UG10170

Harpoon User's Guide

Rev. 3.5 — 12 December 2025 **User guide**

Document information

Information	Content
Keywords	i.MX 8M, i.MX 9, HiFiBerry, MX93AUD-HAT, Arm Cortex-A53/A55 processor (Armv8-A architecture), RTOS, Linux, hardware partitioning, Jailhouse hypervisor, NXP Linux Yocto, Zephyr RTOS, FreeRTOS, MCUXpresso SDK
Abstract	This document presents the Harpoon release 3.5 for i.MX 8M and i.MX 9 device family, using the Arm Cortex-A53/A55 processor (Armv8-A architecture).

1 Overview

This document presents the Harpoon release 3.5 for i.MX 8M and i.MX 9 device families, using the Arm Cortex-A53/A55 processor (Armv8-A architecture).

Harpoon provides an environment for developing real-time demanding applications on an RTOS running on one (or several) Cortex-A core(s) in parallel of a Linux distribution, leveraging the 64-bit Arm architecture for higher performance.

The system starts on Linux and the Jailhouse hypervisor partitions the hardware to run both Linux and the guest RTOS in parallel.

The hardware partitioning is configurable and depends on the use case. Harpoon includes an audio application, an industrial application, and a real-time latency measurement application, all available both for FreeRTOS and Zephyr (some application feature limitations exist depending on the selected platform and RTOS).

Harpoon supports the following software and hardware:

- NXP Linux Yocto
  - i.MX LF6.12.34_2.1.0: For more information, see the i.MX Yocto Project User's Guide.
  - Real-time Edge SW v3.3: For more information, see the Real-time Edge Yocto Project v3.3 User Guide.
- i.MX 8M Series
  - i.MX 8M Mini LPDDR4 EVKB
  - i.MX 8M Nano LPDDR4 EVK
  - i.MX 8M Plus LPDDR4 EVK
- i.MX 9 Series
  - i.MX 93 EVK
  - i.MX 943 EVK
  - i.MX 95 15x15 LPDDR4x EVK
  - i.MX 95 19x19 LPDDR5 EVK
- Jailhouse hypervisor
- FreeRTOS V11.1.0 kernel
  - AARCH64 port, uniprocessor
  - Guest OS running on the Jailhouse cell
- Zephyr RTOS 4.3.0
  - Cortex-A53 and Cortex-A55 port, SMP
  - Guest OS running on the Jailhouse cell
- MCUXpresso SDK 25.09.00
  - GIC, Timer, and MMU AARCH64 drivers
  - FlexCAN, ENET, ENET_QOS, ENETC, GPT, TPM, I2C, LPI2C, SAI, LPUART, and UART SoC drivers
  - Audio Codec drivers
  - PHY drivers
- RTOS applications
  - Audio reference application
  - Industrial reference application
  - Real-time latency measurement application
  - Virtio Networking reference application
  - Hello World application

1.1 Supported features for i.MX 8M Family

Table 1. Harpoon 3.5 supported features

		i.MX 8M Mini		i.MX 8M Nano[1]		i.MX 8M Plus	
		FreeRTOS	Zephyr	FreeRTOS	Zephyr	FreeRTOS	Zephyr
Peripherals	GICv3	•	•	•	•	•	•
	MMU	•	•	•	•	•	•
	UART	•	•	•	•	•	•
	GPT	•	•	•	•	•	•
	I^2C	•	•	•	•	•	•
	SAI	•	•	•	•	•	•
	ENET	•	•	•	•	•	•
	ENET_QOS					•	•
	FlexCAN					•	•
	Audio Codec(s)	•	•	•	•	•	•
	Ethernet PHY(s)	•	•	•	•	•	•
Audio Expansion Boards	HiFiBerry	•	•	•	•	•	•
MiddleWare	GenAVB/TSN	•	•	•	•	•	•
	RPMsg-Lite	•	•	•	•	•	•
Audio Application	SAI pipeline(s)	•	•	•	•	•	•
	AVB pipeline	•	•	•	•	•	•
	AVB pipeline (with MCR)	•	•	•	•	•	•
	SMP pipeline		•		•		•
	AVB + SMP pipeline		•		•		•
	AVB + SMP pipeline (with MCR)		•		•		•
Industrial Application	CAN					•	•
	Ethernet	•	•	•	•	•	•
	TSN	•[2]	•[2]	•[2]	•[2]	•	•
Real-time Latency Application		•	•	•	•	•	•
Virtio Networking Application[3]		•				•	
Hello World Application		•	•	•	•	•	•

[1] i.MX Linux Yocto based image only

[2] Using ENET interface without 802.1Qbv support
[3] Real-time Edge based image only

1.2 Supported features for i.MX 9 Family

Table 2. Harpoon 3.5 supported features

		i.MX 93		i.MX 943		i.MX 95 15x15		i.MX 95 19x19	
		FreeRTOS	Zephyr	Free RTOS	Zephyr	Free RTOS	Zephyr	Free RTOS	Zephyr
Peripherals	GICv3	•	•	•	•	•	•	•	•
	MMU	•	•	•	•	•	•	•	•
	LPUART	•	•	•	•	•	•	•	•
	TPM	•	•	•	•	•	•	•	•
	LPI^{2}C	•	•						
	SAI	•	•						
	ENET	•	•						
	ENET_QOS	•	•						
	ENETC					•	•	•	•
	FlexCAN	•	•			•	•	•	•
	Audio Codec(s)	•	•						
	Ethernet PHY(s)	•	•			•	•	•	•
Audio Expansion Boards	MX93 AUD-HAT	•	•						
MiddleWare	GenAVB/ TSN	•	•			•	•	•	•
	RPMsg-Lite	•	•	•	•	•	•	•	•
Audio Application	SAI pipeline(s)	•	•						
	AVB pipeline	•	•						
	AVB pipeline (with MCR)	•	•						
Industrial Application	CAN	•	•			•	•	•	•
	Ethernet	•	•						
	TSN	•	•			•	•	•	•
Real-time Latency Application		•	•	•	•	•	•	•	•
Virtio Networking Application[1]		•							
Hello World Application		•	•	•	•	•	•	•	•

[1] Real-time Edge based image only

1.3 Architecture

The following figure shows the architecture of the Harpoon solution.

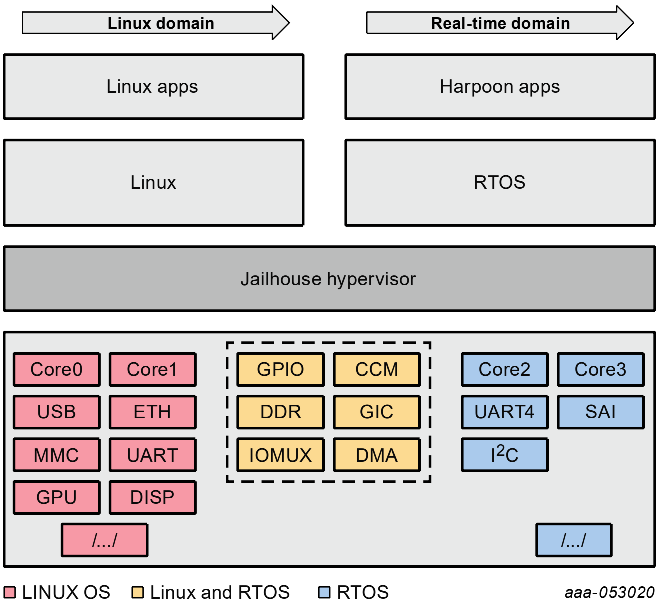


Figure 1. Harpoon solution architecture

The bottom box shows the hardware partitioning between Jailhouse cells.

The boxes in light red (group 1) show the main hardware blocks allocated to the Linux OS.

The boxes in blue (group 3) show the main hardware blocks allocated to the RTOS.

The boxes in light orange (group 2) show the main hardware blocks shared between Linux and the RTOS.

Harpoon-apps is a set of real-time application running on Jailhouse's inmate cell. It is built on top of Zephyr or FreeRTOS, using zephyr and/or MCUXpresso drivers.

1.4 Hardware resource partitioning

Jailhouse hypervisor is used to run an RTOS (FreeRTOS and Zephyr are supported) in parallel with Linux OS.

Jailhouse is a simple hypervisor that assigns hardware resources to a guest OS instead of virtualising them. For instance, a CPU core is statically assigned to a specific guest and is not shared with other guests.

In Jailhouse terms, the inmate (RTOS) runs in a cell. A configuration file describes which hardware resources are assigned to this cell. This configuration file contains descriptions of the following:

- CPU core(s) assigned to the cell
- Interrupt lines assigned to the cell
- Memory regions assigned to the cell
- Virtual PCI devices used for communication between cells

There is also a root cell configuration that describes the hardware assigned to the inmate prior to the hardware partitioning.

The source files of the cell configurations are embedded through patches in the Jailhouse recipe of the Harpoon meta-layer, at the following locations:

- `configs/arm64/imx{8m*,93,943,95}-harpoon-freertos.c` for the cell configuration of the FreeRTOS `hello_world` and `rt_latency` use cases
- `configs/arm64/imx{8m*,93,943,95}-harpoon-zephyr.c` for the cell configuration of the Zephyr `hello_world` and `rt_latency` use cases
- `configs/arm64/imx{8m*,93}-harpoon-freertos-audio.c` for the cell configuration of the FreeRTOS `audio` use case
- `configs/arm64/imx{8m*,93}-harpoon-zephyr-audio.c` for the cell configuration of the Zephyr `audio` use case
- `configs/arm64/imx{8m*,93}-harpoon-freertos-avb.c` for the cell configuration of the FreeRTOS `audio` (AVB) use case
- `configs/arm64/imx{8m*,93}-harpoon-zephyr-avb.c` for the cell configuration of the Zephyr `audio` (AVB) use case
- `configs/arm64/imx{8m*,93,95}-harpoon-freertos-industrial.c` for the cell configuration of the FreeRTOS `industrial` use case
- `configs/arm64/imx{8m*,93,95}-harpoon-zephyr-industrial.c` for the cell configuration of the Zephyr `industrial` use case
- `configs/arm64/imx{8m*,93}-harpoon-freertos-virtio.c` for the cell configuration of the FreeRTOS `Virtio Networking` use case
- `configs/arm64/imx8m*.c`, `configs/arm64/imx93.c`, `configs/arm64/imx943.c`, and `configs/arm64/imx95.c` for the root cell configuration

The CPU core(s) allocated to the RTOS forms a bitmap in the `cpu` structure:

- For i.MX 8M and i.MX 943, CPU core 3 is assigned to the cell:

```
.cpus = {
    0b1000,
    },
```

- For i.MX 93, CPU core 1 is assigned to the cell:

```
.cpus = {
    0b10,
    },
```

- For i.MX 95, CPU core 5 is assigned to the cell:

```
.cpus = {
    0b100000,
    },
```

- For a multicore (SMP) cell, two cores can be used. For instance, on i.MX 8M:

```
.cpus = {
    0b1100,
    },
```

Memory regions assigned to the inmate cell are listed in the `mem_regions` structure. Memory regions can be reserved for the inmate cell or shared with the Linux root cell.

Memory regions can be DDR chunks for the inmate cell use as well as device memory mapped regions, such as UART or SAI.

Interrupts are mapped to the cell with the `irqchips` structure.

Virtual PCI devices are defined with the `pci_devices` structure. These virtual devices are used by Jailhouse to implement IVSHMEM v2 communication channels.

1.5 i.MX 943 and i.MX 95 System Manager

The System Manager (SM) is a low-level system function, which adds support for isolation and management of power domains and hardware resources on multi-core processors.

Harpoon provides a custom System Manager configuration that describes the hardware used for its applications, such as the TPM and LPUART usage for its guest cell.

The System Manager configuration is directly embedded in the Real-Time Edge SW v3.3 Yocto recipes or in the Harpoon meta-layer for i.MX Yocto.

1.6 Acronym Table

The following table describes the acronyms used in this document.

Table 3. Acronyms

Term	Description
ADC	Analog-Digital Converter
AVB	Audio Video Bridging
AVTP	Audio Video Transport Protocol
CAN	Controller Area Network
DAC	Digital-Analog Converter
DIP	Dual In-line Package
DTMF	Dual-Tone Multi-Frequency (audio)
ENET	Ethernet
EVK	EValuation Kit
GIC	Generic Interrupt Controller
GPIO	General Purpose Input/Output
GPT	General Purpose Timer
I2C	Inter-integrated Circuit
IRQ	Interrupt ReQuest
MCR	Media Clock Recovery
MMU	Memory Management Unit
QOS	Quality of Service
RPMSG	Remote Processor Messaging
RTEdge	Real-time edge
RTOS	Real Time Operating System
SAI	Synchronous Audio Interface
SDK	Software Development Kit
SM	System Manager
SMP	Symmetric Multi-processing
TPM	Timer/PWM Module
TRRS	Tip, Ring, Ring, Sleeve (connector)

Table 3. Acronyms*...continued*

Term	Description
TRS	Tip, Ring, Sleeve (connector)
TSN	Time Sensitive Networking

2 Building Harpoon Yocto Images

2.1 i.MX Yocto

To build this release, fetch its Yocto manifest and get the meta-layers:

```
$ mkdir yocto
$ cd yocto
$ repo init -u https://github.com/nxp-imx/imx-manifest -b imx-linux-walnascar -m
 imx-6.12.34-2.1.0_harpoon-v3.5.xml
$ repo sync
```

Then, prepare the environment with the following command:

```
$ DISTRO=fsl-imx-xwayland MACHINE=<machine> source imx-harpoon-setup-release.sh
 -b build.<machine>
```

Where, `<machine>` is one of the following:

- `imx8mm-lpddr4-evk` for i.MX 8M Mini EVKB board
- `imx8mn-lpddr4-evk` for i.MX 8M Nano EVKB board
- `imx8mp-lpddr4-evk` for i.MX 8M Plus EVK board
- `imx93evk` for i.MX 93 EVK board
- `imx943-19x19-lpddr5-evk` for i.MX 943 EVK board
- `imx95-15x15-lpddr4x-evk` for i.MX 95 15x15 LPDDR4x EVK board
- `imx95-19x19-lpddr5-evk` for i.MX 95 19x19 LPDDR5 EVK board

The end user license agreement must be accepted to continue.

Then, build the image with the following command:

```
$ bitbake imx-image-core
```

The image is then available in the subdirectory `tmp/deploy/images/<machine>/`.

Copy the disk image to a micro-SD card. For example, assuming the card is recognized as `/dev/mmcblk0` by your host machine:

```
$ zstdcat imx-image-core-<machine>.wic.zst | sudo dd of=/dev/mmcblk0 bs=1M
```

The micro-SD card now contains the full image.

2.2 Real-Time Edge Yocto

See the Real-time Edge Yocto Project User Guide to build Harpoon and prepare an SD card for supported boards.

3 Hardware Setup

3.1 i.MX reference boards

Harpoon supports the following development boards.

3.1.1 i.MX 8M Mini EVK

Figure 2. i.MX 8M Mini EVK

Note: *For more information to order the board, see Evaluation Kit for the i.MX 8M Mini Applications Processor.*

3.1.2 i.MX 8M Mini EVKB

Figure 3. i.MX 8M Mini EVKB

Note: *For more information to order the board, see Evaluation Kit Rev. B for the i.MX 8M Mini Applications Processor.*

3.1.3 i.MX 8M Nano EVK

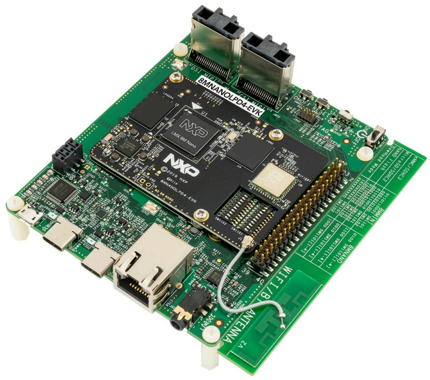


Figure 4. i.MX 8M Nano EVK

Note: *For more information to order the board, see [Evaluation Kit for the i.MX 8M Nano Applications Processor](#).*

3.1.4 i.MX 8M Plus EVK

Figure 5. i.MX 8M Plus EVK

Note: *For more information to order the board, see [Evaluation Kit for the i.MX 8M Plus Applications Processor](#).*

3.1.5 i.MX 93 EVK

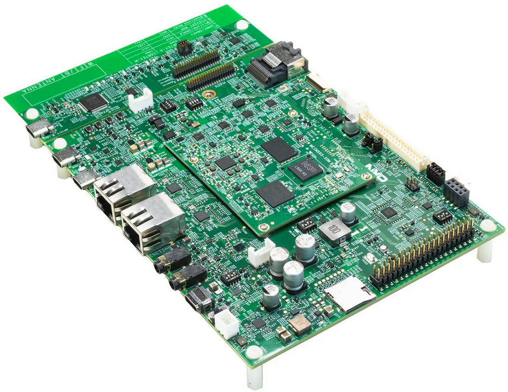


Figure 6. i.MX 93 EVK

Note: *For more information to order the board, see i.MX 93 Evaluation Kit.*

3.1.6 i.MX 943 EVK

Note: *For more information to order the board, see i.MX 94.*

Harpoon uses LPUART12 as the console on the i.MX 943 EVK, which is exposed via GPIO pins.

Required Hardware: A UART-to-USB Adapter (a device that converts serial UART signals to USB, enabling communication between embedded systems and computers) is needed to access the low-power UART console.

The following pins on the EVK's 16-pin connector and 20-pin connector must be connected to the following UART-to-USB Adapter's pins.

Table 4. EVK UART-to-USB adapter connection

EVK	UART-to-USB adapter
J44 pin 2: M1_UART12_RXD	**TX**
J44 pin 4: M1_UART12_TXD	**RX**
J43 pin 14: Ground	**GND**

3.1.7 i.MX 95 15x15 LPDDR4x EVK and i.MX 95 19x19 LPDDR5 EVK

Note: *For more information to order the board, see i.MX 95.*

3.2 Audio use case hardware

Harpoon audio application uses SAI as the digital audio interface to connect with different audio codecs for playback and capture. It uses the On-Board Audio Codecs and can use audio expansion boards as well.

