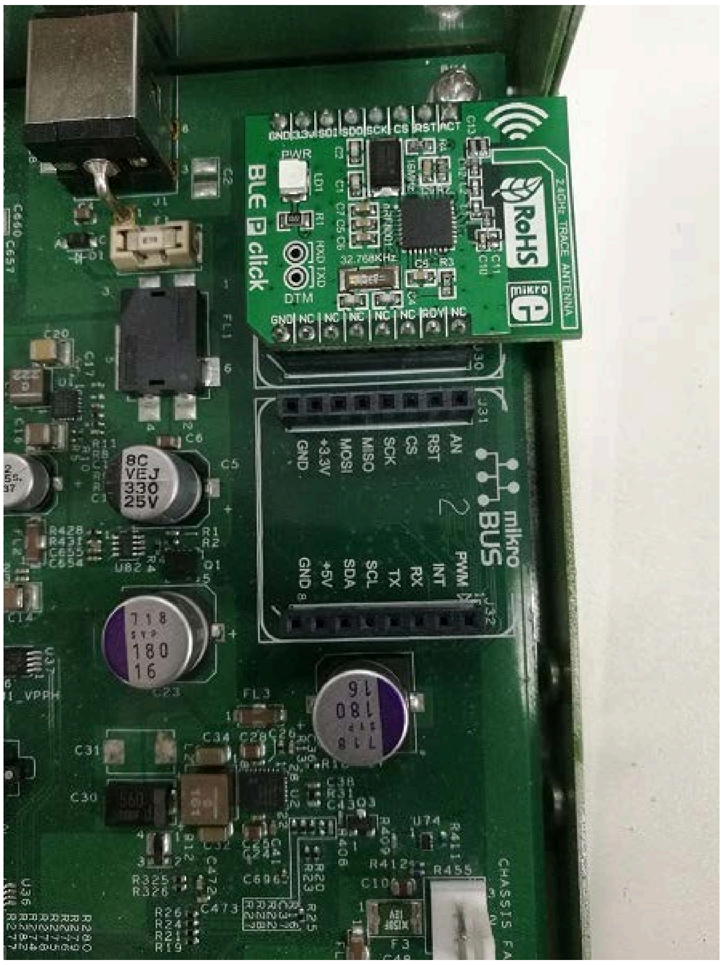

BLE P click
GND 3.3V SDI SDO SCK CS RST ACT
GND NC NC NC NC NC RDY NC
DTM
RoHS
2.4GHz TRACE ANTENNA
mikro BUS
AN RST CS SCK MISO MOSI +3.3V GND
PWM INT RX TX SCL SDA +5V GND
CHASSIS FAN

Figure 198. BLE P Click board hardware setup

6.5.2.5 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:

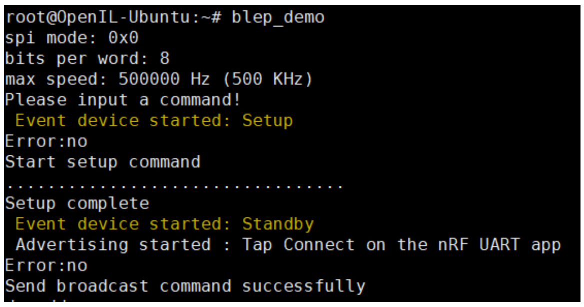
```
Device Drivers --->
    SPI support --->
        <*>   Freescale DSPI controller
        <*>   User mode SPI device driver support
```

  3. Use the `make` command to create the images.

6.5.2.6 Testing the BLE P click board

Use the following steps for testing the BLE P click board:

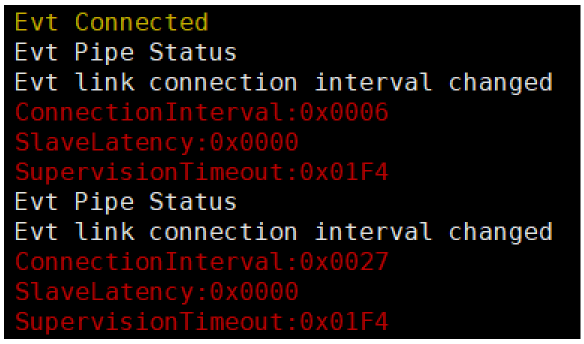
1. **Running the blep_demo application:**
   The following log is displayed to indicate that the BLE P click board is initialized. After this, users can scan from their mobile phone or the Bluetooth device of the computer for the BLE P click board. The name of the BLE P click board used is “**MikroE**”.



```
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 199. Initialization log

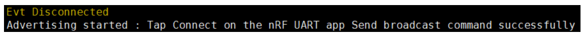
2. **Connection log**
   Connect the BLE P click board via mobile app. On successful connection, the following log is displayed. Thereafter, the application can communicate with the BLE P click board.



```
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 200. 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:

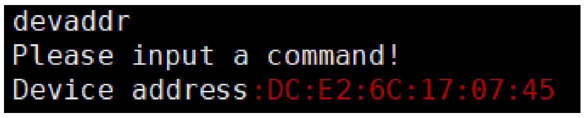


```
Evt Disconnected
Advertising started : Tap Connect on the nRF UART app Send broadcast command successfully
```

Figure 201. 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.

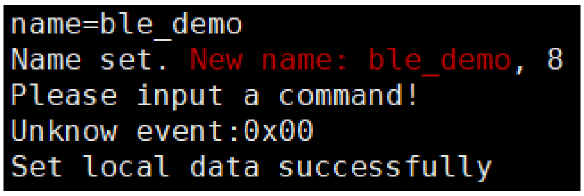


```
devaddr
Please input a command!
Device address:DC:E2:6C:17:07:45
```

Figure 202. 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.

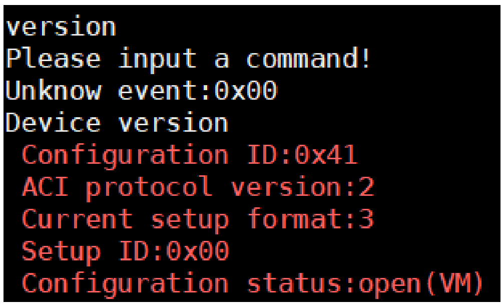


```
name=ble_demo
Name set. New name: ble_demo, 8
Please input a command!
Unknow event:0x00
Set local data successfully
```

Figure 203. 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.



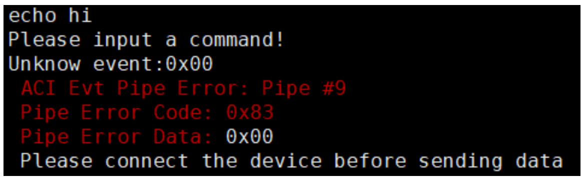
```
version
Please input a command!
Unknow event:0x00
Device version
 Configuration ID:0x41
 ACI protocol version:2
 Current setup format:3
 Setup ID:0x00
 Configuration status:open(VM)
```

Figure 204. version log

d. **echo**

This command is used to send a string to the Android app**.** This command must be executed after the connection is established. The maximum length is 20 characters.

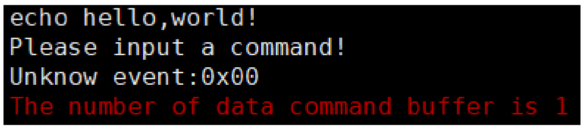
The below log displays the message displayed after user tries to send a string when no connection is established:



```
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 205. User input log (no connection)

The below log is displayed when user sends a string after a connection is established:

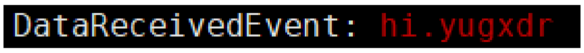


```
echo hello,world!
Please input a command!
Unknow event:0x00
The number of data command buffer is 1
```

Figure 206. User input log (after a connection is established)

5. Receiving data

When the Android app sends a string:



```
DataReceivedEvent: hi.yugxdr
```

Figure 207. Received data log

6.5.3 BEE

This section introduces the features of the BEE Click Board and how to use it on LS1028ARDB.

6.5.3.1 BEE/ZigBEE

LS1028ARDB supports the BEE click board, which implements the MRF24J40MA 2.4 GHz IEEE 802.15.4 radio transceiver module from Microchip.

6.5.3.2 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

6.5.3.3 Hardware preparation

Use the following hardware items for the BEE Click Board demo setup:

- Two LS1028ARDB boards
- Two BEE Click Boards

The Figure 208 describes the hardware setup for the BEE Click Board.

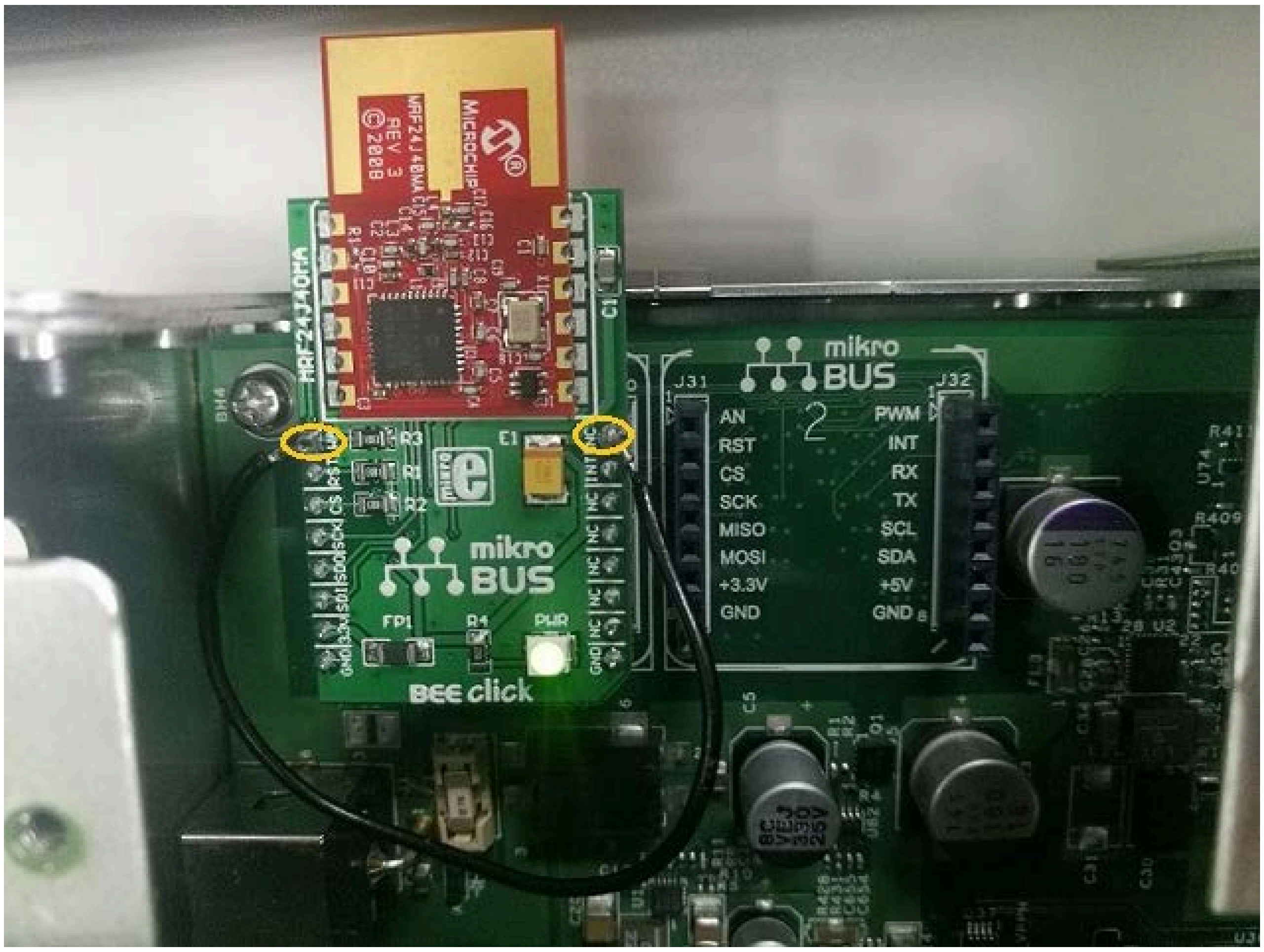


Figure 208. BEE Click Board hardware setup

Note: *The WA pin of BEE Click Board connects with the NC pin.*

6.5.3.4 Software preparation

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.

6.5.3.5 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. Create a file in any path. For example, `./samples/test.txt`.
2. Then, 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 board 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.
   ```

6.6 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
...
```

6.7 Wi-Fi on i.MX 8DXL EVK

6.7.1 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 104. 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
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.

6.7.2 Hardware Setup

Install the Wi-Fi card '**1XL M.2 Module**' to the M.2 connector (J17) on i.MX 8DXL EVK.

6.7.3 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 must 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
……
```

6.8 MODBUS

MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model. The protocol enables client/server communication between devices connected on different types of buses or network.

6.8.1 Libmodbus introduction

Real-time Edge integrates libmodbus library. Libmodbus is a free software library used to send or receive data with a device that conforms to the Modbus protocol. It contains various backends to communicate over different networks (for example, serial in RTU mode or Ethernet in TCP IPv4/IPv6).

All the i.MX series and Layerscape series of boards support modbus. It can be used to write both:

- Client applications that reads/writes data from various devices.
- Server applications that provide data to several clients.

The official website that contains the latest version of the documentation for libmodbus is https://libmodbus.org/.

6.8.2 Modbus-Simulator introduction

Modbus-Simulator is a Modbus tool based on the 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 client address of the device
- Modifies the baud rate of the device

Note: *Only the i.MX 8M Mini and i.MX 8M Plus boards support LED light function and CPU temperature function. The other boards define a placeholder variable. By default, Modbus-Simulator is disabled except i.MX 8M Mini and i.MX 8M Plus. Taking i.MX 93 as an example, add the following line to `meta-real-time-edge/conf/distro/include/real-time-edge-base.inc` to enable it.*

```
DISTRO_FEATURES:append:mx93-nxp-bsp = " modbus-simulator"
```

6.8.3 Modbus-Simulator usage

SYNOPSIS

```
{modbus_device_simulator|modbus_client_simulator} [OPTION]… {RTU-PARAMS|TCP-
PARAMS}… {SERIALPORT|HOST}… [WRITE-DATA]…
```

6.8.3.1 Parameter description

The parameter of [Option] is the general parameter of TCP and RTU startup.