The following table lists the different supported codecs and audio expansion boards supported by the application.

Table 5. Onboard Codec support

Board ID	Codec	Connector	SAI instance
i.MX 8M Nano EVK	DAC WM8524	3.5 mm 4-pole audio jack connector (J401)	3
i.MX 8M Mini EVK	DAC WM8524	3.5 mm 4-pole audio jack connector (J401)	3
i.MX 8M Plus EVK	Codec WM8960	3.5 mm 4-pole audio jack connector (J18)	3
i.MX 93 EVK	Codec WM8962	3.5 mm 4-pole audio jack connector (J1201)	3

Table 6. Audio expansion boards

Board ID	Audio expansion hardware	SAI instance
i.MX 8M Nano EVK	HiFiBerry - ADC PCM1863 and DAC PCM5122	5
i.MX 8M Mini EVK	HiFiBerry - ADC PCM1863 and DAC PCM5122	5
i.MX 8M Plus EVK	HiFiBerry - ADC PCM1863 and DAC PCM5122	5
i.MX 93 EVK	MX93AUD-HAT - Codec CS42448	3

3.2.1 HiFiBerry setup

For the Audio application use, the i.MX 8M family can be complemented by an I2S HiFiBerry audio card *DAC+ ADC Pro*.

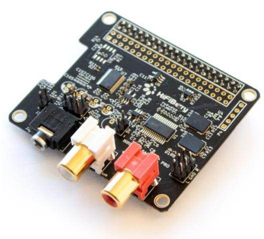


Figure 7. HiFiBerry DAC+ ADC Pro (picture from HiFiBerry's website)

Note: *HifiBerry related information link*

The HiFiBerry DAC+ ADC Pro is an audio card designed for the Raspberry Pi, but it can be connected to EVK boards using the 40-pin connector, provided a few adaptations are made.

The following pins on the EVK's 40-pin connector must be connected to the following HiFiBerry's pins.

Table 7. EVK - HiFiBerry transposition

EVK	HiFiBerry	Function
2	**2**	5V
3	**3**	I2C SDA
5	**5**	I2C SCK
6	**6**	GND
35	**40**	I2S TX
36	**12**	I2S clock

Table 7. EVK - HiFiBerry transposition...*continued*

EVK	HiFiBerry	Function
37	**35**	I2S word select for RX and TX
38	**38**	I2S RX

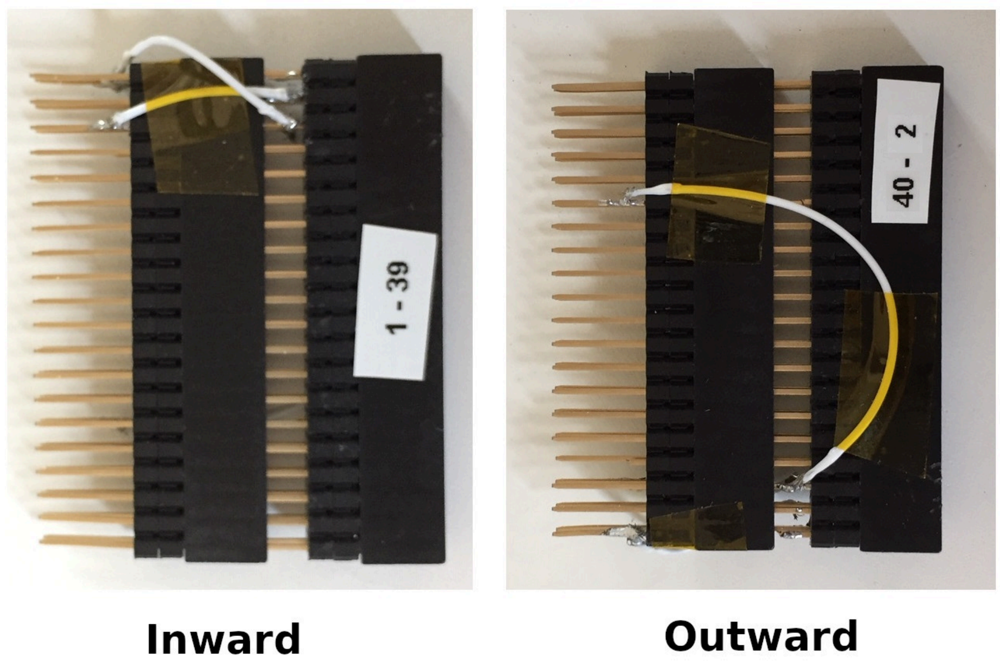


Figure 8. Handmade transposer

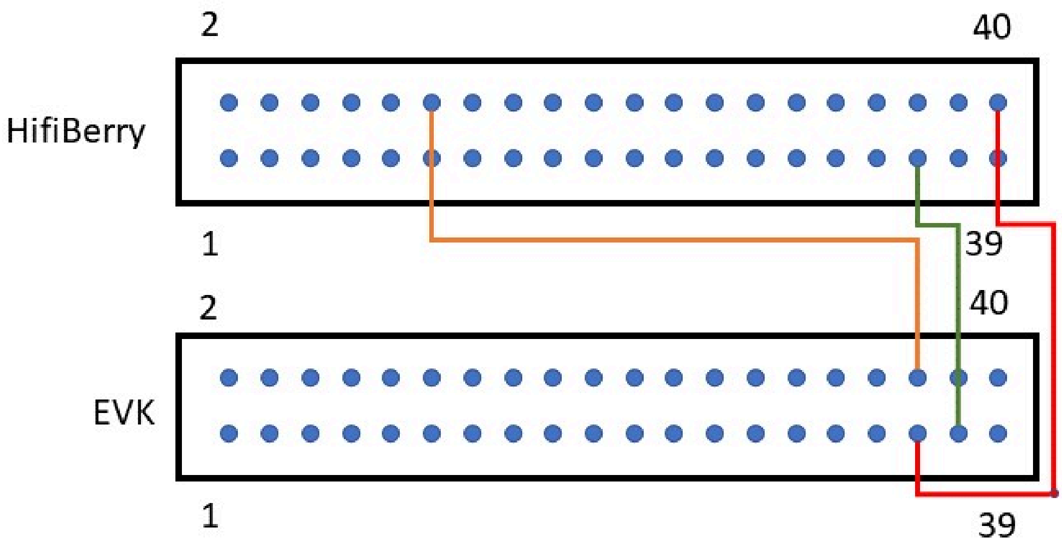


Figure 9. Transposer (Top View)

A complete setup, with a handmade transposer to respect above pinout, is shown as follows.

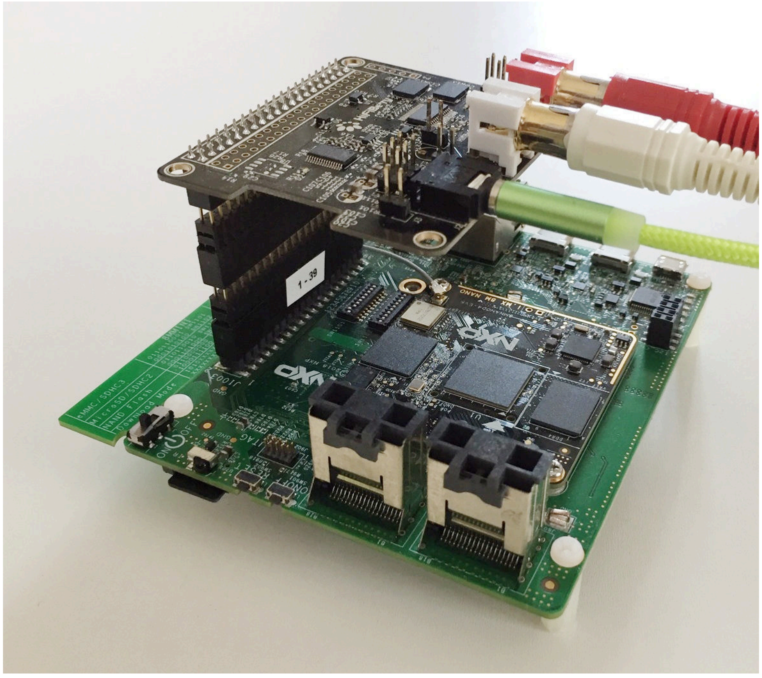


Figure 10. i.MX 8M Mini EVK with HiFiBerry audio card

The audio card has both an ADC (PCM1863) to record audio and a DAC (PCM5122) for audio playback.

Capture is done through the audio jack (connector highlighted in **1** in the following figure) and playback is done through the RCA connectors (highlighted in **2**).

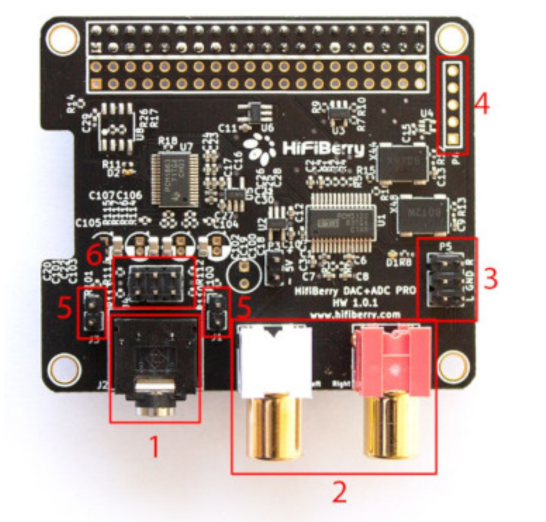


Figure 11. HiFiBerry audio connectors (picture from HiFiBerry's website)

Note: *HiFiBerry purchase link*.

Control of the PCM1863 is done through I2C3, at address 0x4a.

Control of the PCM5122 is done through I2C3, at address 0x4d.

Both the PCM1863 and PCM5122 use i.MX I2S5. The I2S5 is the I2S clock master. Two oscillators (one for sampling frequencies multiple of 44,100 Hz, one for sampling frequencies multiple of 48,000 Hz) are present on the HiFiBerry card, and controlled by PCM5122 GPIOs.

The following diagram shows the HiFiBerry architecture.

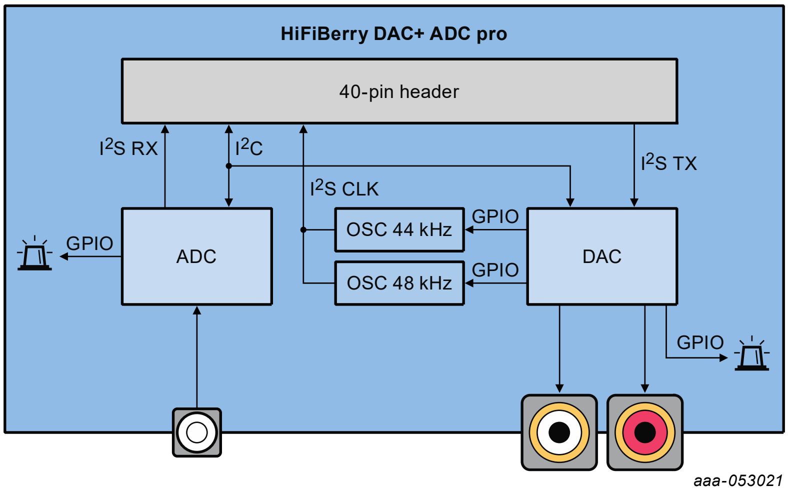


Figure 12. HiFiBerry architecture

The PCM1863 and the PCM5122 use the same signal for I2S word select by using SAI synchronous mode.

3.2.2 MX93AUD-HAT setup

For the Audio application use case, the i.MX 93 EVK can be complemented by an audio expansion board MX93AUD-HAT.

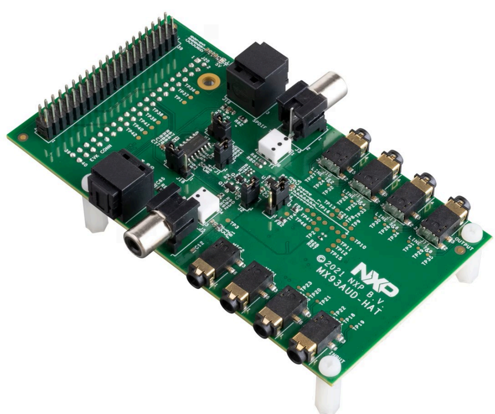


Figure 13. MX93AUD-HAT side image

Note: *MX93AUD-HAT related information link*

Jumpers configuration:

Use the following settings to configure the MX93AUD-HAT jumpers.

Table 8. Jumpers configuration

Jumper ID	Name	Settings
J1	5 V power control	Shorted: Connect 5 V power supply from the motherboard to the MX93AUD-HAT board (Default Setting)
J2	3V3 LDO power control	Shorted: Connect 3.3 V LDO U1 input path (Default Setting)
J4	3.3 V path selection	2-3 shorted: 3.3 V source from on board LDO (Default Setting)
J15	S/PDIF transmit	2-3 shorted: RCA port is used for S/PDIF transmit (Default Setting)
J18	S/PDIF receiver	2-3: RCA port is used as S/PDIF receiver (Default Setting)

The following table lists the MX93AUD-HAT sound input/output connectors used in our applications.

Table 9. Jack connectors

Connector ID	Connector description	Audio application use
J8 (STR1)	Line In, 2-channels per Jack (TRS)	Used as input line for Loopback, Full Pipeline and AVB use cases
J9 (STR2)	Line In, 2-channels per Jack (TRS)	Used as input line for Loopback, Full Pipeline and AVB use cases
J6 (MIC1)	MIC, 1-channel per Jack (TRRS)	Used as input mic for Loopback, Full Pipeline and AVB use cases
J7 (MIC2)	MIC, 1-channel per Jack (TRRS)	Used as input mic for Loopback, Full Pipeline and AVB use cases
J10 (LINE1&2)	Line Out, 2-channels per Jack (TRS)	Used as output line for all audio use cases
J12 (LINE3&4)	Line Out, 2-channels per Jack (TRS)	Used as output line for all audio use cases
J13 (LINE5&6)	Line Out, 2-channels per Jack (TRS)	Used as output line for all audio use cases
J11 (LINE7&8)	Line Out, 2-channels per Jack (TRS)	Used as output line for all audio use cases

3.3 Industrial use case hardware

Harpoon's industrial application may use the following hardware depending on the use case.

Figure 14. LS1028A AVB/TSN network bridge

Note: *For more information to order the board, see Layerscape LS1028A Reference Design Board.*

The LS1028A RDB is used as a TSN bridge/switch in a TSN network to demonstrate the TSN Ethernet use case running from the inmate cell.

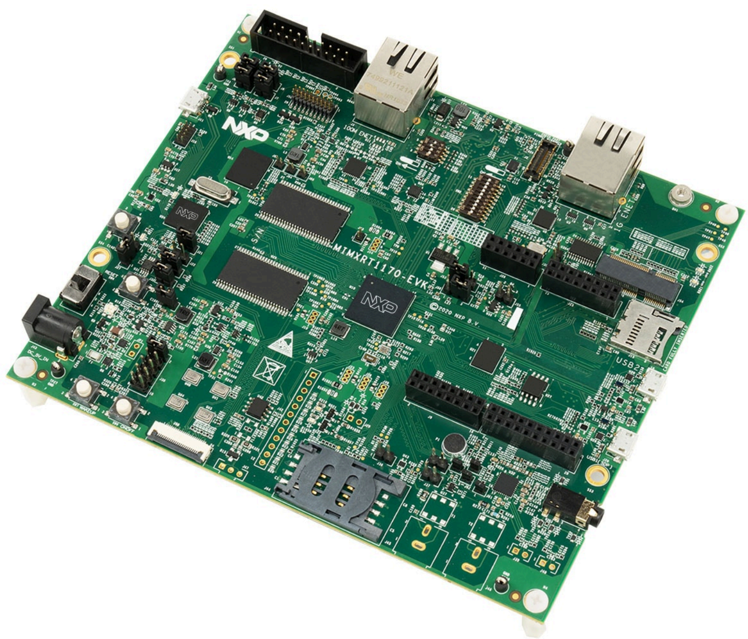


Figure 15. RT1170 TSN endpoint

Note: *For more information to order the board, see i.MX RT1170 Evaluation Kit.*

The RT1170 is used as a TSN endpoint in a TSN network, exchanging packets with a Harpoon TSN Endpoint (i.MX 8M Plus EVK or i.MX 93 EVK) for the motor control usecase.

3.4 Virtio networking use case hardware

User needs to connect **ENET port** on i.MX 8M Mini EVK / i.MX 8M Plus EVK / i.MX 93 EVK to another board/ PC or network switch/router to make sure the networking link is up before running Harpoon Virtio networking use case.

4 Running Harpoon Reference Applications

4.1 Basic setup

The EVK boards expose serial ports through their USB debug interface. One of these serial ports is used by Linux for its console, and another one is used by the guest RTOS.

To run the reference applications, open both serial ports with terminal emulators. Insert the micro-SD card, on which the Yocto image has been flashed, in the EVK and power up the board.

4.2 Starting Linux kernel

The Linux kernel must be started with a (Harpoon specific) Jailhouse compatible device tree.

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 command (based on the board and the application):

- For i.MX 8M Mini EVK:
  - hello_world, audio, or rt_latency:

```
u-boot => setenv jh_root_dtb imx8mm-evk-qca-wifi-harpoon.dtb
u-boot => run jh_mmcboot
```

  - hello_world or audio AVB:

```
u-boot => setenv jh_root_dtb imx8mm-evk-qca-wifi-harpoon-avb.dtb
u-boot => run jh_mmcboot
```

  - hello_world, industrial, or rt_latency:

```
u-boot => setenv jh_root_dtb imx8mm-evk-qca-wifi-harpoon-industrial.dtb
u-boot => run jh_mmcboot
```

  - hello_world or virtio networking:

```
u-boot => setenv jh_root_dtb imx8mm-evk-qca-wifi-harpoon-virtio-net.dtb
# Clear VirtIO magic value in memory in case of warm reboot to avoid MMIO
 probe error.
u-boot => mw b8400000 0 1
u-boot => run jh_mmcboot
```

- For i.MX 8M Mini EVKB:
  - hello_world, audio, or rt_latency:

```
u-boot => setenv jh_root_dtb imx8mm-evk-harpoon.dtb
u-boot => run jh_mmcboot
```

  - hello_world or audio AVB:

```
u-boot => setenv jh_root_dtb imx8mm-evk-harpoon-avb.dtb
u-boot => run jh_mmcboot
```

  - hello_world, industrial, or rt_latency:

```
u-boot => setenv jh_root_dtb imx8mm-evk-harpoon-industrial.dtb
u-boot => run jh_mmcboot
```

  - hello_world or virtio networking:

```
u-boot => setenv jh_root_dtb imx8mm-evk-harpoon-virtio-net.dtb
# Clear VirtIO magic value in memory in case of warm reboot to avoid MMIO
 probe error.
u-boot => mw b8400000 0 1
u-boot => run jh_mmcboot
```

- For i.MX 8M Nano EVK:
  - hello_world, audio, or rt_latency:

```
u-boot => setenv jh_root_dtb imx8mn-evk-harpoon.dtb
u-boot => run jh_mmcboot
```

  - hello_world or audio AVB:

```
u-boot => setenv jh_root_dtb imx8mn-evk-harpoon-avb.dtb
u-boot => run jh_mmcboot
```

  - hello_world, industrial or rt_latency:

```
u-boot => setenv jh_root_dtb imx8mn-evk-harpoon-industrial.dtb
```

```
u-boot => run jh_mmcboot
```

- For i.MX 8M Plus EVK:
  - hello_world, audio, or rt_latency:

```
u-boot => setenv jh_root_dtb imx8mp-evk-harpoon.dtb
u-boot => run jh_mmcboot
```

  - hello_world or audio AVB:

```
u-boot => setenv jh_root_dtb imx8mp-evk-harpoon-avb.dtb
u-boot => run jh_mmcboot
```

  - hello_world, industrial, or rt_latency:

```
u-boot => setenv jh_root_dtb imx8mp-evk-harpoon-industrial.dtb
u-boot => run jh_mmcboot
```

  - hello_world or virtio networking:

```
u-boot => setenv jh_root_dtb imx8mp-evk-harpoon-virtio-net.dtb
# Clear VirtIO magic value in memory in case of warm reboot to avoid MMIO
 probe error.
u-boot => mw fc700000 0 1
u-boot => run jh_mmcboot
```

- For i.MX 93 EVK:
  - hello_world, audio, or rt_latency:

```
u-boot => setenv jh_root_dtb imx93-11x11-evk-harpoon.dtb
u-boot => run jh_mmcboot
```

  - hello_world or audio AVB:

```
u-boot => setenv jh_root_dtb imx93-11x11-evk-harpoon-avb.dtb
u-boot => run jh_mmcboot
```

  - hello_world, industrial, or rt_latency:

```
u-boot => setenv jh_root_dtb imx93-11x11-evk-harpoon-industrial.dtb
u-boot => run jh_mmcboot
```

  - hello_world or virtio networking:

```
u-boot => setenv jh_root_dtb imx93-11x11-evk-harpoon-virtio-net.dtb
# Clear VirtIO magic value in memory in case of warm reboot to avoid MMIO
 probe error.
u-boot => mw fc700000 0 1
u-boot => run jh_mmcboot
```

- For i.MX 943 EVK:
  - hello_world or rt_latency:

```
u-boot => setenv jh_root_dtb imx943-evk-harpoon.dtb
u-boot => run jh_mmcboot
```

- For i.MX 95 15x15 EVK:
  - hello_world or rt_latency:

```
u-boot => setenv jh_root_dtb imx95-15x15-evk-harpoon.dtb
u-boot => run jh_mmcboot
```

  - industrial:

```
u-boot => setenv jh_root_dtb imx95-15x15-evk-harpoon-industrial.dtb
```

```
u-boot => run jh_mmcboot
```

- For i.MX 95 19x19 EVK:
  - hello_world or rt_latency:

```
u-boot => setenv jh_root_dtb imx95-19x19-evk-harpoon.dtb
u-boot => run jh_mmcboot
```

  - industrial:

```
u-boot => setenv jh_root_dtb imx95-19x19-evk-harpoon-industrial.dtb
u-boot => run jh_mmcboot
```

Note: *This configuration is not persistent after a reboot.*

To make changes permanent, execute the following commands once (after `setenv` above):

```
u-boot => setenv bootcmd 'run jh_mmcboot'
u-boot => saveenv
```

Now, at each reboot, the system starts with the Jailhouse compatible configuration and no user interaction is required.

4.3 hello_world application

The `hello_world` application is a simple demo for the basic features like IRQ, generic timer and UART/LPUART on FreeRTOS and Zephyr.

The application binary is available in the Harpoon share directory of the root file system:

```
/usr/share/harpoon/inmates/freertos/hello_world.bin # FreeRTOS binary
/usr/share/harpoon/inmates/zephyr/hello_world.bin # Zephyr binary
```

To use the `hello_world` application, Jailhouse must be started first. To start Jailhouse and the application, create the corresponding Harpoon configuration file and run the Harpoon service using `systemd`; for instance:

To run the FreeRTOS binary, create configuration:

```
# harpoon_set_configuration.sh freertos hello
```

To run the Zephyr binary, create configuration:

```
# harpoon_set_configuration.sh zephyr hello
```

Start the Harpoon service:

```
# systemctl start harpoon
```

The configuration file is stored under `/etc/harpoon/harpoon.conf`. The Harpoon systemd service uses it to start Jailhouse and the application.

Once the Harpoon service is started, the following logs are displayed on the inmate cell console:

FreeRTOS logs:

```
INFO: hello_func              : Hello world.
tic tac tic tac ...
```

Zephyr logs:

```
*** Booting Zephyr OS build zephyr-vxxx ***
INFO: hello_func            : Hello world.
INFO: hello_func            : 2 threads running
tic tac tic tac ...
```

4.4 Audio application

4.4.1 Features of the audio application

The audio application is available in the Harpoon share directory of the target's root file system:

```
/usr/share/harpoon/inmates/freertos/audio.bin   # FreeRTOS binary
/usr/share/harpoon/inmates/zephyr/audio.bin     # Zephyr binary
```

The different modes are:

- DTMF playback: Plays a DTMF sequence.
- Sine wave playback: Plays a generated sine wave.
- Loopback: Records sound from all available SAI sources and plays it live through the same SAI instances' sinks.
- Full Audio pipeline: Implements a flexible 3-stage pipeline with different sources (DTMF, sine waves, SAI input) that can be routed to different sinks (SAI outputs).
- AVB Audio pipeline: Implements a 3-stage pipeline with AVB input as a source that can be routed to different sinks (SAI outputs, AVTP sink).
- Milan AVB Audio pipeline with Media Clock Recovery support: Uses the pipeline above only with elements that support Media Clock Recovery.
- SMP Audio pipeline: Splits the Full Audio pipeline in two pieces to process them onto different cores.
- AVB SMP Audio pipeline: Splits the AVB Audio pipeline in two pieces to process them onto different cores.
- AVB SMP Audio pipeline with Media Clock Recovery support: Uses the pipeline above only with elements that support Media Clock Recovery.

All the modes support (see Notes for exceptions):

- Basic pipeline framework for audio processing.
- 44100, 48000, 88200, 96000, 176400, and 192000 Hz sample frequencies.
- Audio processing period with 2, 4, 8, 16, or 32 frames.
- Audio processing in 64-bit float format.
- Audio playback to SAI (one or more instances, on-board codec and/or audio expansion board).
- Audio capture from SAI (on-board codec and/or audio expansion board).

Table 10. Supported Audio features

	i.MX 8M Mini EVK		i.MX 8M Nano EVK		i.MX 8M Plus EVK		i.MX 93 EVK	
Audio Board/ Codec	On-Board (WM8524)	HiFiBerry	On-Board (WM8524)	HiFiBerry	On-Board (WM8960)	HiFiBerry	On-Board (WM8962)	MX93AUD-HAT
Supports Capture		•		•	•	•	•	•
SAI Instance	SAI3	SAI5	SAI3	SAI5	SAI3	SAI5	SAI3	SAI3

Table 10. Supported Audio features*...continued*

	i.MX 8M Mini EVK		i.MX 8M Nano EVK		i.MX 8M Plus EVK		i.MX 93 EVK	
Supported Frequencies (kHz)	44.1, 88.2, 176.4, 48, 96, 192	44.1, 88.2, 176.4, 48, 96, 192	44.1, 88.2, 176.4, 48, 96, 192	44.1, 88.2, 176.4, 48, 96, 192	44.1, 88.2	44.1, 88.2, 176.4, 48, 96, 192	48, 96, 192 (only playback)	48, 96, 192 (only playback)
Supported Periods (samples)	2, 4, 8, 16, 32	2, 4, 8, 16, 32	2, 4, 8, 16, 32	2, 4, 8, 16, 32	2, 4, 8, 16, 32	2, 4, 8, 16, 32	2, 4, 8, 16, 32	2, 4, 8

Note:

- *i.MX 93 supports only a single SAI instance (SAI3) and can be used to either connect to the on-board audio codec or the MX93AUD-HAT audio evaluation platform through a runtime application configuration.*
- *i.MX 93 does not support any of the SMP Audio pipelines.*
- *Media Clock Recovery: Only supported on i.MX 8M Plus EVK using SAI3 (on-board codec/sound jack).*

4.4.2 Starting the audio application

The Harpoon service uses the `/etc/harpoon/harpoon.conf` configuration file that contains the RTOS and the application to run. By default, the configuration file points to the FreeRTOS audio application.

To use the Zephyr audio application, run the following command to generate an appropriate configuration file:

```
# harpoon_set_configuration.sh zephyr audio
```

To use the audio application, start Jailhouse first. To start Jailhouse and the audio application, run the Harpoon service with Systemd:

```
# systemctl start harpoon
```

Once the Harpoon service is started, `harpoon_ctrl` is used to start or stop the audio modes with optional parameters. The different options for the audio application are:

```
Audio options:
  -f <frequency> audio clock frequency (in Hz):
                 imx8m{n,m,p}: supporting 44100, 48000, 88200, 96000,
                 176400 and 192000 Hz
                 imx93: supporting 48000 and 96000 Hz
                        supporting 48000, 96000, 192000 Hz using MX93AUD-HAT
                 Will use default frequency 48000Hz if not specified
  -p <frames>    audio processing period (in frames)
                 Supporting 2, 4, 8, 16, 32 frames
                 Will use default period 8 frames if not specified
  -a <mac_addr>  set hardware MAC address (default 00:bb:cc:dd:ee:14)
  -r <id>        run audio mode id:
                 0 - dtmf playback
                 1 - sine wave playback
                 2 - playback & recording (loopback)
                 3 - audio pipeline
                 4 - AVB audio pipeline
                 5 - SMP audio pipeline on imx8m{n,m,p}
                 6 - AVB audio pipeline (with MCR support) only on i.mx8mp
                 7 - SMP + AVB audio pipeline on imx8m{n,m,p}
                 8 - SMP + AVB audio pipeline (with MCR support) only on i.mx8mp
  -H             select the MX93AUD-HAT extension audio board. Only on i.mx93
  -s             stop running audio mode
```

```
Audio pipeline options:
  -a <pipeline_id>  audio pipeline id (default 0)
  -d                audio pipeline dump

Audio element options:
  -a <pipeline_id>  audio pipeline id (default 0)
  -d                audio element dump
  -e <element_id>   audio element id (default 0)
  -t <element_type> audio element type (default 0):
                    0 - dtmf source
                    1 - routing
                    2 - sai sink
                    3 - sai source
                    4 - sine source
                    5 - avtp source
                    6 - avtp sink

Routing audio element options:
  -a <pipeline_id>  audio pipeline id (default 0)
  -c                connect routing input/output
  -d                disconnect routing input/output
  -e <element_id>   routing element id (default 0)
  -i <input_id>     routing element input  (default 0)
  -o <output_id>    routing element output (default 0)
```

4.4.3 Audio latency in loopback mode

The loopback mode reads audio samples from HiFiBerry's ADC in an audio buffer and sends this buffer to the HiFiBerry's DAC when fully loaded.

The end-to-end latency, between the analog audio input and the analog audio output, has been measured and is dependent on the audio buffer size and the audio sampling rate. The RTOS and SoC combination does not alter the latency measurements.

Table 11. Audio application latency

Sampling rate (kHz)	Audio latency (µs)				
	Audio buffer size (frames)				
	32	16	8	4	2
192	612	442	363	317	295
176.4	669	488	397	351	329
96	1,202	873	703	623	578
88.2	1,315	952	771	680	635
48	2,392	1,723	1,383	1,224	1,134
44.1	2,596	1,870	1,508	1,327	1,236

4.4.4 Running audio application: examples

4.4.4.1 Playing DTMF

4.4.4.1.1 Using Stereo Codec

To run DTMF playback with the default parameters (48000 Hz sampling rate, 8 frame period size):

```
# harpoon_ctrl audio -r 0
```

The DTMF is played to all available stereo codecs connected to the different SAI outputs :

- On i.MX 8M EVKs: Both on-board Jack (SAI3) and HiFiBerry RCA output (SAI5)
- On i.MX 93 EVK: on-board Jack (SAI3)

To run another audio use case, stop the playback using the following command:

```
# harpoon_ctrl audio -s
```

Note: *To change the playback codec on i.MX 93 EVK, stop and restart the audio application.*

4.4.4.1.2 Using Multi-channel Codec (with MX93AUD-HAT)

On i.MX 93 EVK, to use the MX93AUD-HAT instead of the on-board codec for DTMF playback with the default parameters (48000 Hz sampling rate, 8 frame period size):

```
# harpoon_ctrl audio -r 0 -H
```

This will output audio on all 8-channel MX93AUD-HAT outputs (J10 to J13).

To run another audio use case, stop the playback using the following command:

```
# harpoon_ctrl audio -s
```

Note: *To change the playback codec on i.MX 93 EVK, stop and restart the audio application.*

4.4.4.2 Playing in loopback mode

4.4.4.2.1 Using Stereo Codec

In loopback mode, the SAI input is copied to the SAI output.

To start loopback mode with the default parameters (48000 Hz sampling rate, 8 frame period size):

```
# harpoon_ctrl audio -r 2
```

The loopbak mode is redirecting input for all available stereo ADC to all available stereo DACs (connected to the same SAI instance):

- i.MX 8M Mini and i.MX 8M Nano EVKs: Audio captured on HiFiBerry (SAI5) input is played back on the RCA outputs
- i.MX 8M Plus EVKs: Audio captured on HiFiBerry (SAI5) input is played back on the RCA outputs, and same for the on board jack (SAI3) when using a Headphone/MIC TRRS connector.
- i.MX 93 EVK: Audio captured on the MIC on-board jack (SAI3) is played back on it when using a Headphone/ MIC TRRS connector.

To run another audio use case, stop the playback using the following command:

```
# harpoon_ctrl audio -s
```

4.4.4.2.2 Using Multi-channel Codec (with MX93AUD-HAT)

In loopback mode, the SAI input is copied to the SAI output.

On i.MX 93 EVK, to use the MX93AUD-HAT instead of the on-board codec for the loopback mode with the default parameters (48000 Hz sampling rate, 8 frame period size):

```
harpoon_ctrl audio -r 2 -H
```

The MX93AUD-HAT codec has 6 input and 8 output channels, which are connected as follows for loopback.

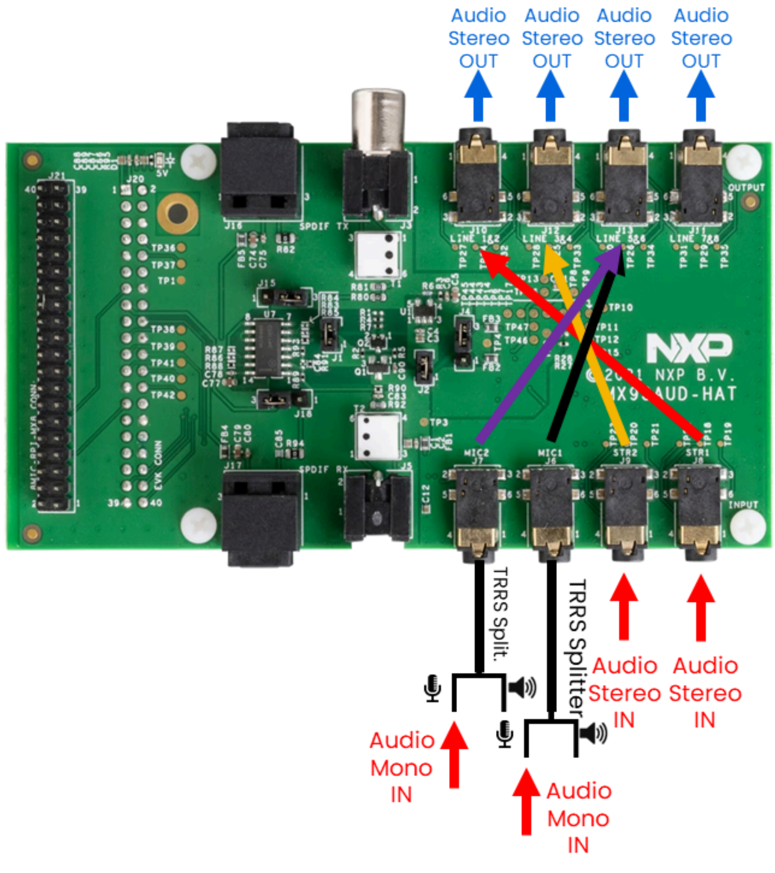


Figure 16. MX93AUD-HAT loopback mode

To run another audio use case, stop the playback using the following command:

```
# harpoon_ctrl audio -s
```

Note: *To change the playback codec on i.MX 93 EVK, stop and restart the audio application.*

4.4.4.3 Playing a full audio pipeline

The reference audio application is based on a basic pipeline framework for audio processing. Different audio processing elements can be assembled in a pipeline to process audio from source(s) to sink(s). The pipeline is processed in real time, cyclically with a fixed period.