Table 105. 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.
-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 106. 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 107. Parameter description

Parameter option	Description
-p,	port.

6.8.3.2 Examples of TCP and RTU

Examples of TCP `modbus_device_simulator` and `modbus_client_simulator` startup are as follows:

1. Start `modbus_device_simulator` locally with port 1502.

```
# 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
   ```

Note: For i.MX 6ULL, do the following:

J1704: pin7 - GND

J1703: pin1 - TX pin2 - RX

```
==> setenv fdtfile imx6ull-14x14-evk-lpuart.dtb
==> run bootcmd
```

Device name: /dev/ttymxc1

For i.MX 93, do the following:

J1001: pin25 - GND pin27 - TX pin28 - RX

```
==> setenv fdtfile imx93-11x11-evk-lpuart.dtb
==> run bootcmd
```

Device name: /dev/ttyLP4

For i.MX 95, do the following:

J27: pin25 - GND pin27 - TX pin28 - RX

```
==> setenv fdtfile imx95-15x15-evk-lpuart.dtb
==> run bootcmd
```

Device name: /dev/ttyLP4

6.8.3.3 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 rtu -a 1 -t 0x01 -r 0 -b 3 -p none /dev/
   ttymxc2
   ```

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
```

6.8.4 Testing Modbus-Simulator

Use two i.MX 8M Plus boards to test the functions of the modbus-simulator.

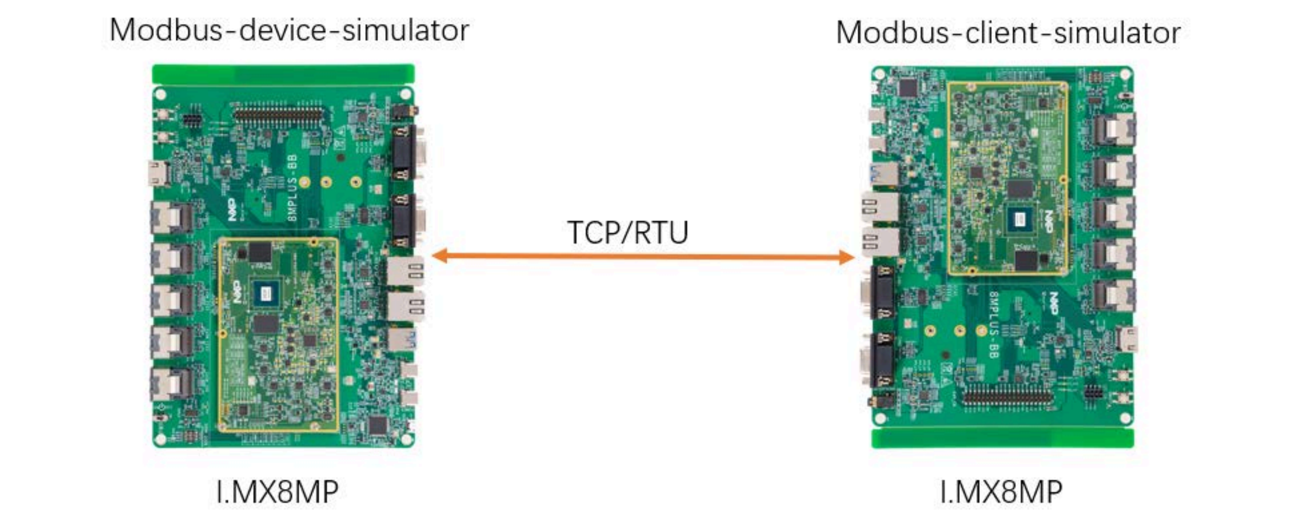


Figure 209. Testing the functions of the modbus-simulator

Note: *Learn about the start up parameters before starting.*

6.8.4.1 Testing TCP functions

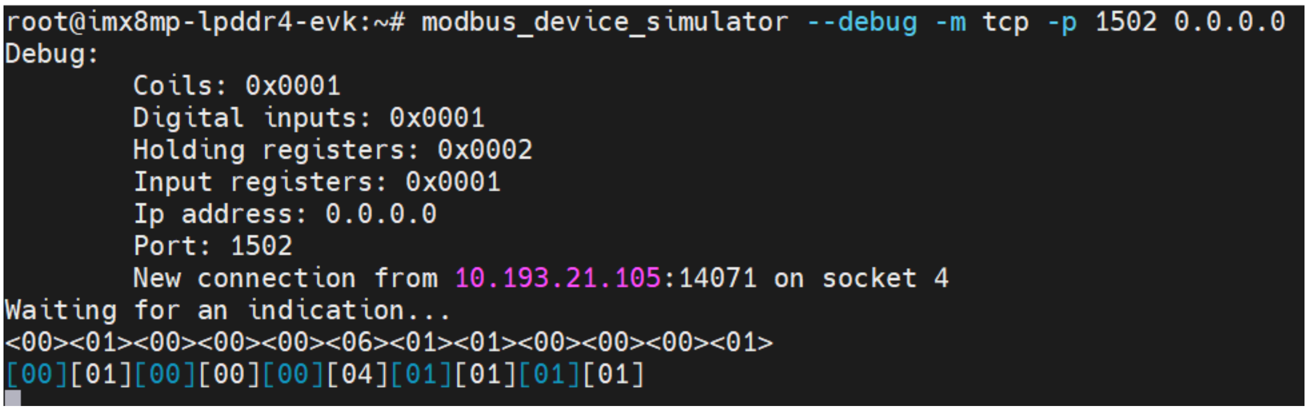
Read the status of the LED light on the board1.

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 210. Debug log on i.mx8mp-lpddr4-evk board

6.8.4.2 Testing RTU functions

Preparation: Connect the serial ports on the two boards, as shown in Figure 211.

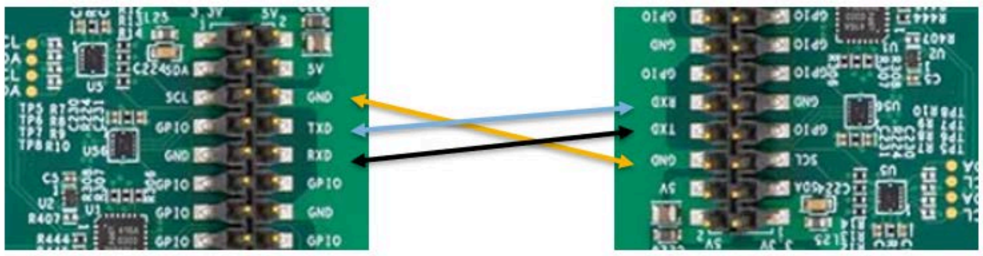


Figure 211. Connection of two i.MX 8MP boards

Use the below pin connections to connect the two boards:

- GND <---> GND
- TXD <---> RXD
- RXD <---> TXD

  Read the temperature of CPU on the board1

1. Enter board1 and start up `modbus_device_simulator`.

   ```
   # modbus_device_simulator --debug -m rtu -a 1 -b 115200 -p none /dev/ttymxc2
   ```

2. 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.*

6.8.5 Modbus for i.MX943 EVK

1. **pre-built image download**

   The pre-built images based on i.MX943-EVK Platform can be downloaded from the link below.

```
https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/tree/walnascar/
recipes-kernel/mcux-kernel/industrial-examples_imx943
```

2. Pre-built image

```
modbus_rtu_client_cm33_core1.bin_lpddr4_flash.bin
modbus_rtu_client_cm33_core1.bin_lpddr5_flash.bin
modbus_rtu_client_cm7_core1.bin_lpddr4_flash.bin
modbus_rtu_client_cm7_core1.bin_lpddr5_flash.bin
modbus_rtu_server_cm33_core1.bin_lpddr4_flash.bin
modbus_rtu_server_cm33_core1.bin_lpddr5_flash.bin
modbus_rtu_server_cm7_core1.bin_lpddr4_flash.bin
modbus_rtu_server_cm7_core1.bin_lpddr5_flash.bin
modbus_tcp_client_cm33_core1.bin_lpddr4_flash.bin
modbus_tcp_client_cm33_core1.bin_lpddr4_flash.bin
modbus_tcp_client_cm7_core1.bin_lpddr4_flash.bin
modbus_tcp_client_cm7_core1.bin_lpddr5_flash.bin
modbus_tcp_server_cm33_core1.bin_lpddr4_flash.bin
modbus_tcp_server_cm33_core1.bin_lpddr5_flash.bin
modbus_tcp_server_cm7_core1.bin_lpddr4_flash.bin
modbus_tcp_server_cm7_core1.bin_lpddr5_flash.bin
```

3. Run the image
   Burn flash.bin to MicroSD at 32 K(0x8000) offset with dd command.

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

   Plug the MicroSD card to the board.
   Change the boot mode to SW4[1:4] = x011 for SD boot.
   Enable MCU UARTs
   Power on the board.
   Please refer to bellow for details.
   https://mcuxpresso.nxp.com/mcuxsdk/25.06.00/html/boards/i.MX/imx943evk/gettingStarted/gsindex.html

Note: flash.bin is the pre-built image. And the /dev/sdx is the SD card device on your PC.

6.9 UART 9-bit Multidrop mode (RS-485) support

6.9.1 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:

- `iuart-9bit-interrupt-transfer` for interrupt mode
- `iuart-9bit-polling` for polling mode.

These two demos are supported on the i.MX 8M Mini LPDDR4 EVK board.

6.9.2 Building and running the demo

6.9.2.1 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 iuart-9bit-interrupt-transfer
```

The demo is located in the below directory:

```
tmp/deploy/images/imx8mm-lpddr4-evk/examples/
```

6.9.2.2 Hardware setup

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.

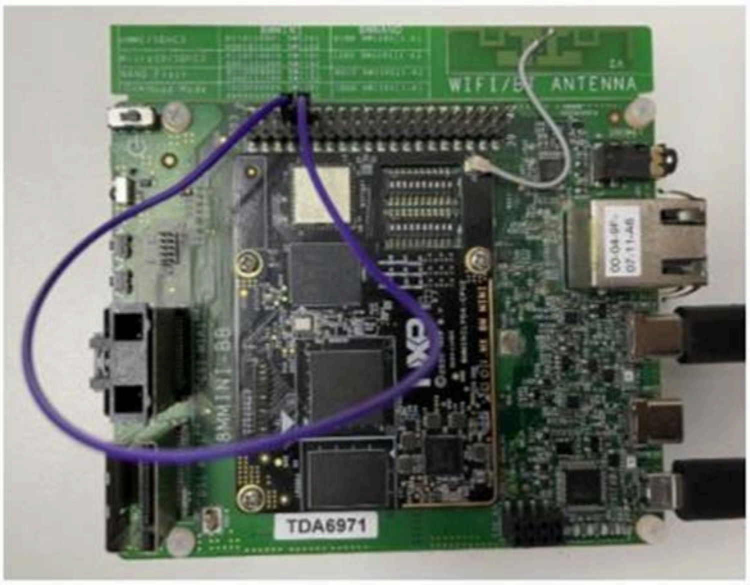


Figure 212. i.MX 8M Mini LPDDR4 EVK UART Connection

6.9.2.3 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

6.9.2.4 Running the demo

Power up the board and start up the M-core firmware under U-Boot by using the commands listed in the below paragraph. Use `iuart-9bit-interrupt-transfer` or `iuart-9bit-polling` to replace the "`DEMO_DIRECTORY`" or "`DEMO_NAME`" in the following commands:

If you choose to run the binary in DRAM:

```
=> ext4load mmc 1:2 0x80000000 /examples/mcuxsdk/DEMO_DIRECTORY/
DEMO_NAME_cm4.bin
=> dcache flush
=> bootaux 0x80000000
```

If you choose to run the binary in TCM:

```
=> ext4load mmc 1:2 0x48000000 /examples/mcuxsdk/DEMO_DIRECTORY/
DEMO_NAME_cm4.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. If the 9th bit of the data is zero, the bits are figured out to be “data”. In general, you must send the “address” with 9-bit “1” to the physical line of UART before sending “data” with 9-bit “0”. 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 first 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!
```

6.10 HSR network

High-availability Seamless Redundancy (HSR) is a Layer 2 redundancy protocol defined in the IEC 62439-3 standard, specifically designed for applications requiring zero recovery time and continuous data availability in the event of a network failure. Unlike traditional redundancy mechanisms that require time to detect and recover from faults, HSR provides seamless failover by duplicating frames and sending them simultaneously over two independent paths in a ring topology.

6.10.1 HSR network topology

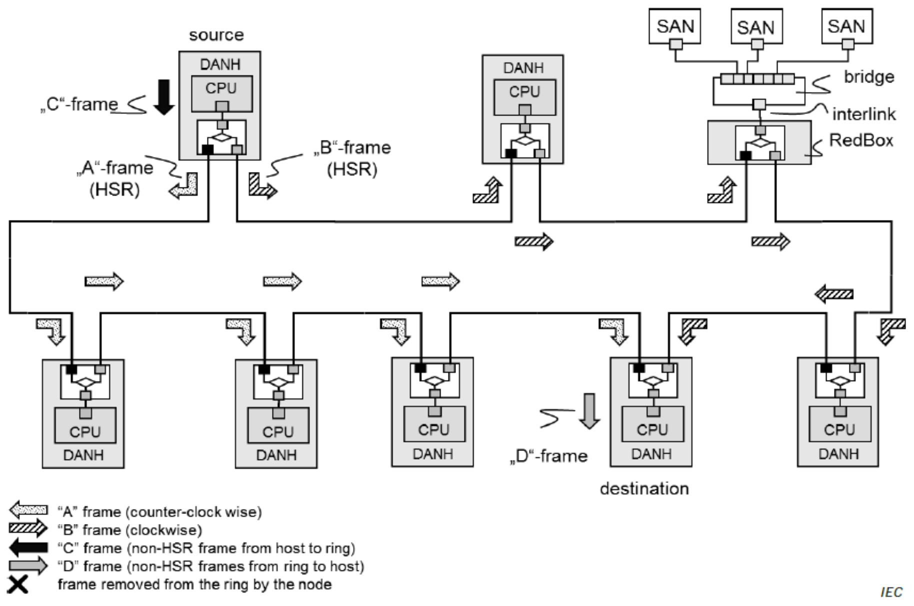

source
DANH
CPU
„C“-frame
„A“-frame (HSR)
„B“-frame (HSR)
SAN
SAN
SAN
bridge
interlink
RedBox
„D“-frame
destination
“A” frame (counter-clock wise)
“B” frame (clockwise)
“C” frame (non-HSR frame from host to ring)
“D” frame (non-HSR frames from ring to host)
frame removed from the ring by the node
IEC

Figure 213. HSR network topology

There are DANH, RedBox, SAN, and QuadBox in a HSR network.

A **DANH (Doubly Attached Node – HSR)** is a device that is natively aware of HSR protocol and has two Ethernet ports connected to the HSR ring. These nodes are fully HSR-compliant and participate in frame duplication, forwarding, and duplicate frame elimination.

A **SAN (Singly Attached Node)** is a non-HSR device that has only one Ethernet port and does not support HSR. It cannot participate in the ring topology directly and must be connected to the HSR network through a RedBox.

A **RedBox (Redundancy Box)** is a specialized bridge that connects SANs to an HSR network, allowing them to participate as if they were HSR-capable.

A **QuadBox** is a redundant interconnection node that connects two separate HSR rings together while maintaining full HSR functionality across both rings. A QuadBox operates conceptually as two RedBoxes in HSR-HSR mode, back-to-back.

6.10.2 Frame format for HSR

HSR frames are identified uniquely by their HSR tag.

The HSR tag consists of the following fields as shown in Figure 214.

- 16-bit Ethertype (`HSR_EtherType` = 0x892F)
- 4-bit path identifier (`PathId`)
- 12 bit frame size (`LSDUsize`)
- 16-bit sequence number (`SeqNr`)

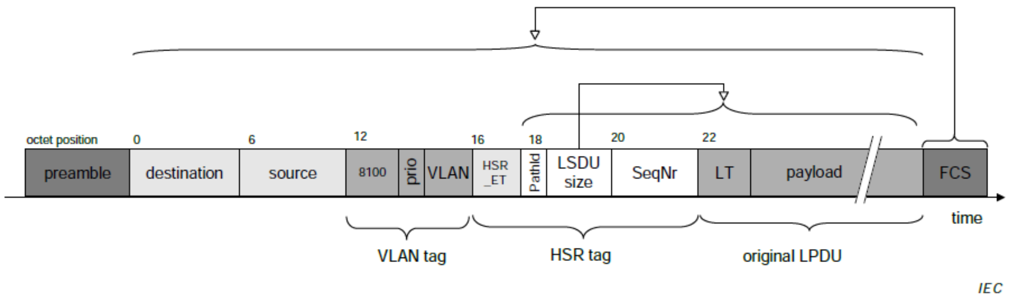


Figure 214. Frame format for HSR

6.10.3 HSR operational modes

A DANH is operable in one of the following modes, which can be modified at runtime using the below management commands:

- **Mode H (mandatory, default mode)**: HSR-tagged forwarding – in this mode, the DANH inserts the HSR tag on behalf of its host and forward the ring traffic, except for frames sent by the node itself, duplicate frames and frames for which the node is the unique destination.
- **Mode N (optional)**: No forwarding – in this mode, the node behaves like mode H with the exception that it not forward ring traffic from port to port.
- **Mode T (optional)**: Transparent forwarding – in this mode the DANH removes the HSR tag before forwarding the frame to the other port and send a frame from the host to both ports, untagged and without discarding duplicates.
- **Mode M (optional)**: Mixed forwarding HSR-tagged and non HSR-tagged – in this mode, the DANH inserts the HSR tag depending on local criteria when injecting frames into the ring. HSR tagged frames from ring ports are handled according to Mode H. Non-HSR tagged frames from ring ports are handled according to IEEE 802.1D forwarding rules.
- **Mode U (optional)**: Unicast forwarding – in this mode, the node behaves as in Mode H, except that it also forwards traffic for which it is the unique destination.
- **Mode X (optional)**: No sending on counter-duplicate – in this mode, the node behaves as in Mode H, except that a port not send a frame that is a duplicate of a frame that it received completely and correctly from the opposite direction.

6.10.4 HSR demo on i.MX boards

6.10.4.1 HSR on i.MX RT1180

The i.MX RT1180 MCU has a redundancy hardware module based on 802.1CB. There is a HSR stack to management the 802.1CB streams configuration to implement the HSR protocol.

The HSR stack includes:

- Software MAC learning
- Stream identification based on source MAC (802.1CB)
- FRER sequence generation and recovery (802.1CB)
- MAC filter drop (PSFP)
- Supervision frame send periodically
- Supervision frame check
- Dynamic FDB entry forget check

The figure below shows the HSR stack flow on i.MX RT1180:

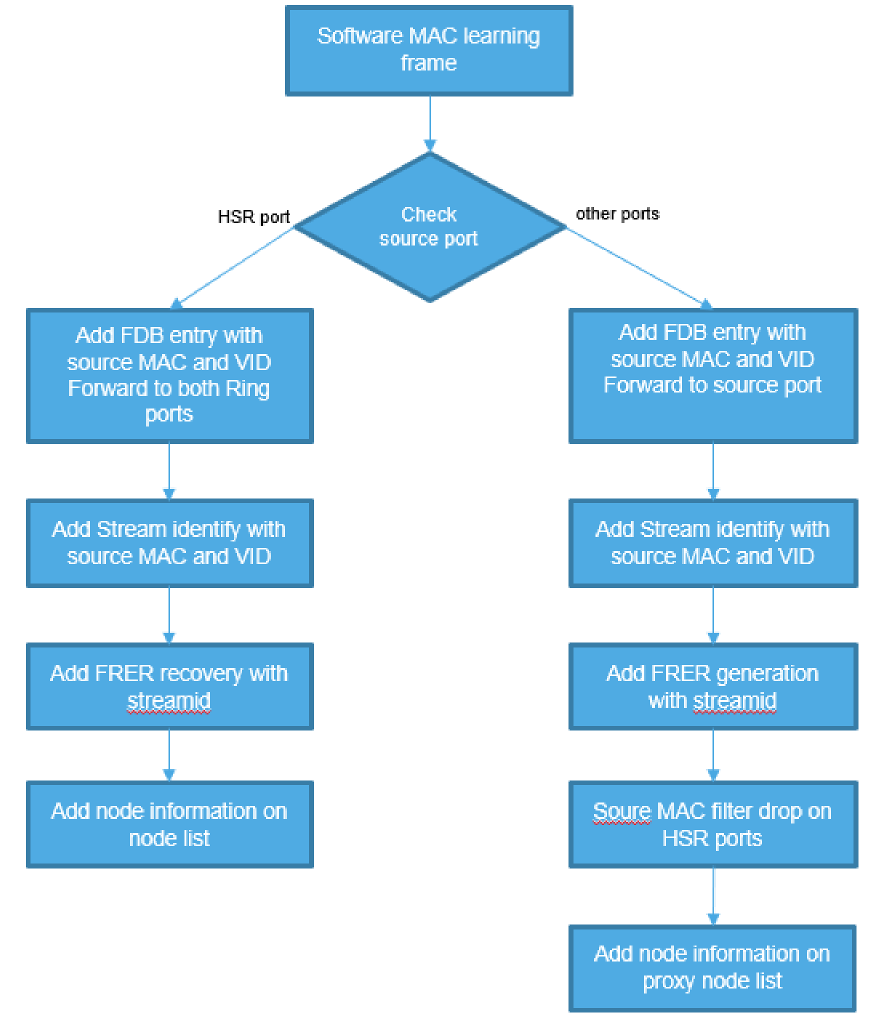

Software MAC learning frame
Check source port
HSR port
other ports
Add FDB entry with source MAC and VID Forward to both Ring ports
Add Stream identify with source MAC and VID
Add FRER recovery with streamid
Add node information on node list
Add FDB entry with source MAC and VID Forward to source port
Add Stream identify with source MAC and VID
Add FRER generation with streamid
Soure MAC filter drop on HSR ports
Add node information on proxy node list

Figure 215. HSR stack flow on i.MX RT1180

6.10.4.2 HSR demo

The following sections display the steps for performing the HSR demo using the supported NXP hardware platforms.

Flash the genAVB image `tsn_app.bin` on the boards. Add the following configuration to the boards:

```
BRIDGE>> cd /
BRIDGE>> mkdir hsr
BRIDGE>> cd hsr
BRIDGE>> write hsr_enabled 1
BRIDGE>> mkdir port0
BRIDGE>> write port0/type 0
BRIDGE>> mkdir port1
BRIDGE>> write port1/type 1
BRIDGE>> mkdir port2
BRIDGE>> write port2/type 4
BRIDGE>> mkdir port3
BRIDGE>> write port3/type 2
BRIDGE>> mkdir port4
BRIDGE>> write port4/type 2
BRIDGE>> mkdir port5
BRIDGE>> write port5/type 4
BRIDGE>> mkdir port6
BRIDGE>> write port6/type 5
```

Reset the boards after configuration.

Send traffic from PC1 to PC2. The traffic drops to zero packets when any link of the ring is disconnected or connected.

Use following command to configure the HSR operation mode:

```
hsr_mode_set <mode> [-p]
Parameters:
<mode>
• 0: Mode H (HSR-tagged forwarding)
• 1: Mode N (No forwarding)
• 2: Mode T (Transparent forwarding)
• 3: Mode U (Unicast forwarding)
```

6.10.4.3 HSR on i.mx943

6.10.4.3.1 HSR on M core

The i.mx943 has a HSR hardware module. There is a HSR driver and example in the MCUXSDK. Use the `netc_hsr_switch` example in MCUXSDK to implement the HSR network.

Configure the HSR in the `netc_hsr_switch.c` file:

```
hsrConfig.enableHsr  = 1;
hsrConfig.srPortIdxA = (netc_hw_port_idx_t)0;
hsrConfig.srPortIdxB = (netc_hw_port_idx_t)2;
hsrConfig.operMode   = kNETC_HSR_OPERATION_MODE_H;
```

Get the default `imagehsr_switch_cm33_core1_lpddr5_flash.bin` from `https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/blob/walnascar/recipes-kernel/mcux-kernel/industrial-examples_imx943/hsr_switch_cm33_core1_lpddr5_flash.bin` and use the following command to flash on TF card. HSR can be working on the i.mx943-evk board.

```
dd if=hsr_switch_cm33_core1_lpddr5_flash.bin of=/dev/sdX bs=1k seek=32 && sync
```

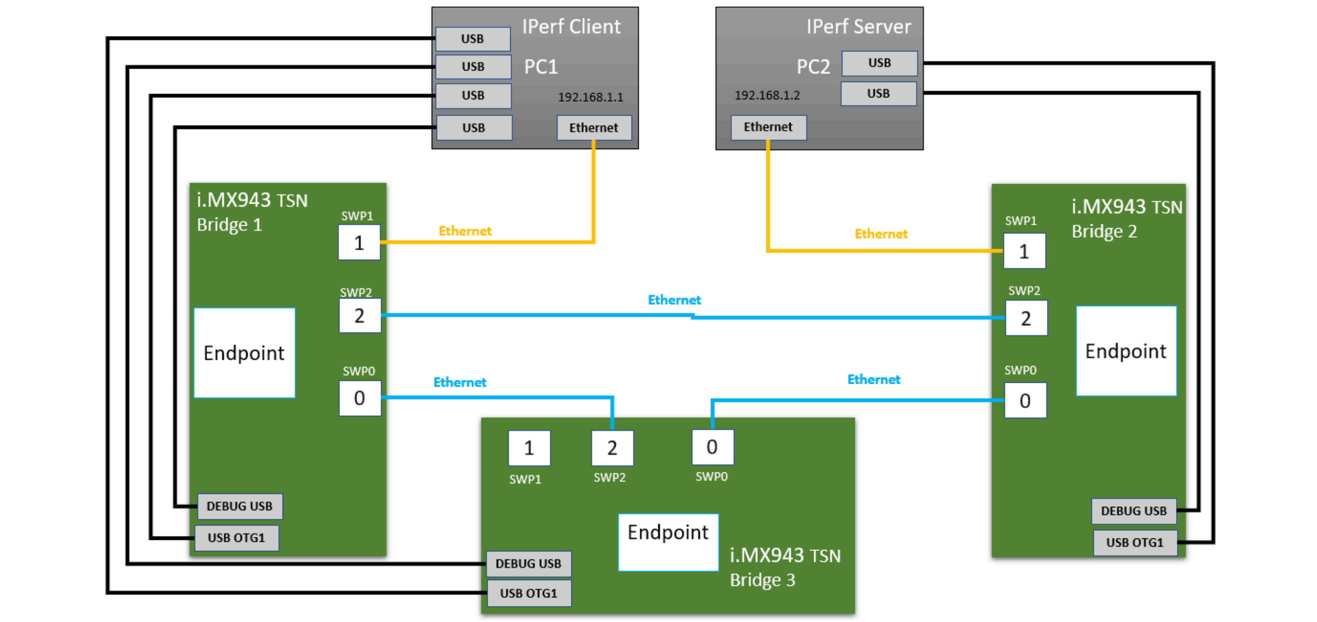


Figure 216. HSR demo setup on i.MX 943

i.MX943 bridge 1 and i.MX943 bridge 2 work as a RedBox. Send traffic from PC1 to PC2. The traffic drops to zero packets when any link of the ring is disconnected or connected.

6.10.4.3.2 HSR on Linux

1. Configure the HSR on each board by using following command