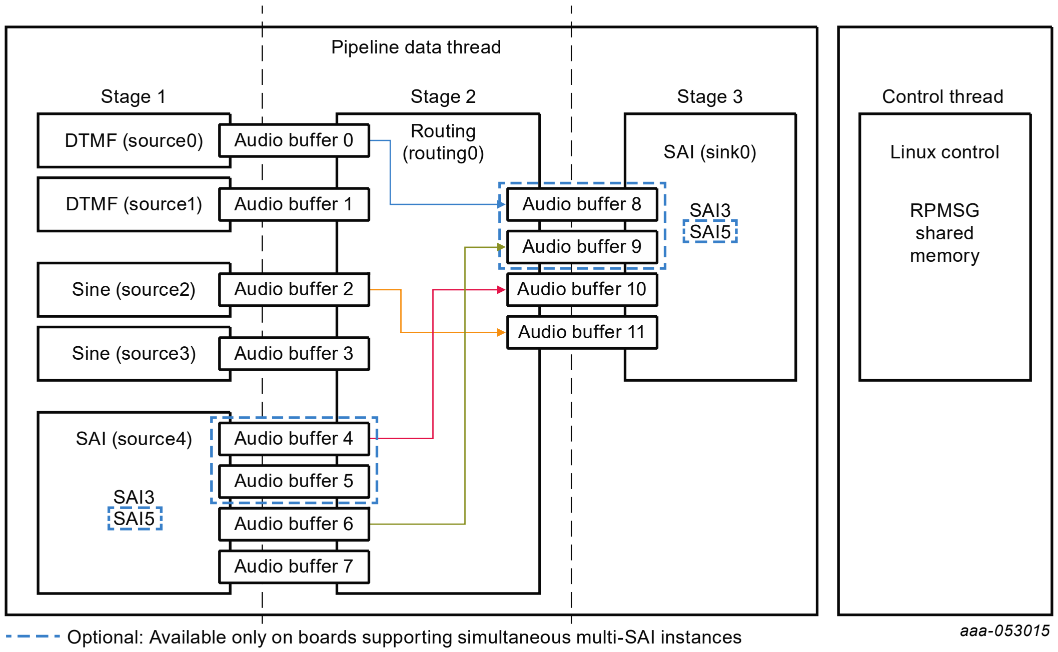
In the audio pipeline mode, there is a three stage pipeline composed of a routing element in stage 2, which can link source elements from stage 1 to sink elements from stage 3.

The full audio pipeline can be used with stereo codec or multi channel codec. The only difference is the number of available audio channel/buffers connected to each SAI instance for input or output.

4.4.4.3.1 Using Stereo Codec

The full audio pipeline on stereo Codecs provides two audio buffers per available SAI instance:

- i.MX 8M EVKs: Multi-SAI pipeline with SAI5 connected to HiFiBerry and SAI3 for the on-board jack.
- i.MX 93 EVK: Single-SAI pipeline with SAI3 connected to on-board jack.



Stereo audio pipeline with multiple sources/sinks and a routing element

Figure 17. Full audio pipeline using stereo codec

When running the audio pipeline, the routes can be configured dynamically with the `harpoon_ctrl` command. This command uses source and sink indexes to connect elements.

Table 12. Indexes of source elements

Index		Source element	Comment
Multi-SAI pipeline	Single-SAI pipeline		
0	0	DTMF, sequence 1	Software generated source
1	1	DTMF, sequence 2	Software generated source
2	2	Sine wave, 440 Hz	Software generated source
3	3	Sine wave, 880 Hz	Software generated source

Table 12. Indexes of source elements*...continued*

Index		Source element	Comment
Multi-SAI pipeline	Single-SAI pipeline		
4	N/A	SAI5, HifiBerry Jack left channel	Hardware source
5	N/A	SAI5, HifiBerry Jack right channel	Hardware source
6	4	SAI3, On-Board Jack left channel	Hardware source
7	5	SAI3, On-Board Jack right channel	Hardware source

Table 13. Indexes of sink elements

Index		Sink element	Comment
Multi-SAI pipeline	Single-SAI pipeline		
0	N/A	SAI5, HifiBerry RCA left channel	Hardware sink
1	N/A	SAI5, HifiBerry RCA right channel	Hardware sink
2	0	SAI3, On-Board Jack left channel	Hardware sink
3	1	SAI3, On-Board Jack right channel	Hardware sink

This makes for a flexible pipeline.

- For instance, on multi-SAI boards (i.MX 8M EVKs), the following commands starts the pipeline and configures the routing element to have a loopback between SAI input and SAI output (i.e., sound recorded by the HiFiBerry card played by the EVK's internal codec or audio jack input) while a DTMF sequence is played on the left channel of SAI's output and a 440 Hz sine wave on the right channel of SAI's output (i.e., HiFiBerry's output or audio jack output):

```
# harpoon_ctrl audio -r 3            # start audio pipeline
# harpoon_ctrl routing -i 4 -o 2 -c  # SAI5's input to SAI3's output(L)
# harpoon_ctrl routing -i 5 -o 3 -c  # SAI5's input to SAI3's output(R)
# harpoon_ctrl routing -i 0 -o 0 -c  # DTMF to SAI5's output(L)
# harpoon_ctrl routing -i 2 -o 1 -c  # sinewave 440Hz to SAI5's output(R)
```

- On the other hand, for boards with single-SAI support (i.MX 93 EVK), the following commands start the pipeline and routing element to have a DTMF sequence played on the left channel of SAI's output and a 440 Hz sine wave on the right channel of SAI's output (i.e., On-Board Codec audio jack output).

```
# harpoon_ctrl audio -r 3 # start audio pipeline using the On-Board Jack
# harpoon_ctrl routing -i 0 -o 0 -c # DTMF to SAI3's output(L)
# harpoon_ctrl routing -i 2 -o 1 -c # sinewave 440Hz to SAI3's output(R)
```

Note:

- *The pipeline dump also outputs the Audio Buffer Routing for an easier Buffer Routing through the "Routing Element".*
- *To change the audio codec used for i.MX 93 EVK, issue a pipeline stop before another run command.*

4.4.4.3.2 Using Multi-channel Codec (with MX93AUD-HAT)

The full audio pipeline on Multi-channel Codec (MX93AUD-HAT) provides 6 input and 8 output audio buffers:

- i.MX 93 EVK: Single SAI pipeline with SAI3 connected to MX93AUD-HAT codec.

Multi-channel audio pipeline with multiple sources/sinks and a routing element

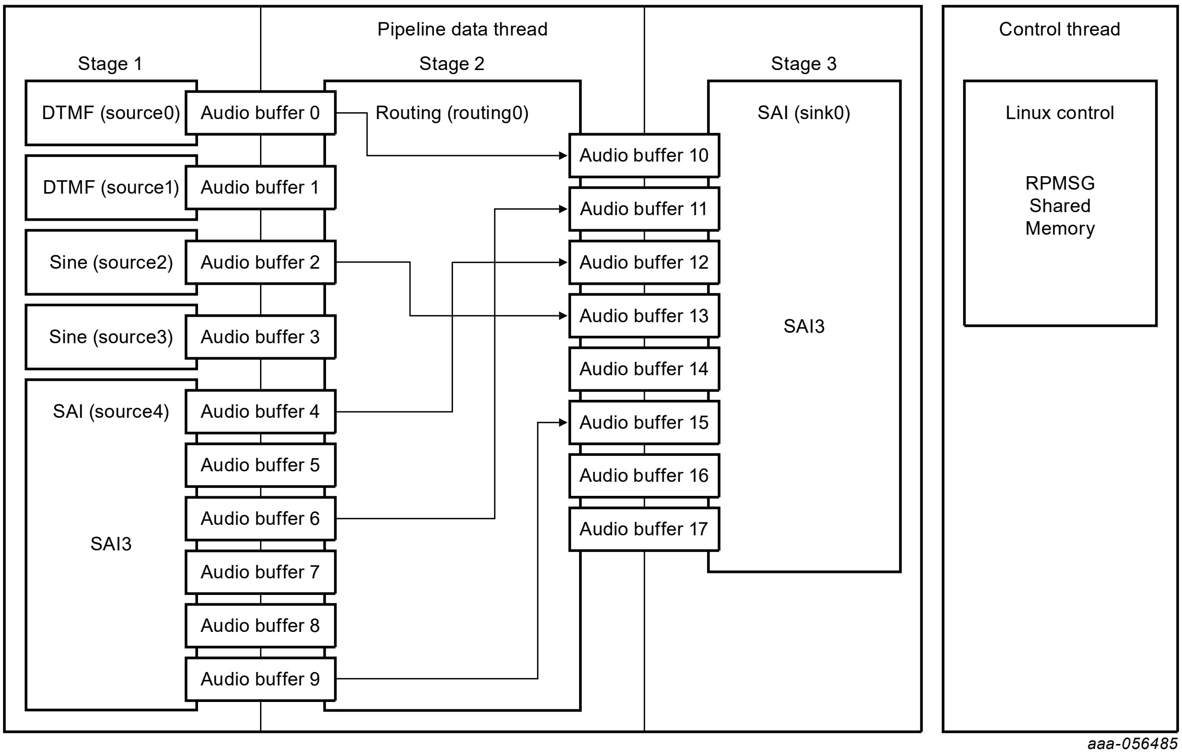


Figure 18. Full audio pipeline using multi-channel codec

When running the audio pipeline, the routes can be configured dynamically with the `harpoon_ctrl` command. This command uses source and sink indexes to connect elements.

Table 14. Indexes of source elements

Index	Source element	Comment
0	DTMF	Software generated source
1	DTMF	Software generated source
2	Sine wave, 440 Hz	Software generated source
3	Sine wave, 880 Hz	Software generated source
4	SAI3, left channel J8	Hardware source
5	SAI3, right channel J8	Hardware source
6	SAI3, left channel J9	Hardware source
7	SAI3, right channel J9	Hardware source
8	SAI3, mono channel J6	Hardware source
9	SAI3, mono channel J7	Hardware source

Table 15. Indexes of sink elements

Index	Sink element	Comment
0	SAI3, left channel J10	Hardware sink

Table 15. Indexes of sink elements...*continued*

Index	Sink element	Comment
1	SAI3, right channel J10	Hardware sink
2	SAI3, left channel J12	Hardware sink
3	SAI3, right channel J12	Hardware sink
4	SAI3, left channel J13	Hardware sink
5	SAI3, right channel J13	Hardware sink
6	SAI3, left channel J11	Hardware sink
7	SAI3, right channel J11	Hardware sink

This makes for a flexible pipeline.

- For instance, the following commands start the pipeline and configure the routing element to have a loopback between SAI input and SAI output (i.e., sound recorded by the MX93AUD-HAT and played back on the MX93AUD-HAT):

```
# harpoon_ctrl audio -r 3 -H          # start audio pipeline for the expansion
 board
# harpoon_ctrl routing -i 0 -o 0 -c   # DTMF to SAI3, left channel J10
# harpoon_ctrl routing -i 2 -o 1 -c   # Sine 440 Hz to SAI3, right channel J10
# harpoon_ctrl routing -i 1 -o 2 -c   # DTMF to SAI3, left channel J12
# harpoon_ctrl routing -i 3 -o 3 -c   # Sine 880 Hz to SAI3, left channel J12
# harpoon_ctrl routing -i 4 -o 4 -c   # SAI3, left channel J8 to SAI3, left
 channel J13
# harpoon_ctrl routing -i 5 -o 5 -c   # SAI3, right channel J8 to SAI3, right
 channel J13
# harpoon_ctrl routing -i 6 -o 6 -c   # SAI3, left channel J9 to SAI3, left
 channel J11
# harpoon_ctrl routing -i 7 -o 7 -c   # SAI3, right channel J9 to SAI3, left
 channel J11
```

Note:

- *The pipeline dump also outputs the Audio Buffer Routing for an easier Buffer Routing through the "Routing Element".*
- *To change the audio codec used for i.MX 93 EVK, issue a pipeline stop before another run command.*

4.4.4.4 Playing an AVB audio pipeline

4.4.4.4.1 AVB: Audio pipeline description

The AVB audio pipeline embeds an AVB Listener as a source element, using the GenAVB/TSN stack streaming APIs. This element is only responsible for the audio data path:

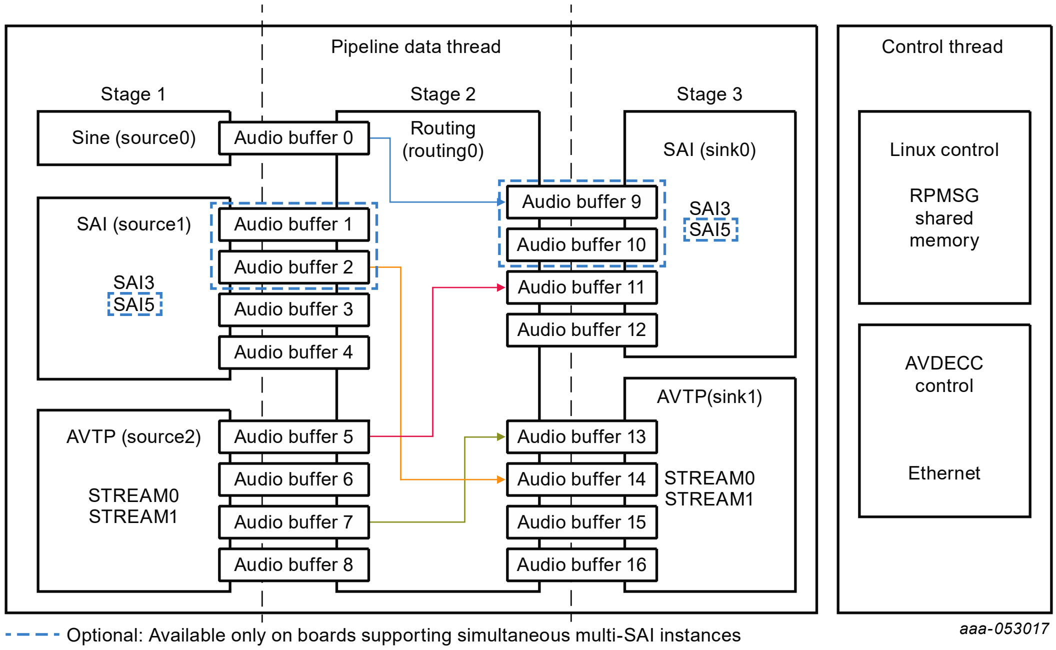
- Supports one or more AVTP Listener streams
- Supports one or more AVTP Talker streams
- Supports multi-channel AVTP streams
- Supports scatter of audio data
- Supports audio format conversion from AVTP stream format to the common format
- Supports Media Clock Recovery

It reuses the pipeline framework of the audio application for audio processing, in which an AVTP Listener is added as a source.

4.4.4.4.1.1 Using Stereo Codec

The AVB audio pipeline on Stereo Codec provides:

- An AVTP Listener with multiple streams as source
- An AVTP Listener with multiple streams as sink
- Two audio buffers per available SAI instance:
  - i.MX 8M EVKs: Multi-SAI pipeline with SAI5 connected to HiFiBerry and SAI3 for the on-board jack.
  - i.MX 93 EVK: Single-SAI pipeline with SAI3 connected to on-board jack.



AVB Audio pipeline showing the AVTP Listener with multiple streams as a source.

Figure 19. AVB Audio pipeline

When running the AVB audio pipeline, the routes can be configured dynamically with the `harpoon_ctrl` command. This command uses source and sink indexes to connect elements.

Table 16. Indexes of source elements

Index		Source element	Comment
Multi-SAI pipeline	Single-SAI pipeline		
0	0	Sine wave, 440 Hz	Software generated source
1	N/A	SAI5, HiFiBerry Jack left channel	Hardware source
2	N/A	SAI5, HiFiBerry Jack right channel	Hardware source
3	1	SAI3, On-Board Jack left channel	Hardware source
4	2	SAI3, On-Board Jack right channel	Hardware source
5	3	AVTP, stream#0 left channel	AVB source from network
6	4	AVTP, stream#0 right channel	AVB source from network
7	5	AVTP, stream#1 left channel	AVB source from network
8	6	AVTP, stream#1 right channel	AVB source from network

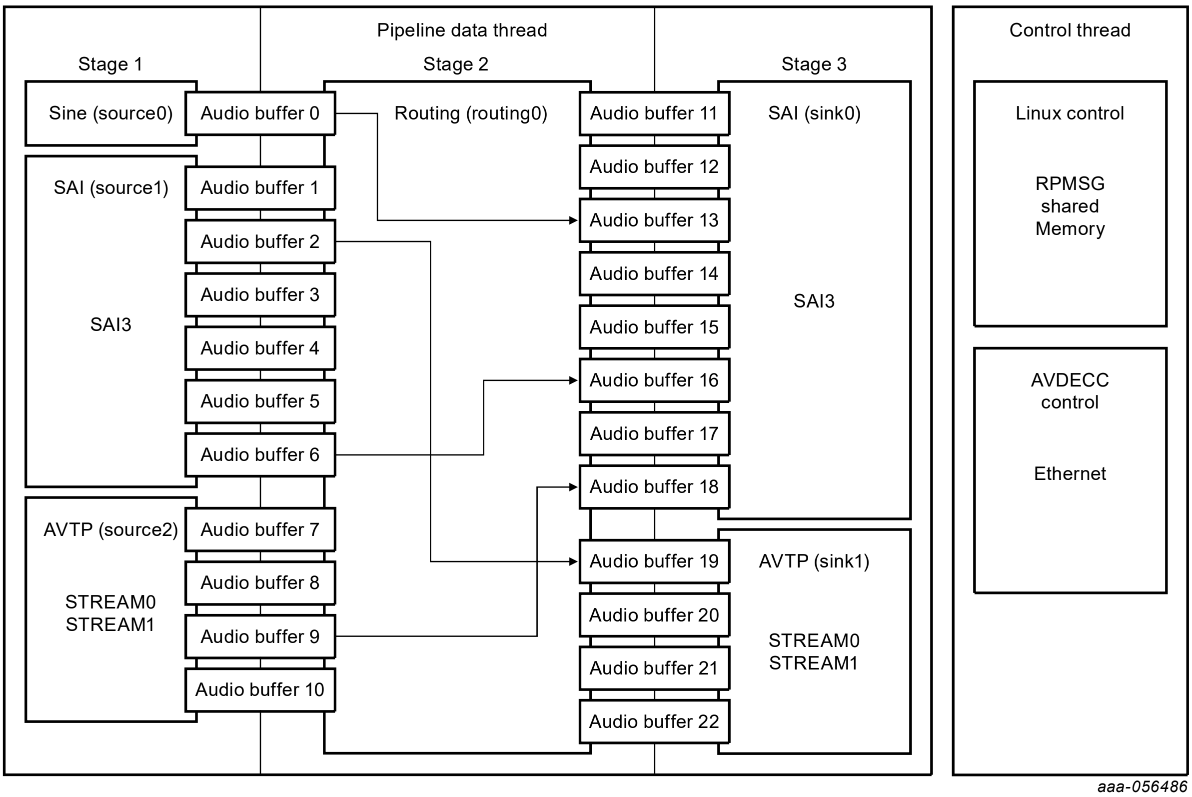
Table 17. Indexes of sink elements

Index		Sink element	Comment
Multi-SAI pipeline	Single-SAI pipeline		
0	N/A	SAI5, HiFiBerry RCA left channel	Hardware sink
1	N/A	SAI5, HiFiBerry RCA right channel	Hardware sink
2	0	SAI3, On-Board Jack left channel	Hardware sink
3	1	SAI3, On-Board Jack right channel	Hardware sink
4	2	AVTP, stream#0 left channel	AVB sink to network
5	3	AVTP, stream#0 right channel	AVB sink to network
6	4	AVTP, stream#1 left channel	AVB sink to network
7	5	AVTP, stream#1 right channel	AVB sink to network

4.4.4.4.1.2 Using Multi-Channel Codec (with MX93AUD-HAT)

The AVB audio pipeline Multi-channel Codec provides:

- An AVTP Listener with multiple streams as source
- An AVTP Listener with multiple streams as sink
- 6 input and 8 output audio buffers:
  - i.MX 93 EVK: Single SAI pipeline with SAI3 connected to MX93AUD-HAT codec



AVB Audio pipeline using multi-channel showing the AVTP Listener with multiple streams as a source.

Figure 20. AVB Audio pipeline using multi-channel

Table 18. Indexes of source elements

Index	Source element	Comment
0	Sine wave, 440 Hz	Software generated source
1	SAI3, left channel J8	Hardware source
2	SAI3, right channel J8	Hardware source
3	SAI3, left channel J9	Hardware source
4	SAI3, right channel J9	Hardware source
5	SAI3, mono channel J6	Hardware source
6	SAI3, mono channel J7	Hardware source
7	AVTP, stream#0 left channel	AVB source from network
8	AVTP, stream#0 right channel	AVB source from network
9	AVTP, stream#1 left channel	AVB source from network
10	AVTP, stream#1 right channel	AVB source from network

Table 19. Indexes of sink elements

Index	Sink element	Comment
0	SAI3, left channel J10	Hardware sink
1	SAI3, right channel J10	Hardware sink
2	SAI3, left channel J12	Hardware sink
3	SAI3, right channel J12	Hardware sink
4	SAI3, left channel J13	Hardware sink
5	SAI3, right channel J13	Hardware sink
6	SAI3, left channel J11	Hardware sink
7	SAI3, right channel J11	Hardware sink
8	AVTP, stream#0 left channel	AVB sink to network
9	AVTP, stream#0 right channel	AVB sink to network
10	AVTP, stream#1 left channel	AVB sink to network
11	AVTP, stream#1 right channel	AVB sink to network

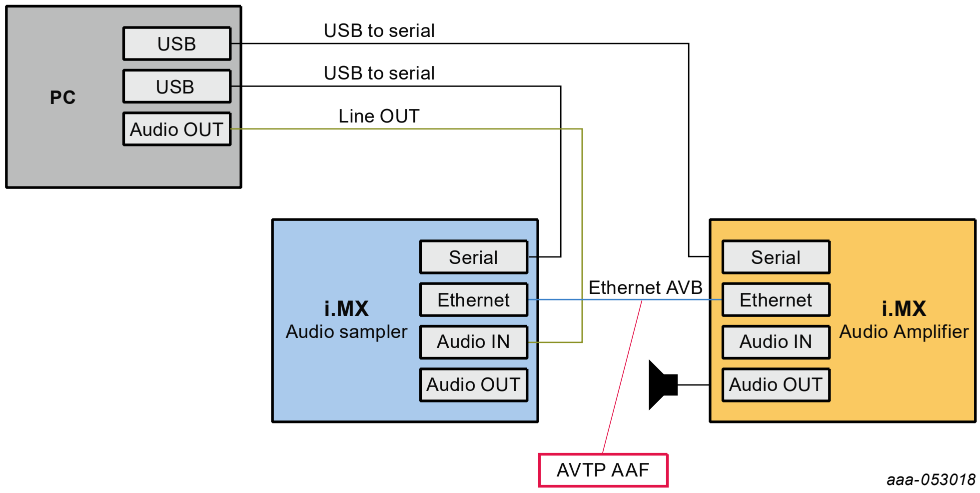
4.4.4.4.2 AVB: Harpoon AVTP Listener

This section describes how to set up an (external) AVB Audio Media Server (Talker) to enable the (Harpoon) AVB Listener.

4.4.4.4.2.1 AVB setup preparation

An i.MX 8M Plus EVK with Real-time Edge SW v3.3 (or above) can be used as a Talker. On the other end, any Harpoon supported EVK can be used as a Listener.

1. Connect the headphones/speakers to the HiFiBerry's RCA output or the Listener's audio Jack port.
2. Connect both the i.MX boards with an Ethernet RJ45 cable.
3. Connect a Serial/USB cable to each i.MX board and to some USB ports of the host PC.
4. Start consoles of the i.MX boards through the serial/USB ports.



AVB Talker (blue box) connected to the AVB Listener (orange box) through Ethernet.

Figure 21. AVB Audio setup

4.4.4.4.2.2 AVB Talker configuration (Linux)

The default AVB script needs to be modified to configure operations of the Talker entity as using a custom Media Application. The AVB Stack is provided with a simple Media Server application example, interfaced to the AVB stack through the GenAVB/TSN API, and supporting reading audio samples from a media file.

To enable AVB streaming using this media application, the endpoint needs to be configured as Endpoint AVB and the GenAVB/TSN configuration files needs to be modified as follows:

1. Power on the i.MX board and let the boot process complete
2. Configure the GenAVB/TSN stack to Endpoint AVB mode by setting GENAVB_TSN_CONFIG to the right value in the GenAVB/TSN mode configuration file:
   ```
   # vi /etc/genavb/config
   ```
   For i.MX 8M Plus EVK:
   ```
   GENAVB_TSN_CONFIG=2
   ```
3. Save and exit the file
4. Edit the GenAVB/TSN AVB configuration file using the following command:
   ```
   # vi /etc/genavb/config_avb
   ```
5. Set the configuration profile to PROFILE 2
   ```
   PROFILE=2
   ```
6. Save and exit the file.
7. A raw audio file `sample1_for_aaf.raw` is available in the `/home/media` repository. The multi-stream application example looks for audio files named `talker_mediaX.raw` in the `/home/media` repository, with X being the stream number. Therefore, before executing the multi-stream application, some symbolic links needs to be created in the `/home/media` directory for associating the talker_mediaX.raw names; here is an example for stream #0:
   ```
   # cd /home/media
   # ln -s sample1_for_aaf.raw talker_media0.raw
   ```

8. Enable the GenAVB/TSN systemd service to start the stack automatically on next reboot:

```
# systemctl enable genavb-tsn
```

9. Reboot the board. The change is saved across reboots, so this has only to be done once.
10. Stop in U-Boot and select the AVB device tree blob before booting Linux:

```
=> setenv fdtfile imx8mp-evk-avb.dtb
=> boot
```

4.4.4.4.2.3 AVB Listener configuration (Harpoon)

The AVB Listener is implemented in Harpoon interfaces with the AVB stack through the GenAVB/TSN API, and supports reading audio samples from the network while pushing out the audio data, through the audio pipeline, on the SAI interfaces.

To enable the AVB Listener on the Harpoon side, perform the following steps:

1. Power on the i.MX board and stop the boot process in U-Boot to fetch the AVB DTB file:

```
=> setenv jh_root_dtb imx8mp-evk-harpoon-avb.dtb
=> run jh_mmcboot
```

2. Start the audio application using the following command at the Linux prompt:
   - On FreeRTOS

   ```
   # harpoon_set_configuration.sh freertos avb
   # systemctl start harpoon
   ```

   - On Zephyr

   ```
   # harpoon_set_configuration.sh zephyr avb
   # systemctl start harpoon
   ```

3. Start the AVB pipeline, connecting the AVTP source element (stream #0) to the SAI output.
   - On Multi-SAI boards (i.MX 8M EVKs):
     Connect the AVTP element to the HiFiBerry board (SAI5).

   ```
   # harpoon_ctrl audio -r 4
   # harpoon_ctrl routing -i 5 -o 0 -c
   # harpoon_ctrl routing -i 6 -o 1 -c
   ```

   - On Single-SAI boards (i.MX 93 EVK using on-board codec):
     Connect the AVTP element to the on-board jack (SAI3).

   ```
   # harpoon_ctrl audio -r 4
   # harpoon_ctrl routing -i 3 -o 0 -c
   # harpoon_ctrl routing -i 4 -o 1 -c
   ```

   - On Single-SAI boards with Multi-channel Codec (i.MX 93 EVK with MX93AUD-HAT):
     Connect the AVTP element to the desired audio expansion board (SAI3) output jack (J10).

   ```
   # harpoon_ctrl audio -r 4 -H
   # harpoon_ctrl routing -i 7 -o 0 -c # AVTP, stream#0 left channel to SAI3,
    left channel J10
   # harpoon_ctrl routing -i 8 -o 1 -c # AVTP, stream#0 right channel to SAI3,
    right channel J10
   ```

4. Watch for AVTP source logs once the stream is connected (see next section):

```
INFO: avtp_source_element_st: rx stream: 0, avtp(C067ABF0, 0)
INFO: avtp_source_element_st:   connected: 1
INFO: avtp_source_element_st:   batch size: 64
```

```
INFO: avtp_source_element_st:   underflow: 459, overflow: 0 err: 0 received: 208617
INFO: avtp_source_element_st: rx stream: 1, avtp(0, 0)
INFO: avtp_source_element_st:   connected: 0
INFO: avtp_source_element_st:   batch size: 0
INFO: avtp_source_element_st:   underflow: 0, overflow: 0 err: 0 received: 0
```

4.4.4.4.2.4 AVB stream connection

This section describes how to use AVDECC events to configure the stream output of the Talker to the input of the Listener. To do so, we may use the GenAVB AVDECC controller application available on the Talker endpoint:

```
# genavb-controller-app -h
NXP's GenAVB AVDECC controller demo application

Usage:
app [options]

Options:
        -S, --set-control <control_type> <entity_id> <control_index> <value> Set a given control to
 the given value where control_type
                                                                       must be uint8 or utf8 (For
 utf8: <value> must be string of max 99 characters)
        -G, --get-control <control_type> <entity_id> <control_index>         Get a control value
 where control_type must be uint8 or utf8
        -l, --list-entities                                                  List discovered AVDECC
 entities
        -c, --connect-stream     <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id> <flags>   Connect a stream between a talker and a listener
        -d, --disconnect-stream  <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id>           Disconnect a stream between a talker and a listener
        -r, --get-rx-state       <listener_entity_id> <listener_unique_id>                Get
 information about a listener sink
        -t, --get-tx-state       <talker_entity_id> <talker_unique_id>                    Get
 information about a talker source
        -s, --get-tx-connection  <talker_entity_id> <talker_unique_id> <index>            Get
 information from a talker about a given connection/stream
        -T, --listener-streaming <talker_entity_id> <talker_unique_id> <start|stop>       Send
 START_STREAMING or STOP_STREAMING command to a talker
        -L, --talker-streaming   <listener_entity_id> <listener_unique_id> <start|stop>    Send
 START_STREAMING or STOP_STREAMING command to a listener
        -n, --set-name <entity_id> <configuration_id> <descritptor_type> <descriptor_index>
 <name_index> "<name>"       Send SET_NAME command to a descriptor
        --list-clock-domain-sources <entity_id> <clock_domain_index>                      List
 clock sources of a clock domain
        --set-clock-domain-source   <entity_id> <clock_domain_index> <clock_source_index>  Set the
 clock source of a clock domain
        -h                                                                                Print
 this help text                                                              print this help text
```

First of all, the Talker's entity information can be displayed by using the AVDECC controller application (available on the talker endpoint):

```
# genavb-controller-app -l
NXP's GenAVB AVDECC controller demo application
Number of discovered entities: 2
Entity ID = 0x49f070f840001     Model ID = 0x49f0000080001    Capabilities = 0x8 Association ID =
 0x0   MAC address= 00:04:9F:07:0F:84   Local MAC address= 00:04:9F:07:0F:84
     Controller
     Controls:
         None


Entity ID = 0x49f070f840000     Model ID = 0x49f0000090001    Capabilities = 0x708 Association ID =
 0x0   MAC address= 00:04:9F:07:0F:84   Local MAC address= 00:04:9F:07:0F:84
     Talker:     sources = 8     capabilities = 0x4801
        Stream   0: name =      Stream output 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =      Stream output 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
```

```
        Stream   2: name =      Stream output 2    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   3: name =      Stream output 3    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   4: name =      Stream output 4    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   5: name =      Stream output 5    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   6: name =      Stream output 6    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   7: name =      Stream output 7    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Listener:   sinks   = 8     capabilities = 0x4801
        Stream   0: name =       Stream input 0    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =       Stream input 1    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =       Stream input 2    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   3: name =       Stream input 3    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   4: name =       Stream input 4    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   5: name =       Stream input 5    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   6: name =       Stream input 6    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   7: name =       Stream input 7    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Controls:
         Control  0: name =     Volume Control 0    type = 0x90e0f00000000004    read-only =  No
  value_type = 1  min = 0 current = 100 max = 100 step = 1
```

Once the Listener is running, its entity ID can be displayed by using the same tool:

```
Entity ID = 0x49fddee100000     Model ID = 0x49fff00000001    Capabilities = 0x708 Association ID =
 0x0   MAC address= 00:BB:CC:DD:EE:10   Local MAC address= 00:04:9F:07:0F:84
     Talker:     sources = 3     capabilities = 0x4801
        Stream   0: name =      Stream output 0    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =      Stream output 1    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =      Stream output 2    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Listener:   sinks   = 3     capabilities = 0x4801
        Stream   0: name =       Stream input 0    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =       Stream input 1    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =       Stream input 2    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Controls:
         Control  0: name =     Volume Control 0    type = 0x90e0f00000000004    read-only =  No
  value_type = 1  min = 0 current = 100 max = 100 step = 1
```

To connect streams, use the following command:

```
# genavb-controller-app -c <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id> <flag>
```

To disconnect a stream, use the following command:

```
# genavb-controller-app -d <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id>
```

In the following example, the Listener's stream #0 is connected to the Talker’s stream #0:

```
# genavb-controller-app -c 0x49f070f840000 0 0x49fddee100000 0 0
NXP's GenAVB AVDECC controller demo application
```

```
Stream connection successful: stream id = 0x49f070f840000  Destination MAC address 91:E0:F0:00:FE:24
 flags = 0x0 connection_count = 1 VLAN id = 0
```

Once the stream is connected, the audio file can be heard on the SAI output lines.

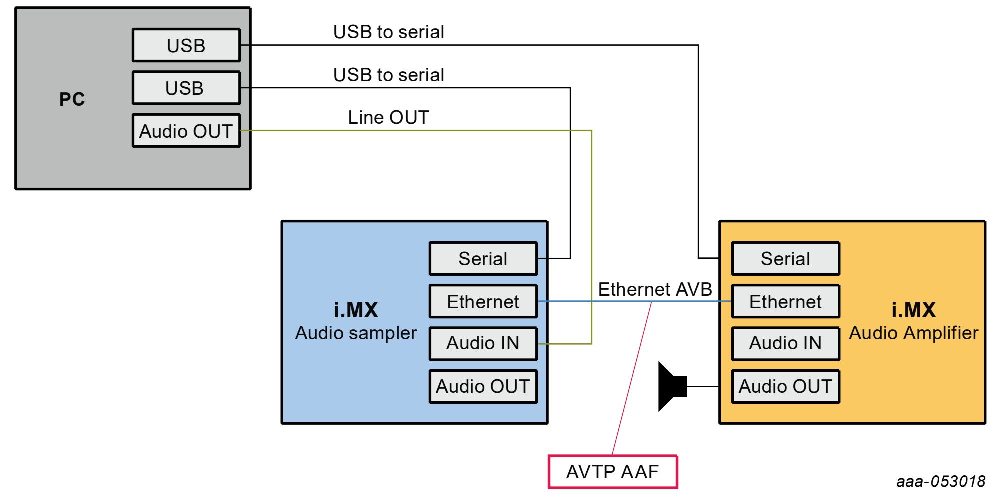
4.4.4.4.3 AVB: Harpoon AVTP Talker

This section describes how to set up an (external) AVB Audio Amplifier (Listener) to enable the (Harpoon) AVB Talker.