```
ip link add name hsr0 type hsr slave1 swp0 slave2 swp2 interlink swp1
 supervision 45 version 1
```

2. Add an IP address to the HSR interface on each board

```
ip addr add <IP_ADDR>/24 dev hsr0
```

3. Bring up the HSR interface

```
ip link set hsr0 up
```

4. Send traffic from PC1 to PC2, or to the hsr0 interface of any board. The traffic drops zero packets when any link of the ring is disconnected or connected.

6.10.4.4 HSR on Multi-Soc

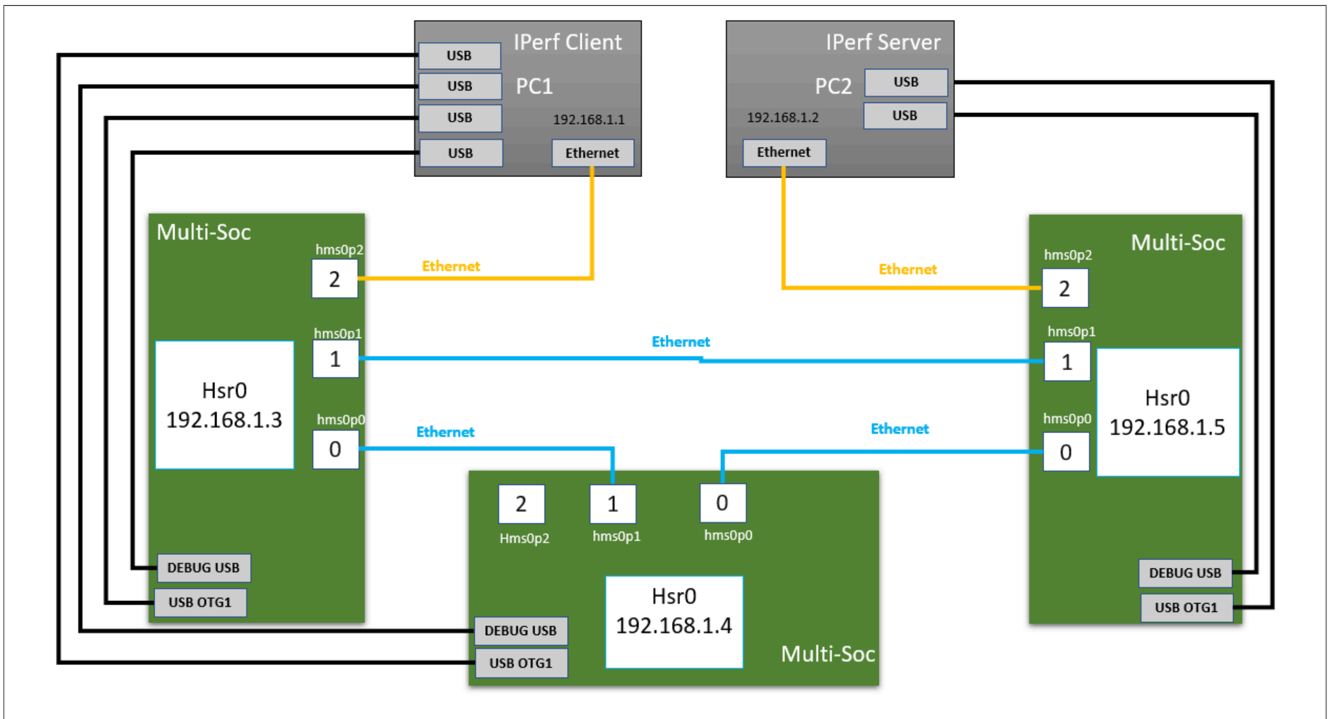


Figure 217. Hardware setup for HSR on multi-SoC

Add the following configuration on the Linux of the MPU.

1. Enable HSR offload for both interfaces.

```
ethtool -K hms0p0/hms0p1 hsr-fwd-offload on ethtool -K hms0p0/hms0p1 hsr-dup-
offload on ethtool -K hms0p0/hms0p1 hsr-tag-ins-offload on ethtool -K hms0p0/
hms0p1 hsr-tag-rm-offload on
```

2. Create an HSR interface and add slave interfaces to it.

```
ip link add name hsr0 type hsr slave1 hms0p0 slave2 hms0p1 supervision 45
 version 1
```

3. Add an IP address to the HSR interface.

```
ip addr add <IP_ADDR>/24 dev hsr0
```

4. Bring up the HSR interface.