4.4.4.4.3.1 AVB setup preparation

An i.MX 8M Plus EVK with Real-time Edge SW v3.3 (or above) can be used as a Listener. On the other end, any Harpoon supported EVK can be used as a Talker.

1. Connect the headphones/speakers to the Listener's audio Jack port.
2. Connect both the i.MX boards with an Ethernet RJ45 cable.
3. Connect a Serial/USB cable to each i.MX board and to some USB ports of the host PC.
4. Start consoles of the i.MX boards through the serial/USB ports.



AVB Talker (blue box) connected to the AVB Listener (orange box) through Ethernet.

Figure 22. AVB Audio setup

4.4.4.4.3.2 AVB Listener configuration (Linux)

The default AVB configuration needs to be modified to enable the Listener entity in a custom Media Application. The AVB Stack is provided with a simple Media Server application example, interfaced to the AVB stack through the GenAVB/TSN API.

To enable AVB listening using this media application, the endpoint needs to be configured as Endpoint AVB and the GenAVB/TSN configuration files needs to be modified as follows:

1. Power on the i.MX board and let the boot process complete
2. Configure the GenAVB/TSN stack to Endpoint AVB mode by setting GENAVB_TSN_CONFIG to the right value in the GenAVB/TSN mode configuration file:

```
# vi /etc/genavb/config
```

For i.MX 8M Plus EVK:

```
GENAVB_TSN_CONFIG=2
```

3. Save and exit the file
4. Edit the GenAVB/TSN AVB configuration file using the following command:

```
# vi /etc/genavb/config_avb
```

5. Set the configuration profile to PROFILE 14

```
PROFILE=14
```

6. Save and exit the file.
7. Enable the GenAVB/TSN systemd service to start the stack automatically on next reboot:

```
# systemctl enable genavb-tsn
```

8. Reboot the board. The change is saved across reboots, so this has only to be done once.
9. Stop in U-Boot and select the AVB device tree blob before booting Linux:
   For i.MX 8M Plus EVK:

```
=> setenv fdtfile imx8mp-evk-avb.dtb
=> boot
```

4.4.4.4.3.3 AVB Talker configuration (Harpoon)

The AVB Talker implemented in Harpoon interfaces with the AVB stack through the GenAVB/TSN API, and supports audio streaming to the network while reading the audio data, through the audio pipeline, from the SAI interfaces.

To enable the AVB Talker on the Harpoon side, perform the following steps:

1. Power on the i.MX board and stop the boot process in U-Boot to fetch the AVB DTB file:

```
=> setenv jh_root_dtb imx8mp-evk-harpoon-avb.dtb
=> run jh_mmcboot
```

2. Start the audio application using the following command at the Linux prompt:
   - On FreeRTOS

```
# harpoon_set_configuration.sh freertos avb
# systemctl start harpoon
```

   - On Zephyr

```
# harpoon_set_configuration.sh zephyr avb
# systemctl start harpoon
```

3. Start the AVB pipeline, connecting the SAI input to the AVTP sink element (stream #0).
   - On Multi-SAI boards (i.MX 8M EVKs):
     Connect the HiFiBerry board (SAI5) to the AVTP element.

```
# harpoon_ctrl audio -r 4 -a 00:bb:cc:dd:be:ef
# harpoon_ctrl routing -i 1 -o 4 -c
# harpoon_ctrl routing -i 2 -o 5 -c
```

   - On Single-SAI boards (i.MX 93 EVK using on-board codec):
     Connect the on-board audio jack (SAI3) to the AVTP element.

```
# harpoon_ctrl audio -r 4 -a 00:bb:cc:dd:be:ef
# harpoon_ctrl routing -i 1 -o 2 -c
```

```
# harpoon_ctrl routing -i 2 -o 3 -c
```

- On Single-SAI boards with Multi-channel Codec (i.MX 93 EVK with MX93AUD-HAT):
  Connect the desired audio expansion board (SAI3) input jack (J8) to the AVTP element:

```
# harpoon_ctrl audio -r 4 -H -a 00:bb:cc:dd:be:ef
# harpoon_ctrl routing -i 1 -o 8 -c # SAI3, left channel J8 to AVTP,
 stream#0 left channel
# harpoon_ctrl routing -i 2 -o 9 -c # SAI3, right channel J8 to AVTP,
 stream#0 right channel
```

4. Watch for the AVTP sink logs once the stream is connected (see next section):

```
INFO: avtp_sink_element_st: rx stream: 0, avtp(C067ABF0, 0)
INFO: avtp_sink_element_st:   connected: 1
INFO: avtp_sink_element_st:   batch size: 64
INFO: avtp_sink_element_st:   underflow: 459, overflow: 0 err: 0 sent: 208617
INFO: avtp_sink_element_st: rx stream: 1, avtp(0, 0)
INFO: avtp_sink_element_st:   connected: 0
INFO: avtp_sink_element_st:   batch size: 0
INFO: avtp_sink_element_st:   underflow: 0, overflow: 0 err: 0 sent: 0
```

4.4.4.4.3.4 AVB stream connection

This section describes how to use AVDECC events to configure the stream output of the Talker to the input of the Listener. To do so, use the GenAVB AVDECC controller application available on the Listener (Linux endpoint):

```
# genavb-controller-app -h
NXP's GenAVB AVDECC controller demo application

Usage:
app [options]

Options:
        -S, --set-control <control_type> <entity_id> <control_index> <value> Set a given control to
 the given value where control_type
                                                                        must be uint8 or utf8 (For
 utf8: <value> must be string of max 99 characters)
        -G, --get-control <control_type> <entity_id> <control_index>         Get a control value
 where control_type must be uint8 or utf8
        -l, --list-entities                                                  List discovered AVDECC
 entities
        -c, --connect-stream     <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id> <flags>   Connect a stream between a talker and a listener
        -d, --disconnect-stream  <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id>           Disconnect a stream between a talker and a listener
        -r, --get-rx-state       <listener_entity_id> <listener_unique_id>                 Get
 information about a listener sink
        -t, --get-tx-state       <talker_entity_id> <talker_unique_id>                     Get
 information about a talker source
        -s, --get-tx-connection  <talker_entity_id> <talker_unique_id> <index>             Get
 information from a talker about a given connection/stream
        -T, --listener-streaming <talker_entity_id> <talker_unique_id> <start|stop>        Send
 START_STREAMING or STOP_STREAMING command to a talker
        -L, --talker-streaming   <listener_entity_id> <listener_unique_id> <start|stop>    Send
 START_STREAMING or STOP_STREAMING command to a listener
        -n, --set-name <entity_id> <configuration_id> <descritptor_type> <descriptor_index>
 <name_index> "<name>"       Send SET_NAME command to a descriptor
        --list-clock-domain-sources <entity_id> <clock_domain_index>                       List
 clock sources of a clock domain
        --set-clock-domain-source   <entity_id> <clock_domain_index> <clock_source_index>  Set the
 clock source of a clock domain
        -h                                                                                 Print
 this help text
```


First of all, the Talker's entity information can be displayed by using the AVDECC controller application (available on the talker endpoint):

```
# genavb-controller-app -l
NXP's GenAVB AVDECC controller demo application
Number of discovered entities: 4
Entity ID = 0x49f05cf720001     Model ID = 0x49f0000080001    Capabilities = 0x8 Association ID =
 0x0   MAC address= 00:04:9F:05:CF:72   Local MAC address= 00:04:9F:05:CF:72
     Controller
     Controls:
         None


Entity ID = 0x49f070f840000     Model ID = 0x49f0000090001    Capabilities = 0x708 Association ID =
 0x0   MAC address= 00:04:9F:07:0F:84   Local MAC address= 00:04:9F:05:CF:72
     Talker:     sources = 8     capabilities = 0x4801
        Stream   0: name =      Stream output 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =      Stream output 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =      Stream output 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   3: name =      Stream output 3    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   4: name =      Stream output 4    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   5: name =      Stream output 5    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   6: name =      Stream output 6    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   7: name =      Stream output 7    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Listener:   sinks   = 8     capabilities = 0x4801
        Stream   0: name =       Stream input 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =       Stream input 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =       Stream input 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   3: name =       Stream input 3    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   4: name =       Stream input 4    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   5: name =       Stream input 5    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   6: name =       Stream input 6    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   7: name =       Stream input 7    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Controls:
         Control  0: name =     Volume Control 0    type = 0x90e0f00000000004    read-only =  No
 value_type = 1  min = 0 current = 100 max = 100 step = 1


Entity ID = 0x49fddbeef0000     Model ID = 0x49fff00000001    Capabilities = 0x708 Association ID =
 0x0   MAC address= 00:BB:CC:DD:BE:EF   Local MAC address= 00:04:9F:05:CF:72
     Talker:     sources = 3     capabilities = 0x4801
        Stream   0: name =      Stream output 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =      Stream output 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =      Stream output 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Listener:   sinks   = 3     capabilities = 0x4801
        Stream   0: name =       Stream input 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =       Stream input 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =       Stream input 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Controls:
         Control  0: name =     Volume Control 0    type = 0x90e0f00000000004    read-only =  No
 value_type = 1  min = 0 current = 100 max = 100 step = 1
```

Once the Talker is running, its entity ID can be displayed by using the same tool:

```
Entity ID = 0x49fddbeef0000     Model ID = 0x49fff00000001    Capabilities = 0x708 Association ID =
 0x0   MAC address= 00:BB:CC:DD:BE:EF   Local MAC address= 00:04:9F:05:CF:72
     Talker:     sources = 3     capabilities = 0x4801
        Stream   0: name =      Stream output 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =      Stream output 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =      Stream output 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Listener:   sinks   = 3     capabilities = 0x4801
        Stream   0: name =       Stream input 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =       Stream input 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =       Stream input 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Controls:
         Control  0: name =     Volume Control 0    type = 0x90e0f00000000004    read-only =  No
 value_type = 1  min = 0 current = 100 max = 100 step = 1
```

To connect streams, use the following command:

```
# genavb-controller-app -c <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id> <flag>
```

To disconnect a stream, use the following command:

```
# genavb-controller-app -d <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id>
```

In the following example, the Listener's stream #0 is connected to the Talker's stream #0:

```
# genavb-controller-app -c 0x49fddbeef0000 0 0x49f070f840000 0 0
NXP's GenAVB AVDECC controller demo application
Stream connection successful: stream id = 0xbbccddbeef0000  Destination MAC address
 91:E0:F0:00:FE:21 flags = 0x0 connection_count = 1 VLAN id = 0
```

Once the stream is connected, the audio file can be heard on the SAI output lines.

4.4.4.4.4 AVB Connect Harpoon Listeners and Talker through an AVB bridge

This section describes how to set up two (or more) Harpoon AVB endpoints as Talker and Listener.

4.4.4.4.4.1 AVB setup preparation

The AVB Listeners and Talker implemented in Harpoon can be connected with each other, and support reading audio samples from the network while pushing out the audio data, through the audio pipeline, on the SAI interfaces.

- Two or more AVB endpoints (i.MX 8M Plus EVK, i.MX 8M Mini EVK, i.MX 8M Nano EVK, or i.MX 93 EVK).
- AVDECC controller (e.g., i.MX 8M Plus EVK with Real-time Edge SW v3.3 or above as AVB endpoint using `genavb-controller-app`) or a third-party AVDECC controller running on the PC connected to the switch (e.g., Hive AVB controller).
- One AVB bridge (e.g., LS1028ARDB with Real-time Edge SW v3.3 or above).

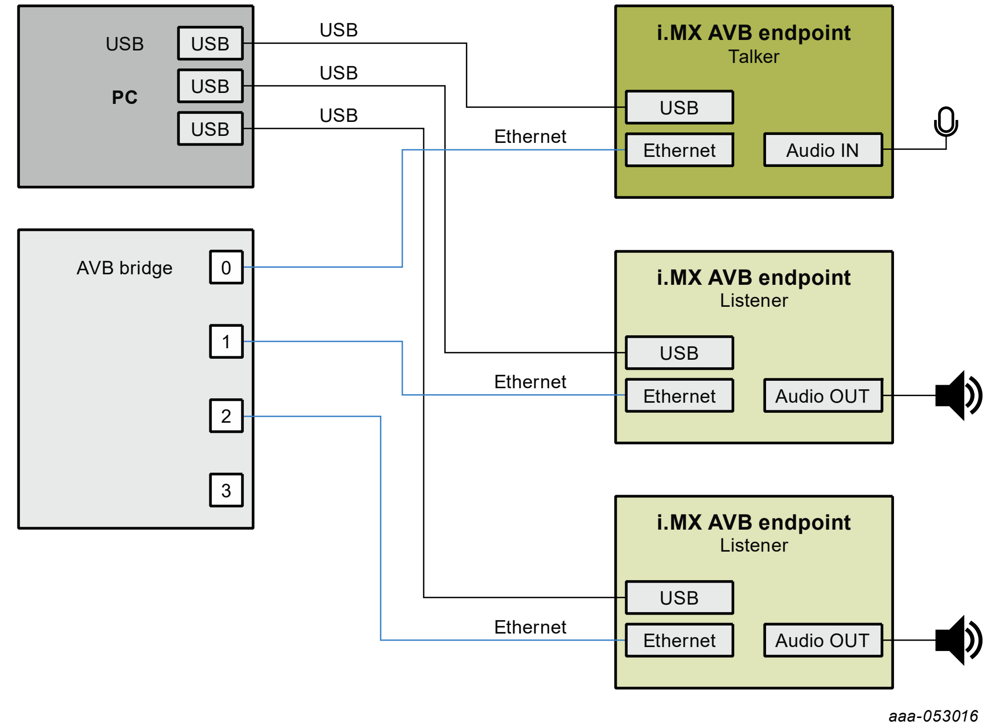


Figure 23. Harpoon AVTP Talker/Listener setup

4.4.4.4.4.2 AVB Bridge Configuration

Use the following commands to configure bridge on LS1028ARDB:

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

4.4.4.4.4.3 AVB Listeners configuration (Harpoon)

The AVB Listener is implemented in Harpoon interfaces with the AVB stack through the GenAVB/TSN API, and supports reading audio samples from the network while pushing out the audio data, through the audio pipeline, on the SAI interfaces.

To enable the AVB Listener on the Harpoon side, perform the following steps:

1. Power on the i.MX board and stop the boot process in U-Boot to fetch the AVB DTB file:

```
=> setenv jh_root_dtb imx8mp-evk-harpoon-avb.dtb
=> run jh_mmcboot
```

2. Start the audio application using the following command at the Linux prompt:
   - On FreeRTOS

```
# harpoon_set_configuration.sh freertos avb
# systemctl start harpoon
```

   - On Zephyr

```
# harpoon_set_configuration.sh zephyr avb
```

```
# systemctl start harpoon
```

3. Start the AVB pipeline, connecting the AVTP source element (stream #0) to the SAI output.
   - On Multi-SAI boards (i.MX 8M EVKs):
     Connect the AVTP element to the HiFiBerry board (SAI5).

```
# harpoon_ctrl audio -r 4 -a 00:bb:cc:dd:be:ef
# harpoon_ctrl routing -i 5 -o 0 -c
# harpoon_ctrl routing -i 6 -o 1 -c
```

   - On Single-SAI boards (i.MX 93 EVK):
     Connect the AVTP element to the on-board jack (SAI3) or to the audio expansion board output jack (using an additional `-H` option in the command).

```
# harpoon_ctrl audio -r 4 -a 00:bb:cc:dd:be:ef # add -H to select the
 MX93AUD-HAT
# harpoon_ctrl routing -i 3 -o 0 -c
# harpoon_ctrl routing -i 4 -o 1 -c
```

4. For other AVB AVTP Listener instances, use a different MAC address:
   - On Multi-SAI boards (i.MX 8M EVKs):

```
# harpoon_ctrl audio -r 4 -a 00:bb:cc:dd:ca:fe
# harpoon_ctrl routing -i 5 -o 0 -c
# harpoon_ctrl routing -i 6 -o 1 -c
```

   - On Single-SAI boards (i.MX 93 EVK using on-board codec):

```
# harpoon_ctrl audio -r 4 -a 00:bb:cc:dd:ca:fe
# harpoon_ctrl routing -i 3 -o 0 -c
# harpoon_ctrl routing -i 4 -o 1 -c
```

   - On Single-SAI boards with Multi-channel Codec (i.MX 93 EVK with MX93AUD-HAT):
     Connect the AVTP element to the desired audio expansion board (SAI3) output jack (J10).

```
# harpoon_ctrl audio -r 4 -H
# harpoon_ctrl routing -i 7 -o 0 -c # AVTP, stream#0 left channel to SAI3,
 left channel J10
# harpoon_ctrl routing -i 8 -o 1 -c # AVTP, stream#0 right channel to SAI3,
 right channel J10
```

5. Watch for AVTP source logs once the stream is connected (see next section):

```
INFO: avtp_source_element_st: rx stream: 0, avtp(C067ABF0, 0)
INFO: avtp_source_element_st:   connected: 1
INFO: avtp_source_element_st:   batch size: 64
INFO: avtp_source_element_st:   underflow: 459, overflow: 0 err: 0 received: 208617
INFO: avtp_source_element_st: rx stream: 1, avtp(0, 0)
INFO: avtp_source_element_st:   connected: 0
INFO: avtp_source_element_st:   batch size: 0
INFO: avtp_source_element_st:   underflow: 0, overflow: 0 err: 0 received: 0
```

4.4.4.4.4.4 AVB Talker configuration (Harpoon)

The AVB Talker implemented in Harpoon interfaces with the AVB stack through the GenAVB/TSN API, and supports audio streaming to the network while reading the audio data, through the audio pipeline, from the SAI interfaces.

To enable the AVB Talker on the Harpoon side, perform the following steps:

1. Power on the i.MX board and stop the boot process in U-Boot to fetch the AVB DTB file:

```
=> setenv jh_root_dtb imx8mp-evk-harpoon-avb.dtb
```

```
=> run jh_mmcboot
```

2. Start the audio application using the following command at the Linux prompt:
   - On FreeRTOS

```
# harpoon_set_configuration.sh freertos avb
# systemctl start harpoon
```

   - On Zephyr

```
# harpoon_set_configuration.sh zephyr avb
# systemctl start harpoon
```

3. Start the AVB pipeline, connecting the SAI input to the AVTP sink element (stream #0).
   - On Multi-SAI boards (i.MX 8M EVKs):
     Connect the HiFiBerry board (SAI5) to the AVTP element.

```
# harpoon_ctrl audio -r 4 -a 00:bb:cc:dd:de:ad
# harpoon_ctrl routing -i 1 -o 4 -c
# harpoon_ctrl routing -i 2 -o 5 -c
```

   - On Single-SAI boards (i.MX 93 EVK using on-board codec):
     Connect the on-board audio jack (SAI3).

```
# harpoon_ctrl audio -r 4 -a 00:bb:cc:dd:de:ad
# harpoon_ctrl routing -i 1 -o 2 -c
# harpoon_ctrl routing -i 2 -o 3 -c
```

   - On Single-SAI boards with Multi-channel Codec (i.MX 93 EVK with MX93AUD-HAT):
     Connect the desired audio expansion board (SAI3) input jack (J8) to the AVTP element:

```
# harpoon_ctrl audio -r 4 -H -a 00:bb:cc:dd:be:ef
# harpoon_ctrl routing -i 1 -o 8 -c # SAI3, left channel J8 to AVTP,
 stream#0 left channel
# harpoon_ctrl routing -i 2 -o 9 -c # SAI3, right channel J8 to AVTP,
 stream#0 right channel
```

4. Watch for AVTP sink logs once the stream is connected (see next section):

```
INFO: avtp_sink_element_st: rx stream: 0, avtp(C067ABF0, 0)
INFO: avtp_sink_element_st:   connected: 1
INFO: avtp_sink_element_st:   batch size: 64
INFO: avtp_sink_element_st:   underflow: 459, overflow: 0 err: 0 sent: 208617
INFO: avtp_sink_element_st: rx stream: 1, avtp(0, 0)
INFO: avtp_sink_element_st:   connected: 0
INFO: avtp_sink_element_st:   batch size: 0
INFO: avtp_sink_element_st:   underflow: 0, overflow: 0 err: 0 sent: 0
```

4.4.4.4.4.5 AVDECC controller configuration (Linux)

To enable the usage the command line AVB AVDECC controller, the AVB stack needs to be started as Endpoint AVB. For that, the GenAVB/TSN configuration files needs to be modified as follows:

1. Power on the i.MX board and let the boot process complete
2. Configure the GenAVB/TSN stack to Endpoint AVB mode by setting GENAVB_TSN_CONFIG to the right value in the GenAVB/TSN mode configuration file:

```
# vi /etc/genavb/config
```

For i.MX 8M Plus EVK:

```
GENAVB_TSN_CONFIG=2
```

For i.MX 8M Mini EVK:

```
GENAVB_TSN_CONFIG=1
```

3. Save and exit the file
4. Enable the GenAVB/TSN systemd service to start the stack automatically on next reboot:

```
# systemctl enable genavb-tsn
```

5. Reboot the board. The change is saved across reboots, so this has only to be done once.
6. Stop in U-Boot and select the AVB device tree blob before booting Linux:
For i.MX 8M Plus EVK:

```
=> setenv fdtfile imx8mp-evk-avb.dtb
=> boot
```

For i.MX 8M Mini EVK:

```
=> setenv fdtfile imx8mm-evk-avb.dtb
=> boot
```

4.4.4.4.4.6 AVB stream connection

This section describes how to use AVDECC events to connect the stream output of the Talker to the stream inputs of the Listeners. To do so, use the GenAVB AVDECC controller previously configured:

```
# genavb-controller-app -h
NXP's GenAVB AVDECC controller demo application

Usage:
app [options]

Options:
        -S, --set-control <control_type> <entity_id> <control_index> <value> Set a given control to
 the given value where control_type
                                                                       must be uint8 or utf8 (For
 utf8: <value> must be string of max 99 characters)
        -G, --get-control <control_type> <entity_id> <control_index>         Get a control value
 where control_type must be uint8 or utf8
        -l, --list-entities                                                  List discovered AVDECC
 entities
        -c, --connect-stream     <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id> <flags>   Connect a stream between a talker and a listener
        -d, --disconnect-stream  <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id>           Disconnect a stream between a talker and a listener
        -r, --get-rx-state       <listener_entity_id> <listener_unique_id>                Get
 information about a listener sink
        -t, --get-tx-state       <talker_entity_id> <talker_unique_id>                    Get
 information about a talker source
        -s, --get-tx-connection  <talker_entity_id> <talker_unique_id> <index>            Get
 information from a talker about a given connection/stream
        -T, --listener-streaming <talker_entity_id> <talker_unique_id> <start|stop>       Send
 START_STREAMING or STOP_STREAMING command to a talker
        -L, --talker-streaming   <listener_entity_id> <listener_unique_id> <start|stop>   Send
 START_STREAMING or STOP_STREAMING command to a listener
        -n, --set-name <entity_id> <configuration_id> <descritptor_type> <descriptor_index>
 <name_index> "<name>"       Send SET_NAME command to a descriptor
        --list-clock-domain-sources <entity_id> <clock_domain_index>                      List
 clock sources of a clock domain
        --set-clock-domain-source   <entity_id> <clock_domain_index> <clock_source_index>  Set the
 clock source of a clock domain
        -h                                                                                Print
 this help text
```

First of all, the Talker's entity information can be displayed by using the AVDECC controller application:

```
# genavb-controller-app -l
NXP's GenAVB AVDECC controller demo application
```

```
Number of discovered entities: 4
Entity ID = 0x49f05cf720001     Model ID = 0x49f0000080001    Capabilities = 0x8 Association ID =
 0x0   MAC address= 00:04:9F:05:CF:72   Local MAC address= 00:04:9F:05:CF:72
     Controller
     Controls:
         None


Entity ID = 0x49f070f840000     Model ID = 0x49f0000090001    Capabilities = 0x708 Association ID =
 0x0   MAC address= 00:04:9F:07:0F:84   Local MAC address= 00:04:9F:05:CF:72
     Talker:     sources = 8     capabilities = 0x4801
        Stream   0: name =      Stream output 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =      Stream output 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =      Stream output 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   3: name =      Stream output 3    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   4: name =      Stream output 4    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   5: name =      Stream output 5    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   6: name =      Stream output 6    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   7: name =      Stream output 7    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Listener:   sinks    = 8     capabilities = 0x4801
        Stream   0: name =       Stream input 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =       Stream input 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =       Stream input 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   3: name =       Stream input 3    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   4: name =       Stream input 4    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   5: name =       Stream input 5    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   6: name =       Stream input 6    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   7: name =       Stream input 7    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Controls:
         Control  0: name =     Volume Control 0    type = 0x90e0f00000000004    read-only =  No
 value_type = 1  min = 0 current = 100 max = 100 step = 1


Entity ID = 0x49fddbeef0000     Model ID = 0x49fff00000001    Capabilities = 0x708 Association ID =
 0x0   MAC address= 00:BB:CC:DD:BE:EF   Local MAC address= 00:04:9F:05:CF:72
     Talker:     sources = 3     capabilities = 0x4801
        Stream   0: name =      Stream output 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =      Stream output 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =      Stream output 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Listener:   sinks    = 3     capabilities = 0x4801
        Stream   0: name =       Stream input 0    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =       Stream input 1    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =       Stream input 2    interface index = 0    number of formats = 1
 flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
     Controls:
         Control  0: name =     Volume Control 0    type = 0x90e0f00000000004    read-only =  No
 value_type = 1  min = 0 current = 100 max = 100 step = 1
```

Once the Talker is running, its entity ID can be displayed by using the same tool:

```
Entity ID = 0x49fdddead0000     Model ID = 0x49fff00000001    Capabilities = 0x708 Association ID =
 0x0   MAC address= 00:BB:CC:DD:BE:EF   Local MAC address= 00:04:9F:05:CF:72
     Talker:     sources = 3     capabilities = 0x4801
```

```
        Stream   0: name =      Stream output 0    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =      Stream output 1    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =      Stream output 2    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
      Listener:  sinks   = 3     capabilities = 0x4801
        Stream   0: name =       Stream input 0    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   1: name =       Stream input 1    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
        Stream   2: name =       Stream input 2    interface index = 0    number of formats = 1
  flags = 0x6    current_format = 0x0205021800806000 ( AAF 2chans 24/32bits 48000Hz 6samples/packet )
      Controls:
         Control  0: name =     Volume Control 0    type = 0x90e0f00000000004    read-only =  No
  value_type = 1  min = 0 current = 100 max = 100 step = 1
```

To connect streams, use the following command:

```
# genavb-controller-app -c <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id> <flag>
```

To disconnect a stream, use the following command:

```
# genavb-controller-app -d <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id>
```

In the following example, the Listener's stream #0 is connected to the Talker’s stream #0:

```
# genavb-controller-app -c 0x49fdddead0000 0 0x49fddbeef0000 0 0
NXP's GenAVB AVDECC controller demo application
Stream connection successful: stream id = 0xbbccdddead0000  Destination MAC address
 91:E0:F0:00:FE:21 flags = 0x0 connection_count = 1 VLAN id = 0
```

If you have another Listener on the network:

```
# genavb-controller-app -c 0x49fdddead0000 0 0x49fddcafe0000 0 0
NXP's GenAVB AVDECC controller demo application
Stream connection successful: stream id = 0xbbccdddead0000  Destination MAC address
 91:E0:F0:00:FE:21 flags = 0x0 connection_count = 1 VLAN id = 0
```

Once the stream is connected, audio captured on the Talker’s audio input port can be heard on the Listener’s output port.

4.4.4.4.5 AVB: Connecting Milan Listeners and Talker through an AVB bridge

Milan mode is enabled only on the pipeline with Media Clock Recovery support.

The media clock recovery feature permits the listener to synchronize its media clock to a remote master clock through gPTP timestamps in the AVTP stream or a dedicated CRF stream.

On the boards that support this feature, the endpoint uses timestamps from the AVTP/CRF stream to tune its own audio PLL and prevent audio clock drifts with the AVB talker/ Master clock.

Note: *Media clock recovery mechanism requires hardware signal connections between the Ethernet controller and a generic timer clocked by the audio PLL. Only the i.MX 8M Plus EVK provides these hardware connections. The other boards (i.MX 8M Mini EVK, i.MX 8M Nano, and i.MX 93 EVK) use a (less accurate) fall-back mechanism based on software sampling of PTP and Audio PLL is used to perform media clock recovery.*

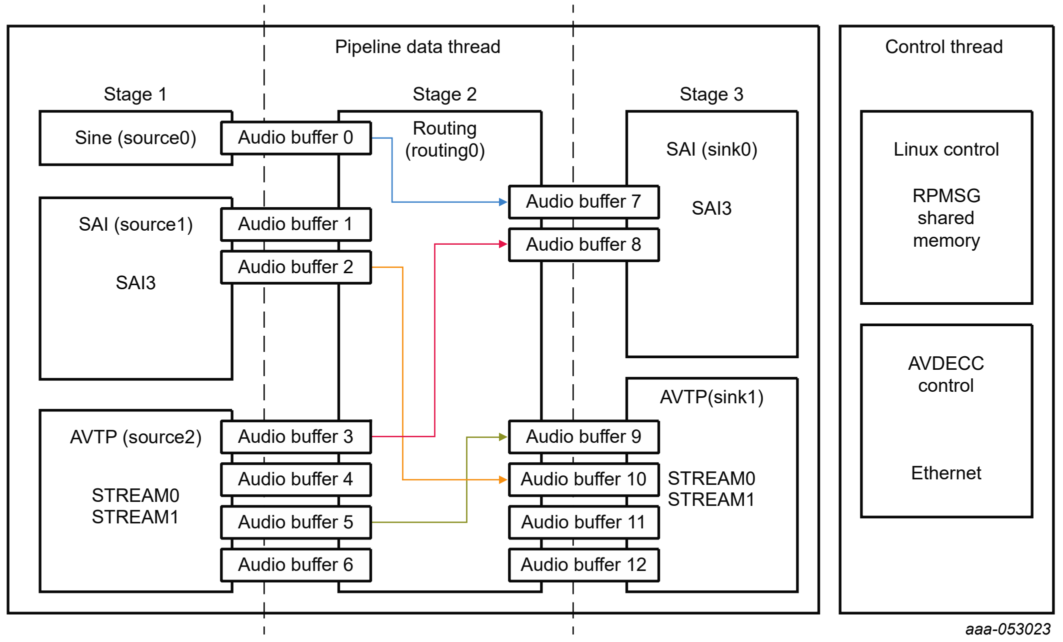


Figure 24. Milan AVB Audio pipeline with Media Clock Recovery

Table 20. Indexes of source elements

Index	Source element	Comment
0	Sine wave, 440 Hz	Software generated source
1	SAI3, On-Board Jack left channel	Hardware source
2	SAI3, On-Board Jack right channel	Hardware source
3	AVTP, stream#0 left channel	AVB source from network
4	AVTP, stream#0 right channel	AVB source from network
5	AVTP, stream#1 left channel	AVB source from network
6	AVTP, stream#1 right channel	AVB source from network

Table 21. Indexes of sink elements

Index	Sink element	Comment
0	SAI3, On-Board Jack left channel	Hardware sink
1	SAI3, On-Board Jack right channel	Hardware sink
2	AVTP, stream#0 left channel	AVB sink to network
3	AVTP, stream#0 right channel	AVB sink to network
4	AVTP, stream#1 left channel	AVB sink to network
5	AVTP, stream#1 right channel	AVB sink to network

4.4.4.4.5.1 AVB Milan setup preparation

The AVB Listeners and Talker implemented in Harpoon can be connected with each other, and support reading audio samples from the network while pushing out the audio data, through the audio pipeline, on the SAI interfaces.

- Two or more AVB endpoints

- AVDECC controller (e.g., Milan Hive Controller)
- One AVB bridge (e.g., LS1028ARDB with Real-time Edge SW v3.3 or above)

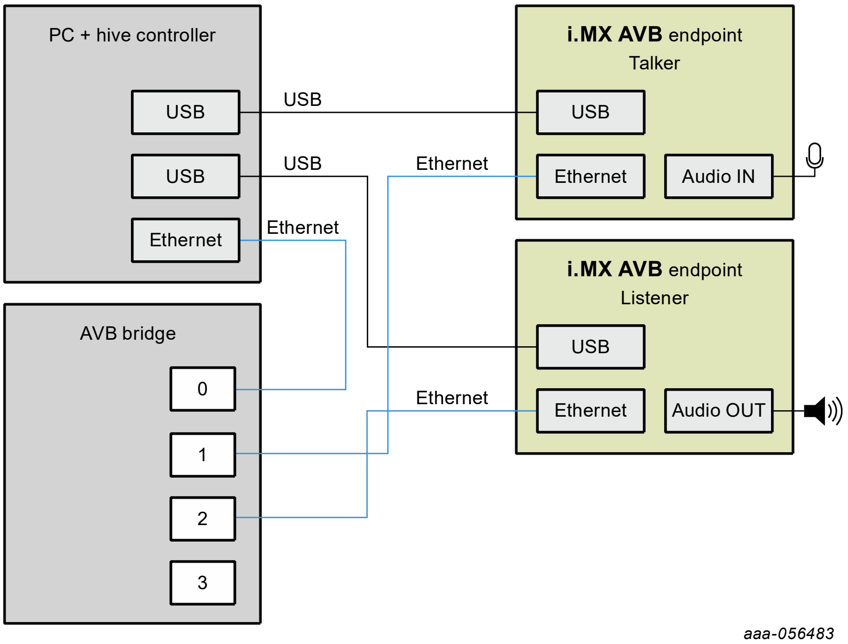


Figure 25. Harpoon AVTP Talker/Listener setup

4.4.4.4.5.2 AVB Milan Listener configuration (Harpoon)

To enable the AVB pipeline (with MCR support) on the Harpoon side, perform the following steps:

1. Power on the i.MX board and stop the boot process in U-Boot to fetch the AVB DTB file:

```
=> setenv jh_root_dtb imx8mp-evk-harpoon-avb.dtb
=> run jh_mmcboot
```

2. Start the audio application using the following command at the Linux prompt:
   - On FreeRTOS

   ```
   # harpoon_set_configuration.sh freertos avb
   # systemctl start harpoon
   ```

   - On Zephyr

   ```
   # harpoon_set_configuration.sh zephyr avb
   # systemctl start harpoon
   ```

3. Start the AVB pipeline, connecting the AVTP source element (stream #0) to the SAI3 output (on-board jack).

```
# harpoon_ctrl audio -r 6 -a 00:bb:cc:dd:be:ef
# harpoon_ctrl routing -i 3 -o 0 -c
# harpoon_ctrl routing -i 4 -o 1 -c
```