```
ip link set hsr0 up
```

Send traffic from PC1 to PC2, or to the hsr0 interface of any board. The traffic drops to zero packets when any link of the ring is disconnected or connected.

6.11 Digital Encoder

6.11.1 Overview

An encoder is a type of device used to give feedback on the speed, position, angular position, and direction of an object.

- **Absolute Encoder**

Absolute Encoder outputs a unique code for each position, ensuring accurate feedback regardless of power interruptions or resets. This capability makes encoder absolute highly reliable in applications where maintaining accurate position data is critical.
The Absolute Encoder is commonly used in industries such as automation, robotics, aerospace, medical equipment, and more, where high precision and real-time position feedback are required.

- **Incremental Encoder**
  Incremental Encoder tracks movement using pulse counts. They generate consistent pulses that allow this type of encoder to measure how fast and how much rotation or linear movement occurs relative to where it started.

Encoder interfaces are used to communicate between the encoder and electronics (drive, control or some of readout). Unlike incremental encoders using pulses, absolute encoders are serialized and communicate in bits and bytes. The below encoder interfaces are supported in this release:

- BiSS
- EnDat2.2
- EnDat3.0
- HIPERFACE DSL
- Nikon A-Format
- Tamagawa T-Format

Table 108 shows the support of all kinds of encoder interfaces on different platforms:

Table 108. Encoder interfaces supported on different platforms

Encoder interface	i.MX 943 EVK-Mcore	MIMXRT1180-EVK
BiSS	Y	-
EnDat2.2	Y	-
EnDat3.0	Y	-
HIPERFACE DSL	Y	-
Nikon A-Format	Y	Y
Tamagawa T-Format	Y	Y

6.11.2 Feature release

6.11.2.1 BiSS

- BiSS-C Protocol support
- Point-to-Point connection support
- Daisy chain connection support with up to 8 slaves
- Interface speed up to 10MHZ
- Line-Delay compensation support
- Control Communication support
- Single-Cycle Data up to 64bits for each slave including position data, error and warning flag.
- Slaves register operations during cyclic data transfers
- Trigger event: extern signal / program / a constant rate / salve’s timeout
- Data verification by CRC polynomials of up to 16 bits per slave
- EOT interrupt support

6.11.2.2 EnDat2.2

- EnDat2.2 command set support
- EnDat2.1 command set support
- Transmission Rate up to 16 MHZ
- Propagation delay compensation
- Strobe source: hardware strobe / Software strobe / timer strobe
- Channel selection
- Safety Readiness: Recovery time
- Interrupt support
- Status support including Error1/ Error2/ WRN/ RM/ Busy
- Cyclical Redundance Check (CRC) support

6.11.2.3 EnDat3.0

- FG request
  - Provides a set of low-level API interfaces to send and process all FG requests and associated responses.
  - Supports Software Strobe mode.
  - Suppoorts Extern trigger signal strobe mode (Sync mode).
- BG request
  - Provides a stack to process all BG requests and associated responses.
- Bus Operation
  - Support Broadcast and Peer-to-Peer communication.
  - Provide an address assignment algorithm to assign and clean the address for all participants.
- Memory accessing
  - Provide a set of low-level API interfaces to read and write the memory areas.
  - Provide memory cache mechanism to support cache fetch, modification and write-back operations.
  - Provide API interfaces to support LPF configuration for LPSET and LPLIVE.
- Safety support
  - Support to configurate the collection of up to two different items of safety information.

  - Provide a stack to process the safety packets getting from SAFETY_COLLECTOR_MEM.
- FID-based memory
  - Provide a stack to process the LPF data getting from FID_BASED_MEM.

6.11.2.4 HIPERFACE DSL

- Resource Database (RDB) access:
  - Provides access to the stack for RDB reading and writing.
  - Provides API interfaces to scan and inquire about RDB entries.
- Long Messages
  - Provides a set of low-level API interfaces to start a “Long Message” transaction to exchange the general parameter data.
- Short Messages
  - Remote (DSL motor feedback system) registers that indicate interface information are mirrored via short messages. A set of API interfaces is provided to access these remote registers.
- Sync mode
  - Supports Extern trigger mode.
- Safety functions
  - Provides a set of API interfaces to configure and handle the events associated with Safety functions

6.11.2.5 Nikon A-Format

- Hardware interface
- Automatically switches the read and write directions of RS-485
- Supports encoders with different baud rates
  - 2.5 Mbit/s
  - 4 Mbit/s
  - 6.67 Mbit/s
  - 8 Mbit/s
  - 16 Mbit/s
- Connection types:
  - One-to-one connection
  - Bus connection: Up to 8 units
- Communication Mode
  - Individual transmission mode
    - Only the encoder that matched in the comparison sends the data.
    - Available connection types for the transmission line are one-to-one connection and bus connection.
  - Multiple transmission modes
    - Multiple encoders consecutively send the data according to the set encoder address.
    - The available connection type for the transmission line is bus connection.

- The modes of sending commands to the encoder
  - External signal trigger
  - Call the sending API directly

- The modes of receiving data from the encoder
  - Interrupt mode
  - Polling mode
  - Support CRC verification
  - Access the encoder EEPROM

  - Provide a set of API interfaces to configure and get the encoder information
  - Setting the encoder address
  - Reading out the multi turn and single turn data
  - Reading out the encoder status code
  - Reading out the encoder temperature data

6.11.2.6 Tamagawa T-Format

- Hardware interface
- Automatically switches the read and write directions of RS-485
- Only one encoder can be connected. The modes of sending commands to the encoder are:
  - External signal trigger
  - Call the sending API directly
- The modes for receiving data from the encoder are:
  - Interrupt mode
  - Polling mode
- Supports CRC verification
- Accesses the encoder EEPROM
- Provides a set of API interfaces to get the encoder information by
  - Reading out the multi turn and single turn data
  - Reading out the encoder status code
  - Reading out the encoder temperature data

6.11.3 Hardware modules

This section describes the associated hardware modules with encoders demo setup.

6.11.3.1 i.MX 943 EVK

The i.MX 943 EVK board offers a rich set of peripherals targeting industrial, automotive telematics, and consumer IoT market segments. The i.MX 943 applications processor integrates up to four Arm Cortex-A55 cores and supports functional safety with built-in 2x Arm Cortex-M33 and Cortex-M7 cores.

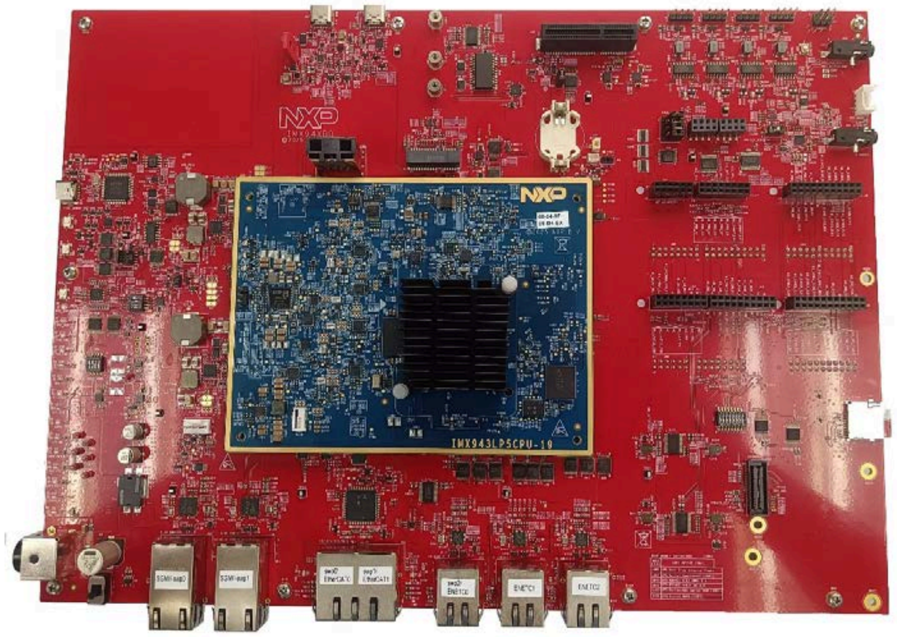


Figure 218. i.MX 943 EVK board

6.11.3.2 FRDM-LVPMSM-FA

The FRDM-LVPMSM-FA shield board implements a 3-phase Permanent Magnet Synchronous Motor (PMSM) interface platform that adds Field Oriented Control (FOC) motor control capabilities, such as rotational or linear motion.

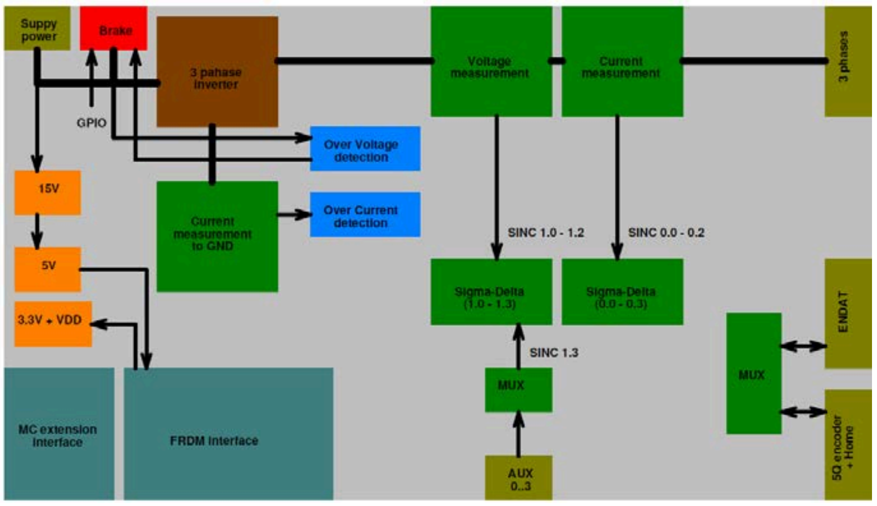


Figure 219. FRDM-LVPMSM-FA Block Diagram

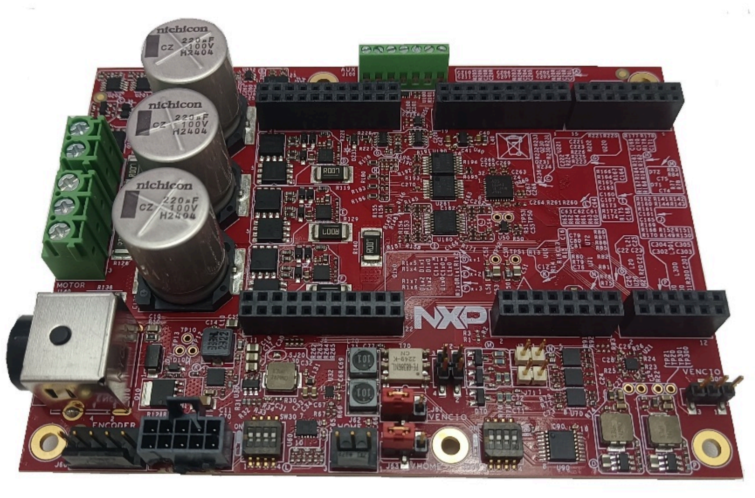


Figure 220. FRDM-LVPMSM-FA Board

Switches SW30 and SW90 must be set with regard to the encoder type used.

- SW30
  Select the appropriate voltage based on the properties of the encoder sensor.

Table 109. Switch setting for FRDM-LVPMSM-FA board

SW Position	State	Description
SW [1]	ON	3.3 V
SW [2]	ON	5.2 V
SW [3]	ON	9 V
SW [4]	ON	12 V

- SW90
  Route the encoder signal wires connection based on the used encoder.

Table 110. Switch settings and description

SW Position	State	Description
SW [1:2]	ON: OFF	Full duplex
	OFF: ON	Half duplex
	ON: ON	Invalid settings
	ON: OFF	Invalid settings
SW [3]	OFF	Echo enabled
	ON	Echo disabled
SW [4]	OFF	ENC is used

Table 110. Switch settings and description*...continued*

SW Position	State	Description
	ON	ENDAT is used

HIPERFACE DSL and EnDat3.0 interfaces support 2-wires mode. The FRDM-LVPMSM-FA jumpers below must be shorted if 2-wires mode is used.

- J72: pin2 <--> pin 1 (pin2 must be shorted to pin1)
- J72: pin4 <--> pin 3 (pin4 must be shorted to pin3)
- J73: pin2 <--> pin 1 (pin2 must be shorted to pin1)

6.11.3.3 MIMXRT1180-EVK

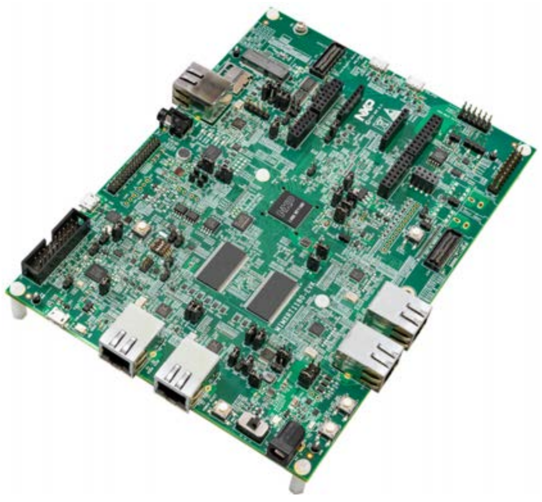


Figure 221. MIMXRT1180-EVK board

The i.MX RT1180 evaluation kit provides a powerful, extendable platform to support most features of the i.MX RT1180 Crossover MCU, which is powered by the Arm Cortex-M7 and Arm Cortex-M33 cores. Five Ethernet ports are provided for Ethernet or TSN switches, endpoints, and EtherCAT applications. An Arduino UNO site is provided for expansion using NXP or 3rd party shield boards.

6.11.3.4 RS-485 adapter board

This adapter is used to convert TTL to RS-485 differential signaling and has two 4-pin headers on the assembly.

- **1 x 4 Header (Data side)**
- **RO**=Receiver Output. Connects to a serial RX pin on the microcontroller.
- **RE**= Receiver Enable. Active low. Connects to a digital output pin on a microcontroller. Drive LOW to enable the receiver, high to enable the driver.
- **DE**=Driver Enable. Active high. Typically, it is shorted to the RE pin.
- **DI=**Driver Input. Connects to the serial TX pin on the microcontroller.
- **1 x 4 Header (Output side)**
- **VCC=** 3.3 V
- **B=** Data ‘B’ Inverted Line. Common with the B.
- **A**= Data ‘A’ Non-Inverted Line. Connects to A on the far end module.
- **GND=**Ground

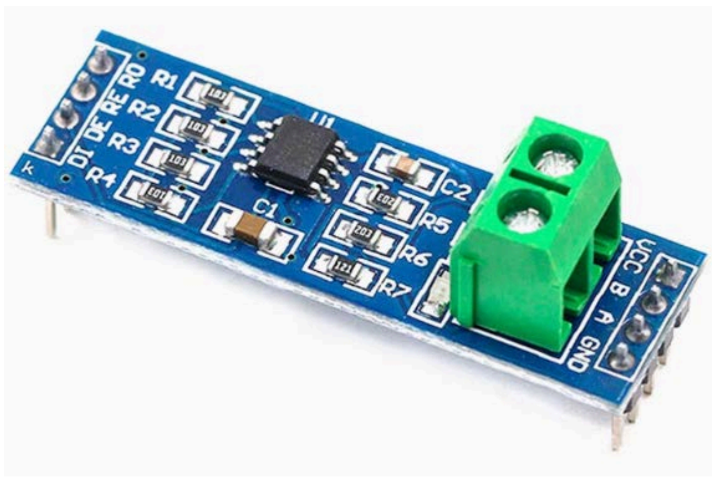


Figure 222. RS-485 Adapter Board

6.11.4 BiSS on i.MX 943 EVK platform

6.11.4.1 Hardware requirements

Table 111 lists the hardware modules which are required for BiSS encoders on i.MX 943 EVK platform:

Table 111. BiSS encoders on i.MX 943 EVK platform

Hardware	Type	Note
Encoder	HM16	Build-in the Heidrive HMD06 motor
Adapter	FRDM-LVPMSM-FA	-

6.11.4.2 Hardware assembly

For the BiSS, the FRDM-LVPMSM-FA board is used to convert TTL to RS-485 signals. Below are the steps describing how to set up the hardware environment.

- Connect the FRDM-LVPMSM-FA shield Motor Control 2 connector of the i.MX943-EVK board.
- Plug in the USB cable from the USB host to the FTDI_DEBUG USB connector, J15, on the i.MX943-EVK board.
- Plug the 12-V DC power supply to the P1 connector on the i.MX943-EVK board.
- Plug the 24-V DC power supply to the J10 connector on the FRDM-LVPMSM-FA board.
- Set the FRDM-LVPMSM-FA switches as below:

Table 112. FRDM-LVPMSM-FA switch settings

Encoder interface	SW30 [1-4]	SW90 [1-4]
BiSS	OFF-OFF-ON-OFF	ON-OFF-OFF-ON

- Connect the encoder signal wires to the J70 connector on the FRDM-LVPMSM-FA board.

For a **Point-to-Point connection**, use the settings listed below:

Table 113. Settings for Point-to-Point connection

Pin Name	J70 [0-9]	Signal Name
ENC_CLK_P	J70 [2]	MASTER_CLOCK_INPUT_P
ENC_CLK_N	J70 [7]	MASTER_CLOCK_INPUT_N
ENC_DATA_IN_P	J70 [3]	SLAVE_DATA_P
ENC_DATA_IN_N	J70 [8]	SLAVE_DATA_N
VENC	J70 [1]	VDC - Power supply voltage
GND	J70 [6]	GND

For a **Daisy-chain connection**, use the settings below:

Table 114. Settings for Daisy-chain connection

Pin Name	J70 [0-9]	Signal Name
ENC_CLK_P	J70 [2]	MASTER_CLOCK_INPUT_P
ENC_CLK_N	J70 [7]	MASTER_CLOCK_INPUT_N
ENC_DATA_IN_P	J70 [3]	SLAVE_DATA_P

Table 114. Settings for Daisy-chain connection...*continued*

Pin Name	J70 [0-9]	Signal Name
ENC_DATA_IN_N	J70 [8]	SLAVE_DATA_N
ENC_DATA_IO_P	J70 [4]	SLAVE_DATA_INPUT_P
ENC_DATA_IO_N	J70 [9]	SLAVE_DATA_INPUT_N
VENC	J70 [1]	VDC - Power supply voltage
GND	J70 [6]	GND

6.11.4.3 Quick evaluation

6.11.4.3.1 Pre-built image download

The pre-built images based on the i.MX 943 EVK Platform can be downloaded from the link below.

https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/tree/walnascar/recipes-kernel/mcux-kernel/industrial-examples_imx943

Table 115. Pre-built images for i.MX 943 EVK platform

Pre-built images
biss_cm7_core1.bin_lpddr4_flash.bin
biss_cm33_core1.bin_lpddr4_flash.bin
biss_cm7_core1.bin_lpddr5_flash.bin
biss_cm33_core1.bin_lpddr5_flash.bin

6.11.4.3.2 Running the image

Use the below steps to run the image:

- Burn `flash.bin` to MicroSD at 32 K(0x8000) offset using the `dd` command.

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

  Note: *Flash.bin is the pre-built image and the `/dev/sdx` is the SD card device on your PC.*
- Plug the MicroSD card to the board.
- Change the boot mode to SW4[1:4] = x011 for SD boot.
- Enable MCU UARTs.
  Follow the description on the link Enable MCU UARTs — MCUXpresso SDK Documentation to setup the UART.
- Power on the board.
  Refer to the Getting Started with Package — MCUXpresso SDK Documentation for details.

6.11.5 EnDat2.2 on i.MX 943 EVK Platform

6.11.5.1 Hardware requirements

The below lists the hardware modules which are required for EnDat2.2 encoders on i.MX 943 EVK platform:

Table 116. Hardware modules for EnDat2.2 encoders on i.MX 943 EVK platform

Hardware	Type	Note
Encoder	EQI1331	EnDat2.2 encoder
Adapter	FRDM-LVPMSM-FA	

6.11.5.2 Hardware assembly

For the Endat2.2, the FRDM-LVPMSM-FA board is used to convert TTL to RS-485 signals. Below are the steps to set up the hardware environment.

- Connect the FRDM-LVPMSM-FA Shield Motor Control to the connector of the i.MX 943 EVK board.
- Plug the USB cable from the USB host to the FTDI_DEBUG USB connector J15 on the i.MX 943 EVK board.
- Plug the 12-V DC power supply to the P1 connector on the i.MX 943 EVK board.
- Plug the 24-V DC power supply to the J10 connector on the FRDM-LVPMSM-FA board.
- FRDM-LVPMSM-FA switches must be set as described in Table 117:

Table 117. FRDM-LVPMSM-FA switch settings

Encoder interface	SW30 [1-4]	SW90 [1-4]
EnDat2.2	OFF-OFF-ON-OFF	OFF-ON-OFF-ON

- Connect the encoder signal wires to the J70 connector on the FRDM-LVPMSM-FA board.

Table 118. Encoder to FRDM-LVPMSM-FA board connection

Pin Name	J70 [0-9]	Signal Name
ENC_CLK_P	J70 [2]	CLOCK_P
ENC_CLK_N	J70 [7]	CLOCK_N
ENC_DATA_IO_P	J70 [4]	DATA_P
ENC_DATA_IO_N	J70 [9]	DATA_N
VENC	J70 [1]	VDC - Power supply voltage
GND	J70 [6]	GND

6.11.5.3 Quick evaluation

6.11.5.3.1 Pre-built image download

The pre-built images based on i.MX 943 EVK Platform can be downloaded from the link below.

https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/tree/walnascar/recipes-kernel/mcux-kernel/industrial-examples_imx943

Table 119. Pre-built images based on i.MX 943 EVK

Pre-built image
endat2p2_cm7_core1.bin_lpddr4_flash.bin

Table 119. Pre-built images based on i.MX 943 EVK*...continued*

Pre-built image
endat2p2_cm33_core1.bin_lpddr4_flash.bin
endat2p2_cm7_core1.bin_lpddr5_flash.bin
endat2p2_cm33_core1.bin_lpddr5_flash.bin

6.11.5.3.2 Running the image

Use the below steps to run the image:

- Burn `flash.bin` to MicroSD at 32 K(0x8000) offset using the `dd` command.

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

Note: *Flash.bin is the pre-built image. And the /dev/sdx is the SD card device on your PC.*

- Plug the MicroSD card to the board.
- Change the boot mode to SW4[1:4] = x011 for SD boot.
- Enable MCU UARTs.
  Follow the description on the link Enable MCU UARTs — MCUXpresso SDK Documentation to setup the UART.
- Now, power on the board.
  Refer to Getting Started with Package — MCUXpresso SDK Documentation for details.

6.11.6 EnDat3 on i.MX 943 EVK Platform

6.11.6.1 Hardware requirements

The below lists the hardware modules that are required for EnDat3 encoders on i.MX 943 EVK platform:

Table 120. Hardware modules for EnDat3 encoders on i.MX 943 EVK platform

Hardware	Type	Note
Encoder	MRS2231	EnDat3 Encoder
Adapter	FRDM-LVPMSM-FA	

6.11.6.2 Hardware assembly

For the Endat3 interfaces, the FRDM-LVPMSM-FA board is used to convert TTL to RS-485 signals. Below are the steps to setup the hardware environment for these encoder interfaces.

- Connect the FRDM-LVPMSM-FA shield Motor Control 2 connector of the i.MX 943 EVK board.
- Plug the USB cable from the USB host to the FTDI_DEBUG USB connector J15 on the i.MX 943 EVK board.
- Plug the 12-V DC power supply to the P1 connector on the i.MX 943 EVK board.
- Plug the 24-V DC power supply to the J10 connector on the FRDM-LVPMSM-FA board.
- FRDM-LVPMSM-FA switches must be set as below:

Table 121. FRDM-LVPMSM-FA switch settings

Encoder interface	SW30 [1-4]	SW90 [1-4]
EnDat3	OFF-OFF-ON-OFF2	OFF-ON-OFF-ON

- Connect the encoder signal wires to the J70 connector on the FRDM-LVPMSM-FA board:

Table 122. Encoder to FRDM-LVPMSM-FA board connection

Pin Name	J70 [0-9]	Signal Name
ENC_DATA_IO_P	J70 [4]	DATA_P
ENC_DATA_IO_N	J70 [9]	DATA_N
VENC	J70 [1]	VDC - Power supply voltage
GND	J70 [6]	GND

6.11.6.3 Quick evaluation

6.11.6.3.1 Pre-built image download

The pre-built images based on i.MX 943 EVK platform can be downloaded from the link below.

https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/tree/walnascar/recipes-kernel/mcux-kernel/industrial-examples_imx943

Table 123. Pre-built images based on i.MX 943 EVK platform

Pre-built image
endat3_point2point_cm7_core1.bin_lpddr4_flash.bin
endat3_point2point_cm33_core1.bin_lpddr4_flash.bin
endat3_point2point_cm7_core1.bin_lpddr5_flash.bin

Table 123. Pre-built images based on i.MX 943 EVK platform*...continued*

Pre-built image
endat3_point2point_cm33_core1.bin_lpddr5_flash.bin

6.11.6.3.2 Running the image

Use the below steps to run the image:

- Burn `flash.bin` to MicroSD at 32 K(0x8000) offset using the `dd` command.

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

Note: *`flash.bin` is the pre-built image. And the `/dev/sdx` is the SD card device on your PC.*

- Plug the MicroSD card to the board.
- Change the boot mode to SW4[1:4] = x011 for SD boot.
- Enable MCU UARTs.
  Please follow the description on the link Enable MCU UARTs — MCUXpresso SDK Documentation to setup the UART.
- Power on the board.
  Refer to Getting Started with Package — MCUXpresso SDK Documentation for details.

6.11.7 HIPERFACE DSL on i.MX 943 EVK Platform

6.11.7.1 Hardware Requirements

The below lists the hardware modules which are required for HIPERFACE encoders on i.MX 943 EVK platform:

Table 124. Hardware modules for HIPERFACE encoders on i.MX 943 EVK

Hardware	Type	Note
Encoder	EDM35	HIPERFACE DSL Encoder, Two-wires mode
Adapter	FRDM-LVPMSM-FA	

6.11.7.2 Hardware assembly

For the HIPERFACE DSL interfaces, the FRDM-LVPMSM-FA board is used to convert TTL to RS-485 signals. Below are the steps how to set up the hardware environment for these encoder interfaces.

- Connect the FRDM-LVPMSM-FA Shield Motor Control 2 connector of the i.MX 943 EVK board.
- Plug the host USB cable to the FTDI_DEBUG USB connector J15 on the i.MX 943 EVK board.
- Plug the 12-V DC power supply to the P1 connector on the i.MX 943 EVK board.
- Plug the 24-V DC power supply to the J10 connector on the FRDM-LVPMSM-FA board.
- FRDM-LVPMSM-FA switches must be set as below:

Table 125. Switch settings on FRDM-LVPMSM-FA board

Encoder interface	SW30 [1-4]	SW90 [1-4]
HIPERFACE DSL	OFF-OFF-ON-OFF	OFF-ON-OFF-ON

- Connect the encoder signal wires to the J70 connector on the FRDM-LVPMSM-FA board:

Table 126. Encoder connection to the FRDM-LVPMSM-FA board

Pin Name	J70 [0-9]	Signal Name
ENC_DATA_IO_VENC_P	J70 [5]	DATA_P
ENC_DATA_IO_VENC_N	J70 [10]	DATA_N

- HIPERFACE DSL works in 2-wires mode. The FRDM-LVPMSM-FA jumpers below must be shorted.
  - J72: pin2 <--> pin 1
  - J72: pin4 <--> pin 3
  - J73: pin2 <--> pin 1

6.11.7.3 Quick Evaluation

6.11.7.3.1 Pre-built image download

The pre-built images based on i.MX 943 EVK Platform can be downloaded from the link below.

https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/tree/walnascar/recipes-kernel/mcux-kernel/industrial-examples_imx943

Table 127. Pre-built images on i.MX 943 EVK Platform

Pre-built images
hiperface_cm7_core1.bin_lpddr4_flash.bin

Table 127. Pre-built images on i.MX 943 EVK Platform *...continued*

Pre-built images
hiperface_cm33_core1.bin_lpddr4_flash.bin
hiperface_cm7_core1.bin_lpddr5_flash.bin
hiperface_cm33_core1.bin_lpddr5_flash.bin

6.11.7.3.2 Running the image

Use the below steps to run the image:

- Burn `flash.bin` to MicroSD at 32 K(0x8000) offset using the `dd` command.

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

  Note: *`Flash.bin` is the pre-built image and the `/dev/sdx` is the SD card device on your PC.*
- Plug the MicroSD card to the board.
- Change the boot mode to SW4[1:4] = x011 for SD boot.
- Enable MCU UARTs.
  Follow the description on the link Enable MCU UARTs — MCUXpresso SDK Documentation to setup the UART.
- Power on the board.
  Refer to the Getting Started with Package — MCUXpresso SDK Documentation for details.

6.11.8 Tamagawa T-Format on i.MX 943 EVK Platform

6.11.8.1 Hardware requirements

The below lists the hardware modules which are required for Tamagawa T-Format encoders on i.MX 943 EVK platform:

Table 128. Hardware requirement for Tamagawa T-Format encoders on i.MX 943 EVK

Hardware	Type	Note
Encoder	TS5700N8501	Tamagawa T-Format Encoder
Adapter	RS-485 Adapter Board	

6.11.8.2 Hardware assembling

The Tamagawa T-Format interface is implemented based on FlexIO. An RS-485 adapter is required to convert the TTL to RS-485. Below are the connections:

- Connect the RS-485 Adapter Board to the J48 connector on the i.MX 943 EVK board.

Table 129. RS-485 Adapter Board connection to i.MX 943 EVK board

Pin Name	i.MX 943 EVK	RS-485 Adapter Board
FLEXIO_A_FORMAT_DR	J48-10	DE and RE
FLEXIO_A_FORMAT_TX	J48-9	RO
FLEXIO_A_FORMAT_RX	J48-11	DI
VCC 3.3V	J49-8	VCC
GND	J49-12	GND

- Connect the RS-485 Adapter board to the specific encoder.

Table 130. RS-485 Adapter board and encoder connection

Pin Name	RS-485 Adapter Board	Encoder
RS485-A	A	SD+
RS485-B	B	SD-

- Power supply for encoder.
  The power supply voltage of Tamagawa T-format and Nikon A-format encoder used in the demo is +5V.

Table 131. i.MX 943 EVK and Encoder voltages

Pin Name	i.MX 943 EVK	Encoder
VCC	J49-10	VCC
GND	J49-14	GND

6.11.8.3 Quick evaluation

6.11.8.3.1 Pre-built image download

The pre-built images based on i.MX 943 EVK platform can be downloaded from the link below.

https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/tree/walnascar/recipes-kernel/mcux-kernel/industrial-examples_imx943

Table 132. pre-built images based on i.MX 943 EVK platform

Pre-built image
t_format_interrupt_transfer_cm7_core1.bin_lpddr4_flash.bin
t_format_interrupt_transfer_cm33_core1.bin_lpddr4_flash.bin
t_format_interrupt_transfer_cm7_core1.bin_lpddr5_flash.bin
t_format_interrupt_transfer_cm33_core1.bin_lpddr5_flash.bin
t_format_polling_transfer_cm7_core1.bin_lpddr4_flash.bin
t_format_polling_transfer_cm33_core1.bin_lpddr4_flash.bin
t_format_polling_transfer_cm7_core1.bin_lpddr5_flash.bin
t_format_polling_transfer_cm33_core1.bin_lpddr5_flash.bin

6.11.8.3.2 Running the image

Use the below steps to run the image:

- Burn `flash.bin` to MicroSD at 32 K(0x8000) offset using the `dd` command.

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

Note: *Flash.bin is the pre-built image. And the /dev/sdx is the SD card device on your PC.*
- Plug the MicroSD card to the board.
- Change the boot mode to SW4[1:4] = x011 for SD boot.
- Enable MCU UARTs.
  Follow the description on the link Enable MCU UARTs — MCUXpresso SDK Documentation to setup the UART.
- Power on the board.
  Refer to the Getting Started with Package — MCUXpresso SDK Documentation for details.

6.11.9 Nikon A-Format on i.MX 943 EVK platform

6.11.9.1 Hardware requirements

The below table lists the hardware modules which are required for Nikon A-Format encoders on i.MX 943 EVK platform:

Table 133. Hardware modules required for Nikon A-Format encoders on i.MX 943 EVK

Hardware	Type	Note
Encoder	M50AHN00	Nikon A-Format Encoder
Adapter	RS-485 Adapter Board	

6.11.9.2 Hardware assembly

The Nikon A-Format interfaces is implemented based on FlexIO. An RS-485 adapter is required to convert the TTL to RS-485. Below are the connections:

- Connect the RS-485 Adapter board to the J48 connector on i.MX 943 EVK board.

Table 134. RS-485 Adapter board to i.MX 943 EVK board connections

Pin Name	i.MX 943 EVK	RS-485 Adapter Board
FLEXIO_A_FORMAT_DR	J48-10	DE and RE
FLEXIO_A_FORMAT_TX	J48-9	RO
FLEXIO_A_FORMAT_RX	J48-11	DI
VCC 3.3V	J49-8	VCC
GND	J49-12	GND

- Connect the RS-485 Adapter Board to the specific encoder.

Table 135. RS-485 Adapter Board to Encoder connection

Pin Name	RS-485 Adapter Board	Encoder
RS485-A	A	SD+
RS485-B	B	SD-

- Power supply for encoder.
  The power supply voltage of Tamagawa T-format and Nikon A-format encoder used in the demo is +5 V.

Table 136. Power supply voltage for i.MX 943 EVK board

Pin Name	i.MX 943 EVK	Encoder
VCC	J49-10	VCC
GND	J49-14	GND

6.11.9.3 Quick evaluation

6.11.9.3.1 Pre-built image download

The pre-built images based on i.MX 943 EVK platform can be downloaded from the link below.

https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/tree/walnascar/recipes-kernel/mcux-kernel/industrial-examples_imx943

Table 137. Pre-built images for i.MX 943 EVK

Pre-built image
a_format_interrupt_transfer_cm7_core1.bin_lpddr4_flash.bin
a_format_interrupt_transfer_cm33_core1.bin_lpddr4_flash.bin
a_format_interrupt_transfer_cm7_core1.bin_lpddr5_flash.bin
a_format_interrupt_transfer_cm33_core1.bin_lpddr5_flash.bin
a_format_polling_transfer_cm7_core1.bin_lpddr4_flash.bin
a_format_polling_transfer_cm33_core1.bin_lpddr4_flash.bin
a_format_polling_transfer_cm7_core1.bin_lpddr5_flash.bin
a_format_polling_transfer_cm33_core1.bin_lpddr5_flash.bin

6.11.9.3.2 Running the image

Use the below steps to run the image:

- Burn `flash.bin` to MicroSD at 32 K(0x8000) offset using the `dd` command.

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

  Note: *Flash.bin is the pre-built image and the `/dev/sdx` is the SD card device on your PC.*
- Plug the MicroSD card to the board.
- Change the boot mode to SW4[1:4] = x011 for SD boot.
- Enable MCU UARTs.
  Follow the description on the link Enable MCU UARTs — MCUXpresso SDK Documentation to setup the UART.
- Now, power on the board.
  Refer to the Getting Started with Package — MCUXpresso SDK Documentation for details.

6.11.10 Tamagawa T-Format on MIMXRT1180-EVK Platform

6.11.10.1 Hardware requirements

The below hardware modules are required on MIMXRT1180-EVK platform:

Table 138. Hardware modules for MIMXRT1180-EVK platform (Tamagawa T-Format)

Hardware	Type	Note
Encoder	TS5700N8501	Tamagawa T-Format Encoder
Adapter	RS-485 Adapter Board	

6.11.10.2 Hardware assembly

1. The Tamagawa T-Format interfaces are also implemented based on FlexIO. An RS-485 adapter is required to convert the TTL to RS-485. Below are the connections:
2. Connect the RS-485 Adapter board to the J48 connector on the MIMXRT1180-EVK board.

Table 139. RS-485 Adapter and MIMXRT1180-EVK board connections

Pin Name	MIMXRT1180-EVK	RS-485 Adapter Board
FLEXIO_A_FORMAT_DR	J48-10	DE and RE
FLEXIO_A_FORMAT_TX	J48-9	RO
FLEXIO_A_FORMAT_RX	J48-11	DI
VCC 3.3V	J49-8	VCC
GND	J45-12	GND

3. Connect the RS-485 Adapter board to the specific encoder.

Table 140. RS-485 Adapter board connection to the encoder

Pin name	RS-485 Adapter board	Encoder
RS485-A	A	SD+
RS485-B	B	SD-

4. Power supply for the encoder.
   The power supply voltage of Tamagawa T-format encoder used in the demo is +5 V.

Table 141. Connections to supply power to the encoder connected to MIMXRT1180-EVK

Pin Name	MIMXRT1180-EVK	Encoder
VCC	J45-16	VCC
GND	J45-14	GND

6.11.10.3 Quick evaluation

6.11.10.3.1 Pre-built image download

The pre-built images based on MIMXRT1180-EVK Platform can be downloaded in the link below:

https://github.com/nxp-real-time-edge-sw/meta-rtos-industrial/tree/walnascar/recipes-kernel/mcux-kernel/industrial-examples_evkmimxrt1180

Table 142. Pre-built images for MIMXRT1180-EVK (Tamagawa T-Format)

Pre-built image
t_format_interrupt_transfer_cm7.elf
t_format_interrupt_transfer_cm33.elf
t_format_polling_transfer_cm7.elf
t_format_polling_transfer_cm33.elf

6.11.10.3.2 Running the image

First, ensure that a standalone J-Link is connected to the debug interface of the MIMXRT1180-EVK board. After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

- Install SEGGER software, which can be downloaded from SEGGER.
- Connect the MINXRT1180-EVK board to the PC via USB cable between the OpenSDA USB connector and the PC USB connector.
- Enable the UART of the MCU with the settings below:
  - 115200 Baud Rate
  - No parity
  - 8 data bits
  - 1 stop bit

Refer to How to determine COM port.

- Open the J-Link GDB Server application on the PC.
  - Open Powershell terminal on the PC.
  - Change the folder to the SEGGER install folder. In general, the folder location is: *C:\Program Files(x86)\SEGGER\JLink_Vxxx*
  - To debug CM33, run the following command:

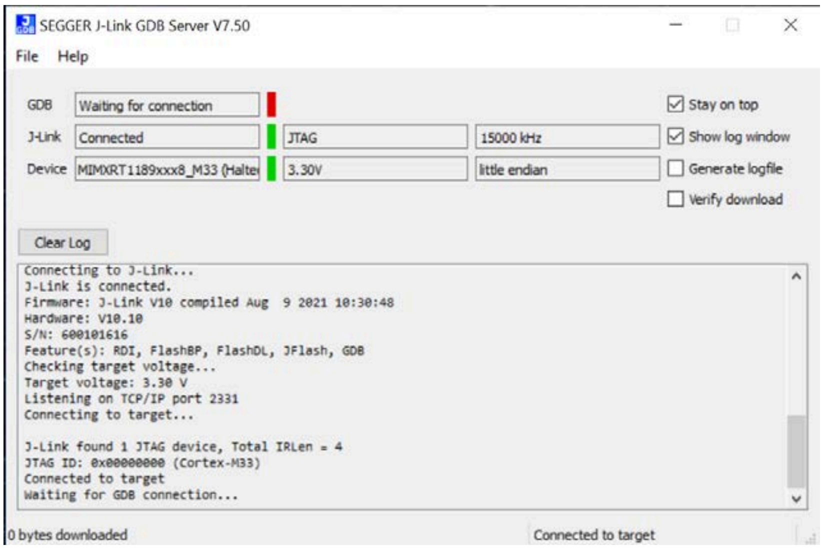
```
JLinkGDBServer.exe -device MIMXRT1189xxx8_M33 -if SWD -speed 4000 -
jlinkscriptfile <evkmimxrt1180_cm33.jlinkscript> -stayontop -ir
```

  - To debug CM7, run the following command:

```
JLinkGDBServer.exe -device MIMXRT1189xxx8_M7 -if SWD -speed 4000 -
jlinkscriptfile <evkmimxrt1180_cm7.jlinkscript> -stayontop -ir
```

Note: *The supporting jlinkscript file can be found in meta-rtos-industrial/recipes-kernel/mcux-kernel/industrial-examples_evkmimxrt1180 at walnascar · nxp-real-time-edge-sw/meta-rtos-industrial · GitHub*

- After it is connected, the screen must resemble the figure below:



SEGGER J-Link GDB Server V7.50

File Help

GDB: Waiting for connection

J-Link: Connected | JTAG | 15000 kHz

Device: MIMXRT1189xxx8_M33 (Halte | 3.30V | little endian

[✓] Stay on top
[✓] Show log window
[ ] Generate logfile
[ ] Verify download

Clear Log

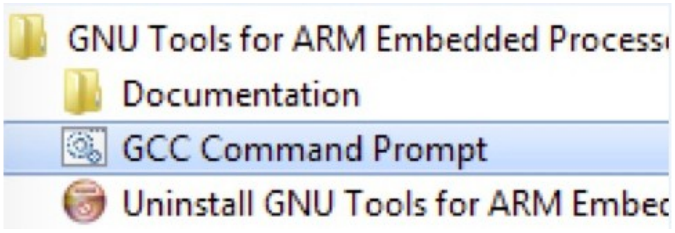
```
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Aug  9 2021 10:30:48
Hardware: V10.10
S/N: 600101616
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 3.30 V
Listening on TCP/IP port 2331
Connecting to target...

J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x00000000 (Cortex-M33)
Connected to target
Waiting for GDB connection...
```

0 bytes downloaded | Connected to target

Figure 223. J-Link Server window

- If GCC Arm Embedded tool chain is not installed on the PC, install it as below.
  - Download and run the installer from GNU Arm Embedded Toolchain. The GCC toolchain must correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes for MIMXRT1180-EVK* (document MCUXSDKMIMXRT118XKRN).
- Open a GCC ARM Embedded tool chain command window. Launch the window from the **Start menu** of the Windows operating system. Then, go to **Programs**GNU Tools ARM Embedded <version>**** and select **GCC Command Prompt**.

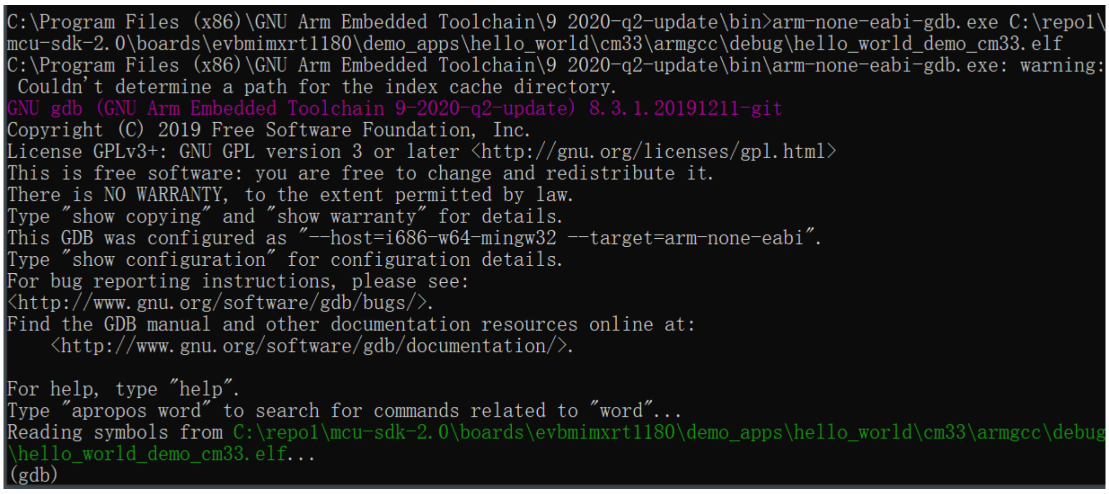
–



GNU Tools for ARM Embedded Process
Documentation
GCC Command Prompt
Uninstall GNU Tools for ARM Embec

Figure 224. GCC ARM Embedded tool chain command window

- Change to the directory that contains the pre-built image.
- Run the `arm-none-eabi-gdb.exe <pre_built_image_name>.elf`.



```
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin>arm-none-eabi-gdb.exe C:\repo1\
mcu-sdk-2.0\boards\evbmimxrt1180\demo_apps\hello_world\cm33\armgcc\debug\hello_world_demo_cm33.elf
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin\arm-none-eabi-gdb.exe: warning:
 Couldn't determine a path for the index cache directory.
GNU gdb (GNU Arm Embedded Toolchain 9-2020-q2-update) 8.3.1.20191211-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 C:\repo1\mcu-sdk-2.0\boards\evbmimxrt1180\demo_apps\hello_world\cm33\armgcc\debug
\hello_world_demo_cm33.elf...
(gdb)
```

Figure 225. Command log after running the image

- Run the commands below:
  - target remote localhost:2331
  - monitor reset
  - monitor halt
  - load
- The application is now downloaded and halted at the reset vector. Execute the `continue` or `monitor go` command to start the demo application.

For more details, refer to the Getting Started with Package — MCUXpresso SDK Documentation.

6.11.11 Nikon A-Format interfaces on MIMXRT1180-EVK platform

6.11.11.1 Hardware requirements

The below hardware modules are required on MIMXRT1180-EVK platform:

Table 143. Hardware modules for Nikon A-Format interfaces on MIMXRT1180-EVK

Hardware	Type	Note
Encoder	M50AHN00	Nikon A-Format Encoder
Adapter	RS-485 Adapter Board	

6.11.11.2 Hardware assembly

The Nikon A-Format interfaces is implemented based on FlexIO. An RS-485 adapter is required to convert the TTL to RS-485. Below are the connections:

1. Connect the RS-485 Adapter board to the J48 connector on MIMXRT1180-EVK board.

Table 144. RS-485 Adapter board connection to the MIMXRT1180-EVK board

Pin Name	MIMXRT1180-EVK	RS-485 Adapter Board
FLEXIO_A_FORMAT_DR	J48-10	DE and RE
FLEXIO_A_FORMAT_TX	J48-9	RO
FLEXIO_A_FORMAT_RX	J48-11	DI
VCC 3.3V	J49-8	VCC
GND	J45-12	GND

2. Connect the RS-485 Adapter board to the specific encoder.

Table 145. RS-485 Adapter board connection to the Encoder

Pin Name	RS-485 Adapter board	Encoder
RS485-A	A	SD+
RS485-B	B	SD-

3. Power supply for encoder.
The power supply voltage of the Nikon A-format encoder used in the demo is +5 V.

Table 146. Connection on MIMXRT1180-EVKto supply power to the Encoder used for the demo

Pin name	MIMXRT1180-EVK	Encoder
VCC	J45-16	VCC
GND	J45-14	GND

6.11.11.3 Quick evaluation

6.11.11.3.1 Prebuilt image download

The prebuilt images based on MIMXRT1180-EVK platform can be downloaded from the link below.

meta-rtos-industrial/recipes-kernel/mcux-kernel/industrial-examples_evkmimxrt1180 at walnascar · nxp-real-time-edge-sw/meta-rtos-industrial · GitHub

Table 147. Filename of prebuilt image (Nikon A-Format)

Prebuilt image filename
a_format_interrupt_transfer_cm7.elf
a_format_interrupt_transfer_cm33.elf
a_format_polling_transfer_cm7.elf
a_format_polling_transfer_cm33.elf

6.11.11.3.2 Running the image

First, ensure that a standalone J-Link is connected to the debug interface of the MIMXRT1180-EVK board. After the J-Link interface is configured and connected, follow these steps to download and run the demo applications:

- Install SEGGER software, which can be downloaded from SEGGER.
- Connect the MIMXRT1180-EVK board to the PC via USB cable between the OpenSDA USB connector and the PC USB connector.
- Enable MCU UARTs with these settings:
- 115200 Baud Rate
- No parity
- 8 data bits
- 1 stop bit

Refer to the How to determine COM port.

- Open the J-Link GDB Server application on the PC.
  - Open a Powershell terminal on the PC.
  - Change the folder to the SEGGER install folder. In general, the folder location is in *C:\Program Files(x86)\SEGGER\JLink_Vxxx*
  - To debug CM33, run the following command:

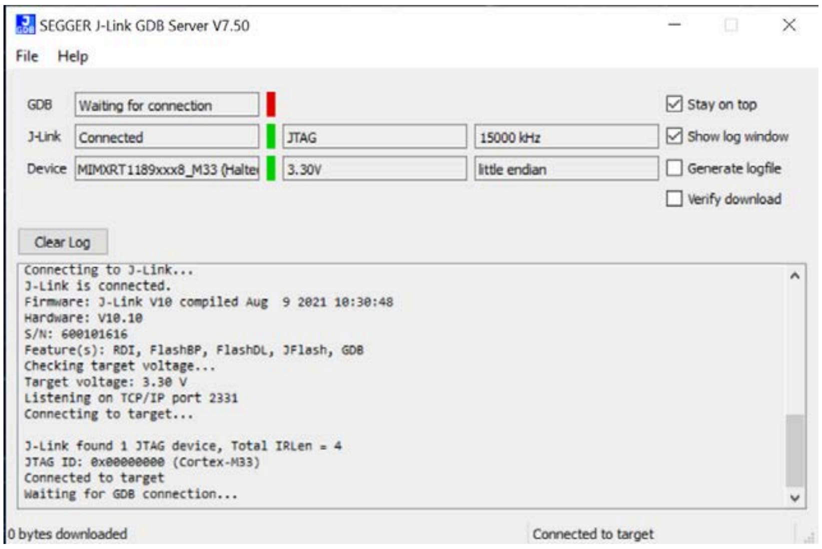
```
JLinkGDBServer.exe -device MIMXRT1189xxx8_M33 -if SWD -speed 4000 -
jlinkscriptfile <evkmimxrt1180_cm33.jlinkscript> -stayontop -ir
```

  - To debug CM7, run the following command:

```
JLinkGDBServer.exe -device MIMXRT1189xxx8_M7 -if SWD -speed 4000 -
jlinkscriptfile <evkmimxrt1180_cm7.jlinkscript> -stayontop -ir
```

Note: *The supporting jlinkscript file can be found in meta-rtos-industrial/recipes-kernel/mcux-kernel/industrial-examples_evkmimxrt1180 at walnascar · nxp-real-time-edge-sw/meta-rtos-industrial · GitHub*

- After it is connected, a window similar to the one shown below is displayed.



SEGGER J-Link GDB Server V7.50

GDB: Waiting for connection
J-Link: Connected | JTAG | 15000 kHz
Device: MIMXRT1189xxx8_M33 (Halte | 3.30V | little endian

Stay on top [✓], Show log window [✓], Generate logfile [ ], Verify download [ ]

```
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V10 compiled Aug  9 2021 10:30:48
Hardware: V10.10
S/N: 600101616
Feature(s): RDI, FlashBP, FlashDL, JFlash, GDB
Checking target voltage...
Target voltage: 3.30 V
Listening on TCP/IP port 2331
Connecting to target...