4. Observe the AVTP source logs when the stream is connected (see next section):

```
INFO: avtp_source_element_st: rx stream: 0, avtp(C067ABF0, 0)
INFO: avtp_source_element_st:   connected: 1
INFO: avtp_source_element_st:   batch size: 64
INFO: avtp_source_element_st:   underflow: 459, overflow: 0 err: 0 received:
 208617
```

```
INFO: avtp_source_element_st: rx stream: 1, avtp(0, 0)
INFO: avtp_source_element_st:   connected: 0
INFO: avtp_source_element_st:   batch size: 0
INFO: avtp_source_element_st:   underflow: 0, overflow: 0 err: 0 received: 0
```

You can see the logs about the Media Clock Recovery execution with real-time adjustments:

```
INFO 2464460.253445872 os mclock_rec_pll_stats : adjust                  = 700
INFO 2464460.253445872 os mclock_rec_pll_stats : reset                   = 1
INFO 2464460.253445872 os mclock_rec_pll_stats : start                   = 2
INFO 2464460.253445872 os mclock_rec_pll_stats : stop                    = 1
INFO 2464460.253445872 os mclock_rec_pll_stats : GPTP error              = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : GPTP start error        = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : GPTP gettime error      = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : measurement error       = 2
INFO 2464460.253445872 os mclock_rec_pll_stats : watchdog error          = 1
INFO 2464460.253445872 os mclock_rec_pll_stats : ts error                = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : drift error             = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : error (Hz/s)            = 1
INFO 2464460.253445872 os mclock_rec_pll_stats : gpt_rec event           = 878825
INFO 2464460.253445872 os mclock_rec_pll_stats : gpt_rec event fec       = 10972
INFO 2464460.253445872 os mclock_rec_pll_stats : gptp_reloaded           = 10972
INFO 2464460.253445872 os mclock_rec_pll_stats : numerator               = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : measure                 = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : err_set_pll_rate        = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : err_pll_prec            = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : last_app_adjust         = 8789
INFO 2464460.253445872 os mclock_rec_pll_stats : locked state transition = 1
```

4.4.4.4.5.3 AVB Milan Talker configuration (Harpoon)

To enable the Milan Talker AVB pipeline (with MCR support) on Harpoon side, perform the following steps:

1. Power on the i.MX board and stop the boot process in U-Boot to fetch the AVB DTB file:

```
=> setenv jh_root_dtb imx8mp-evk-harpoon-avb.dtb
=> run jh_mmcboot
```

2. Start the audio application using the following command at the Linux prompt:
   - On FreeRTOS

   ```
   # harpoon_set_configuration.sh freertos avb
   # systemctl start harpoon
   ```

   - On Zephyr

   ```
   # harpoon_set_configuration.sh zephyr avb
   # systemctl start harpoon
   ```

3. Start the AVB pipeline, connecting the AVTP source element (stream #0) to the SAI output (for example, on the board jack).

```
# harpoon_ctrl audio -r 6 -a 00:bb:cc:dd:de:ad
# harpoon_ctrl routing -i 0 -o 2 -c
# harpoon_ctrl routing -i 1 -o 3 -c
```

4. Observe the AVTP sink logs when the stream is connected (see next section):

```
INFO: avtp_sink_element_st: tx stream: 0, avtp(C067ABF0, 0)
INFO: avtp_sink_element_st:   connected: 1
INFO: avtp_sink_element_st:   batch size: 64
INFO: avtp_sink_element_st:   underflow: 459, overflow: 0 err: 0 sent: 208617
INFO: avtp_sink_element_st: tx stream: 1, avtp(0, 0)
INFO: avtp_sink_element_st:   connected: 0
```

```
INFO: avtp_sink_element_st:   batch size: 0
INFO: avtp_sink_element_st:   underflow: 0, overflow: 0 err: 0 sent: 0
```

You can also see logs about the Media Clock Recovery execution:

```
INFO 2464460.253445872 os mclock_rec_pll_stats : adjust                  = 700
INFO 2464460.253445872 os mclock_rec_pll_stats : reset                   = 1
INFO 2464460.253445872 os mclock_rec_pll_stats : start                   = 2
INFO 2464460.253445872 os mclock_rec_pll_stats : stop                    = 1
INFO 2464460.253445872 os mclock_rec_pll_stats : GPTP error              = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : GPTP start error        = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : GPTP gettime error      = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : measurement error       = 2
INFO 2464460.253445872 os mclock_rec_pll_stats : watchdog error          = 1
INFO 2464460.253445872 os mclock_rec_pll_stats : ts error                = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : drift error             = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : error (Hz/s)            = 1
INFO 2464460.253445872 os mclock_rec_pll_stats : gpt_rec event           = 878825
INFO 2464460.253445872 os mclock_rec_pll_stats : gpt_rec event fec       = 10972
INFO 2464460.253445872 os mclock_rec_pll_stats : gptp_reloaded           = 10972
INFO 2464460.253445872 os mclock_rec_pll_stats : numerator               = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : measure                 = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : err_set_pll_rate        = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : err_pll_prec            = 0
INFO 2464460.253445872 os mclock_rec_pll_stats : last_app_adjust         = 8789
INFO 2464460.253445872 os mclock_rec_pll_stats : locked state transition = 1
```

4.4.4.4.5.4 AVB Milan Connection using Hive

On the Host PC, retrieve Hive Controller’s binaries and follow the instructions to compile and install it depending on your platform. Then, follow the provided instructions to execute the Hive Controller. When the stack is started on each endpoint and the PC is connected to the same AVB switch, two (or more) Milan compatible entities would be detected and appear on the interface.

Note: *The entity ID of each endpoint is based on the configured MAC address of the network interface.*

Setting up the media clock domain

Both endpoints use the same AVDECC entity model with a default clock source set to the "Internal clock source".

The user, using the Hive controller, needs to set clock domain sources on both endpoints to a valid configuration and connect the required clock source streams.

For this use case:

- Endpoint acting as Talker for both AAF (audio) and CRF (clock): Keep and verify that the clock source is "Internal clock source" (See the left side of the figure below).
- Endpoint acting as Listener for both AAF (audio) and CRF (clock): Set the clock source to "Stream Input CRF" (See the left side of the figure below).

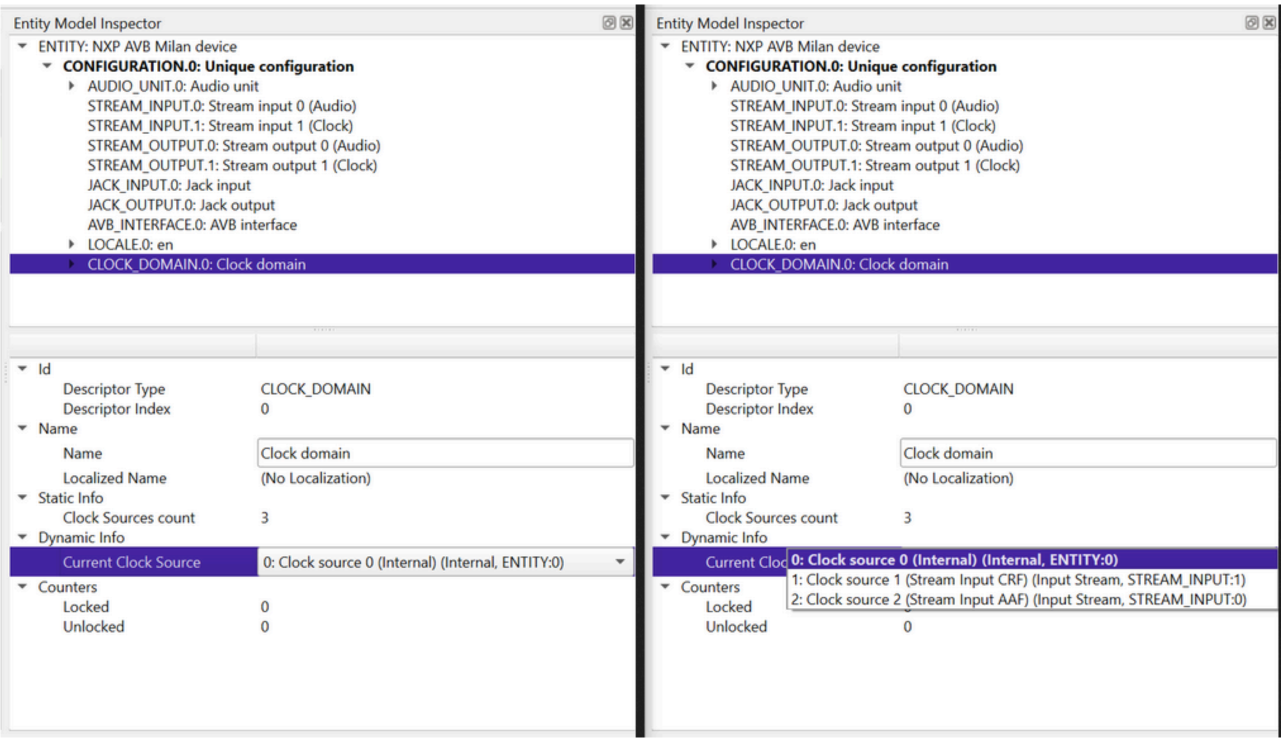


Figure 26. Hive Controller

Note:

- *If an endpoint's clock domain is set to use a stream input for its clock source, this endpoint's stream input needs to be active for the media clock to be properly set up.*
- *The endpoint acting as the CRF Listener should support media clock recovery.*

Connecting clock streams

To connect a stream, click on the corresponding box in the matrix of the **Stream Based** section (see the following figure). Once the streams are successfully connected, the selected box turns green.

Connect the CRF stream input **Stream input 1 (Clock)** of the endpoint using its CRF stream input as its clock source (the endpoint should support media clock recovery and would act as the CRF Listener) to the CRF stream output **Stream output 1 (Clock)** of the endpoint that would act as the CRF Talker.

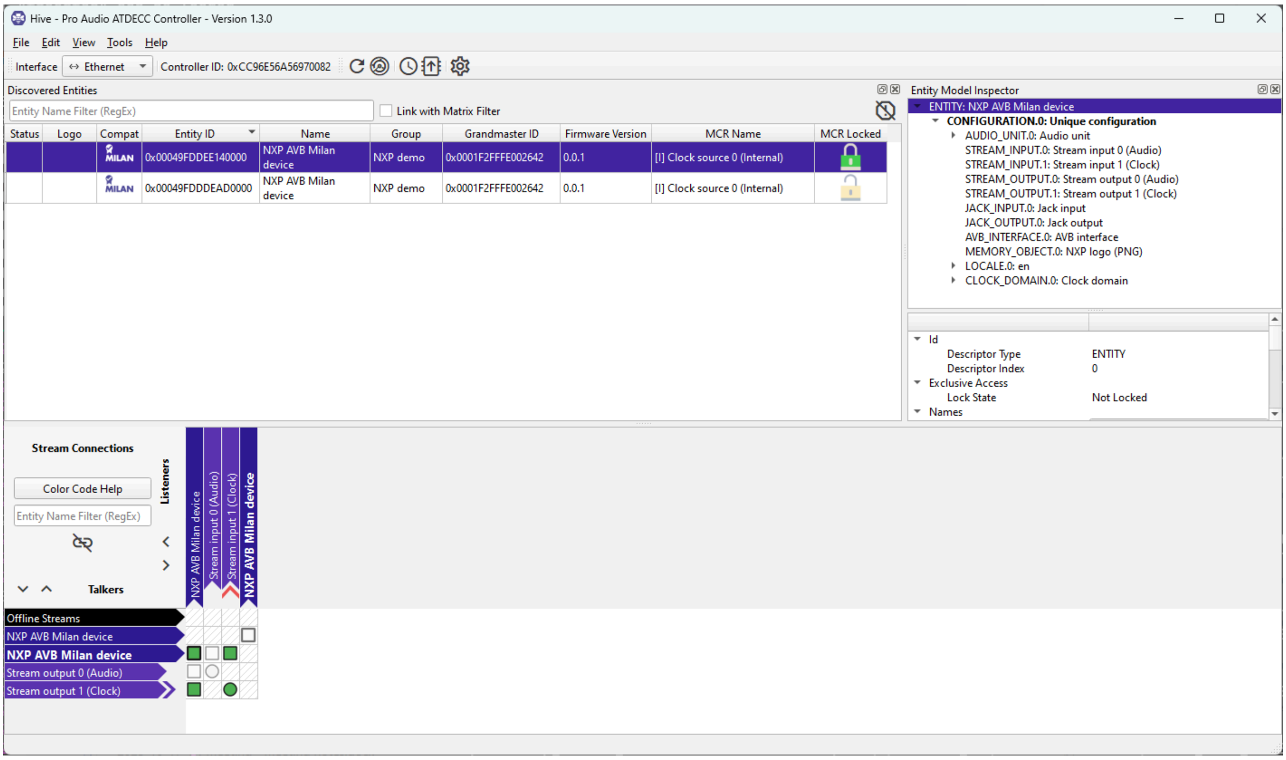


Figure 27. Hive Controller - Clock connection

A successful connection is reported with a green dot.

Connecting Streams using Hive

Connect the Audio stream input **Stream input 0 (Audio)** of the endpoint that you want to use as a Listener to the Audio stream output **Stream output 0 (Audio)** of the endpoint that you want to use as a Talker.

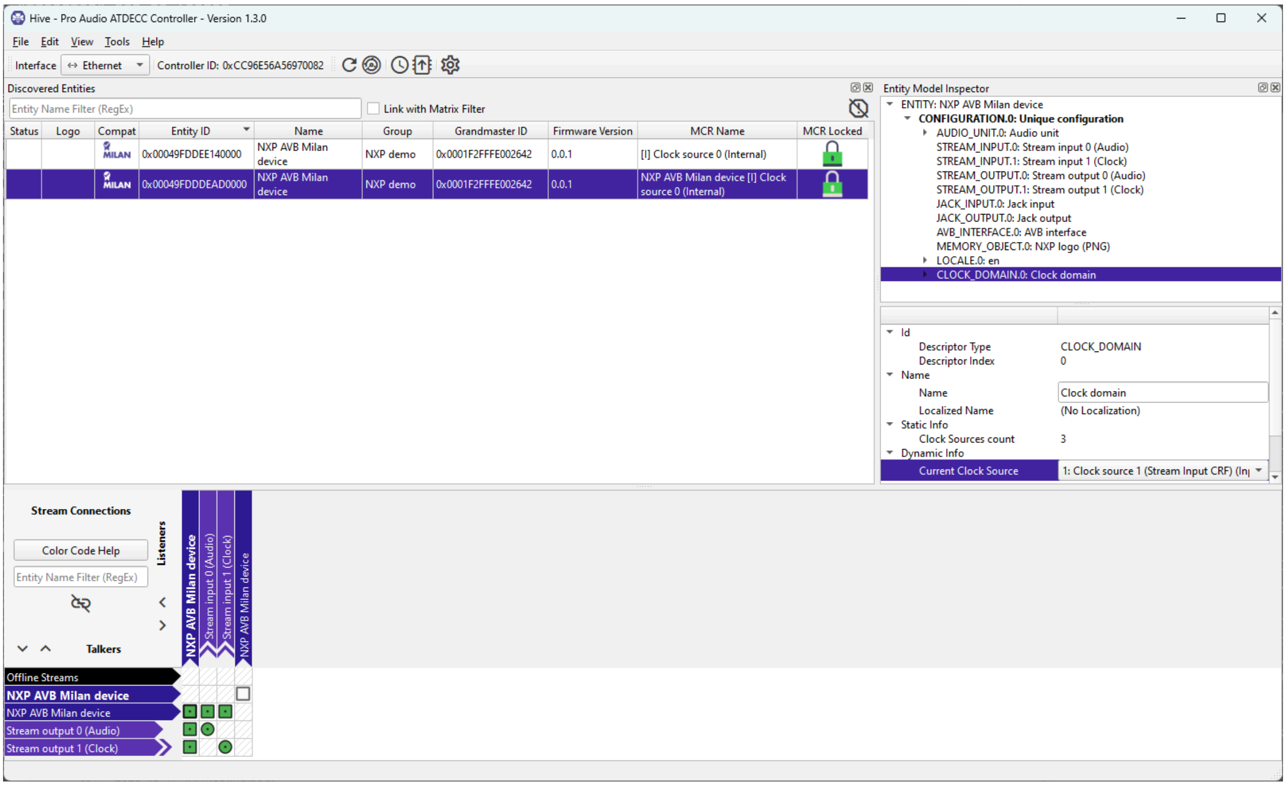


Figure 28. Hive Controller

Once the connection is reported successful (green dot), media clock recovery logs should be printed and you can hear on the Listener's output any sound passed into the Talker's audio input port.

4.4.4.5 Playing an SMP audio pipeline

The use case for SMP audio pipeline is only supported on Zephyr, which runs the SMP kernel on two CPU Cores. It creates and binds one dedicated data thread for each CPU Core.

The main motivation for SMP support is to distribute the CPU load of the pipeline processing across available cores, and thus is able to run pipelines that consume more than one single core CPU resources.

The main approach used is to split the existing pipelines in two pieces, and process them, asynchronously, in different cores/data threads. This allows the two pieces to fully run in parallel, but usually requires a one period increase in the end-to-end latency. For example:

- Before: 1 audio pipeline, running in one core/data thread. Processing period P, with an end-to-end latency of 2 x P.
- After: Pipeline is split into two 2 pipelines. Each runs on a separate core. Explicit synchronization between the two threads/pipelines is avoided by adding an extra buffer of P length between the two pipelines. Processing period is still P, but end-to-end latency is now 3 x P.

This basically models one pipeline as two independent ones:

- The first one has no sink elements. It terminates with output buffers.
- The second one has a specific source element, which implements the extra buffer between pipelines.
- The scheduling of all the thread handling is done based on the same IRQ.

This approach can also be scaled to more CPUs, each time splitting the pipeline into several pieces, each new thread/piece increasing the end-to-end latency by P.

4.4.4.5.1 Playing an SMP full audio pipeline

The reference audio application splits the pipeline used by the "full audio pipeline" use case into two audio data pipelines. Each pipeline runs on a dedicated thread bound to a dedicated CPU core.