J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x00000000 (Cortex-M33)
Connected to target
Waiting for GDB connection...
```

0 bytes downloaded | Connected to target

Figure 226. J-Link Server window

- If the GCC Arm Embedded tool chain is not installed on the PC, install it as below.
  - Download and run the installer from the GNU Arm Embedded Toolchain. The GCC toolchain must correspond to the latest supported version, as described in *MCUXpresso SDK Release Notes for MIMXRT1180-EVK* (document MCUXSDKMIMXRT118XKRN).
- Open a ***GCC ARM Embedded*** tool chain command window. From the **Start menu** of the Windows operating system, go to **Programs**, select the **GNU Tools ARM Embedded <version>** and then select **GCC Command Prompt**.

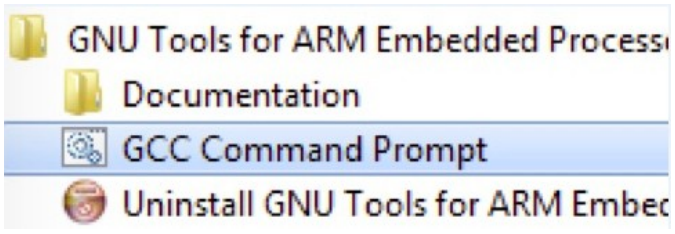
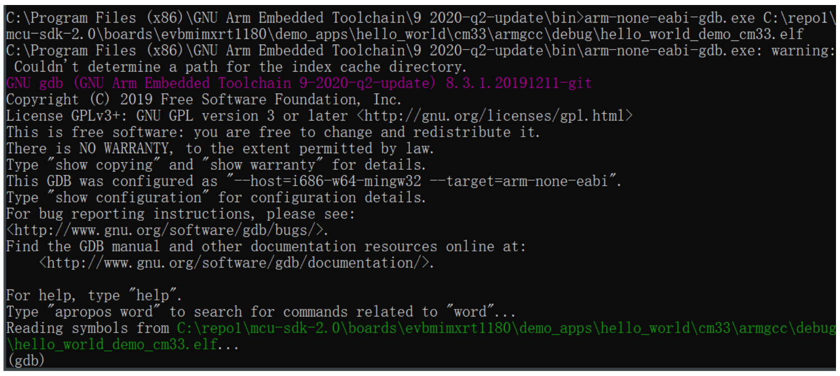

GNU Tools for ARM Embedded Process
Documentation
GCC Command Prompt
Uninstall GNU Tools for ARM Embec

Figure 227. Selecting GCC Command prompt

- Change to the directory that contains the prebuilt image.
- Run the `arm-none-eabi-gdb.exe <pre_built_image_name>.elf`.



```
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin>arm-none-eabi-gdb.exe C:\repo1\
mcu-sdk-2.0\boards\evbmimxrt1180\demo_apps\hello_world\cm33\armgcc\debug\hello_world_demo_cm33.elf
C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin\arm-none-eabi-gdb.exe: warning:
 Couldn't determine a path for the index cache directory.
GNU gdb (GNU Arm Embedded Toolchain 9-2020-q2-update) 8.3.1.20191211-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 C:\repo1\mcu-sdk-2.0\boards\evbmimxrt1180\demo_apps\hello_world\cm33\armgcc\debug
\hello_world_demo_cm33.elf...
(gdb)
```

Figure 228. Running the prebuilt image on MIMXRT1180-EVK log

- Run the commands below:
  - target remote localhost:2331

  - monitor reset
  - monitor halt
  - load
- The application is now downloaded and halted at the reset vector. To start the demo application, execute the `continue` or `monitor go` command.

For more details, Refer to the Getting Started with Package — MCUXpresso SDK Documentation.

6.11.12 Building the software

Follow the steps described below for Software building. All examples of digital encoder are integrated into MCUXpresso SDK 25.06. For the MCUX-SDK installation and procedure to set up the build environment, refer to **MCUXpresso SDK Documentation — MCUXpresso SDK Documentation.**

Table 148 lists the location of Encoder sample projects.

Table 148. Location of the Encoder sample projects

Encoder	Location
BiSS	examples/digital_encoder_examples/biss/
EnDat 2.2	examples/digital_encoder_examples/endat2p2/
EnDat 3.0	examples/digital_encoder_examples/endat3/
HIPERFACE DSL	examples/digital_encoder_examples/hiperface/
Nikon A-Format	examples/digital_encoder_examples/a-format/
Tamagawa T-Format	examples/digital_encoder_examples/t-format/

Table 149 lists the location of the Encoder drivers.

Table 149. Encoder formats and their driver paths

Encoder	Driver path
BiSS	drivers/biss/
EnDat 2.2	drivers/endat2p2/
EnDat 3.0	drivers/endat3/
HIPERFACE DSL	drivers/hiperface/
Nikon A-Format	drivers/flexio/a-format/
Tamagawa T-Format	drivers/flexio/t-format/

7 Acronyms and abbreviations

The Table 150 lists the acronyms used in this document.

Table 150. 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
DSA	Distributed switch architecture
EtherCAT	Ethernet for control automation technology
EVK	Evaluation kit
ECU	Electronic control units
FDB	Forwarding database
FQTSS	Forwarding and queuing enhancements for time-sensitive streams
FMan	Frame manager
GPU	General processor unit
HSR	High-availability Seamless Redundancy
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	Message Unit
NFC	Near field communication
NCI	NFC controller interface
NMT	Network management

Table 150. Acronyms and abbreviations*...continued*

Term	Description
OC	Ordinary clock
OpenIL	Open industry Linux
OPC-UA	Open platform communications unified architecture
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
RTC	Real-time clock
RTT	Round-trip times
RX	Receiver
SABRE	Smart application blueprint for rapid engineering
SDO	Service data object
SOEM	Simple Open EtherCAT master
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

Table 150. Acronyms and abbreviations*...continued*

Term	Description
TX	Transmitter
TZASC	Trust zone address space controller
UDP	User datagram protocol
VLAN	Virtual local area network

8 Note about the source code in the document

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

9 Revision history

Table 151 summarizes the revisions to this document.

Table 151. Revision history

Document ID	Release date	Description
REALTIMEEDGEUG v.3.3	15 December 2025	Updated for Real Time Edge Software 3.3 release.
REALTIMEEDGEUG v.3.2	31 July 2025	Updated for Real Time Edge Software 3.2 release. Added the sections. • Section 6.11 • Section 6.2.4 • Section 6.3.7.7
REALTIMEEDGEUG v.3.1	26 March 2025	Updated for Real Time Edge Software 3.1 release.
REALTIMEEDGEUG v.3.0	17 December 2024	Updated for Real Time Edge Software 3.0 release.
REALTIMEEDGEUG v.2.9	25 July 2024	• Updated for Real Time Edge Software Rev 2.9 release. • Added i.MX 93 14x14 EVK support.
REALTIMEEDGEUG v.2.8	29 March 2024	• Updated for Real Time Edge Software Rev 2.8 release. • 'What's New' section moved to *REALTIMEE DGERN*.
REALTIMEEDGEUG v. 2.7	18 December 2023	Updated for Real Time Edge Software 2.7 release. Added Section 4
REALTIMEEDGEUG v.2.6	28 July 2023	Updated for Real Time Edge Software 2.6 release.
		Added the Section Section 6.1.5.4
REALTIMEEDGEUG v.2.5	30 March 2023	Updated for Real Time Edge Software 2.5 release.
		Added the Section Section 3.6
		Added the Sections Section 6.1.5, Section 2.5.4.14
		Added the Section Section 3.5.6
		Modified the Sections: • Section 3.4.3 • Section 2.5.4.4
REALTIMEEDGEUG v.2.4	16 December 2022	Added support for i.MX 8DXL and i.MX 93 EVK boards throughout the document.
		Added the Section, "Configuring Flow Meter policy on stream".
		Added the Section Section 5.1.4.4.1
		Added the Section 2.5.4.15
		Added the Section Section 6.7.
		Added the Section Section 6.1.3.4.5
REALTIMEEDGEUG v.2.3	28 July 2022	Updated the Section, "What's new in Real-time Edge software v2.3".
		Added the Section Heterogeneous Multicore Framework.

Table 151. Revision history*...continued*

Document ID	Release date	Description
		Updated the Section Section 5.2
		Other updates throughout the document.
REALTIMEEDGEUG v.2.2	29 March 2022	Added the Section. Updated for Real-time Edge Software Rev 2.2. Support for TLS protocol removed for NETCONF feature.
REALTIMEEDGEUG v.2.1	15 December 2021	Added the Section. Updated for Real-time Edge Software Rev 2.1.
		Updated the Section, "Open, fixed, and closed issues"
REALTIMEEDGEUG v.2.0	29 July 2021	• '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*.
		Added the "What's new" Section.
		Integrated BareMetal framework in the document.
OPENILUG v.1.11	26 April 2021	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.
OPENILUG v.1.10	22 December 2020	Added the Chapter Section 5.2 and related contents.
		Added the 'Camera' Section and related details.
		Added the Host setup for i.MX 8M Plus EVK board details.
OPENILUG v.1.9	15 September 2020	Added the Section Section 5.3.
		Added the Chapter "GPU"and related description
		Added the Chapter "Wayland and Weston" and related description
		Made the section "The 'real-time-edge-servo' stack" a part of the Chapter "EtherCAT"
OPENILUG v.1.8	29 May 2020	Added the Section Section 2.4 in Section 6.
		Updated this Section Interface naming in Linux for LS1028ARDB.
		Updated the Section Host system requirements for Open IL.
		Updated the Section Running Selinux demo.
OPENILUG v.1.7.1	20 February 2020	Updated this Section Section 6.4.3.7.
OPENILUG v.1.7	17 January 2020	Added the Chapter (nxp servo) Section 6.1.3.4.

Table 151. Revision history*...continued*

Document ID	Release date	Description
		Added the Chapter Section 5.3.
		Updated the Section Getting Open IL
		Other updates in Section 6.4.
OPENILUG v.1.6	31 August 2019	• Updated Section 5.1.5. • Information related to pcpmap command removed from the Section Section 5.1.5.1 and Section 5.1.5.2. • Port names "eno/swp0" changed to "swp0" for a few tsntool commands. • Note added in Section Section "Stream identification" for usage of `nulltagged` and `streamhandle` parameters. • Added the Section Section 5.1.5.2.8. • Other minor updates.
		Updated the table "Host system mandatory packages". Added `autogen autoconf libtool` and `pkg-config` packages.
		Added this Chapter Section 6.5.3.
		Updated Section 6.4.
		• Added the Section Section 6.4.3.1 and other updates in Section 6.4.
OPENILUG v.1.5	01 May 2019	Added the Section to describe interface naming for U-Boot and Linux for LS1028ARDB. • Added the Chapter 'EdgeScale Client'. • Updated the OpenIL version and Git tag in the Section 'Getting Open IL'.
		Updated this Section in the Chapter Section 5.1.
OPENILUG v.1.4	01 February 2019	• 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 LS1028ARDB support. • Added the Chapter QT.
		Reorganized this Chapter and added separate Section for Section 5.1.5
		Minor updates in this Chapter. Also added the Section, Section 6.2.3.1 and Section 6.2.3.3.
OPENILUG v.1.3.1	15 October 2018	Updated the OpenIL version and Git tag in the Section 'Getting Open IL'.
OPENILUG v.1.3	31 August 2018	Added the Chapters.
		Added the Section in Chapter 'NXP OpenIL platforms'. Updated other Sections for i.MX6Q Sabre support.
		Updated the Section, "Getting OpenIL".

Table 151. Revision history*...continued*

Document ID	Release date	Description
		Added the Section, "Selinux demo" for enabling SELinux and updated Basic setup. Updates in other Sections.
OPENILUG v.1.2	31 May 2018	Updated the Section, "Hardware requirements" for RTnet.
		Updated the Section, "Software requirements" for RTnet.
OPENILUG v.1.1.1	18 April 2018	Added the Section, "RTnet".
		Added a note for LS1043A switch setting.
OPENILUG v.1.1	30 March 2018	Added support for industrial IoT BareMetal framework in this Section.
		Added a note for steps to be performed before booting up the board.
		Added the Section.Section 1.4.
OPENILUG v.1.0	22 Decemeber 2017	Added the Chapter Section 6.3.
		Section 5.1: Chapters for "1-board TSN demo" and "3-board TSN demo" replaced by a single Chapter, "TSN demo".
		In Section 6 • Updated the Section, 'Industrial Features'. • -IEEE 1588 -'sja1105-ptp' support removed.
OPENILUG v.0.3	25 August 2017	Set up the OpenIL website https://www.openil.org/.
		OTA - Xenomai Cobalt 64-bit and SJA1105 support added.
		Qbv support added in Section 5.1. • 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 5 added. Ubuntu Userland support for 64-bit LS1043ARDB and 64-bit LS1046ARDB added.
OPENILUG v.0.2	26 May 2017	Initial public release.

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

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

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents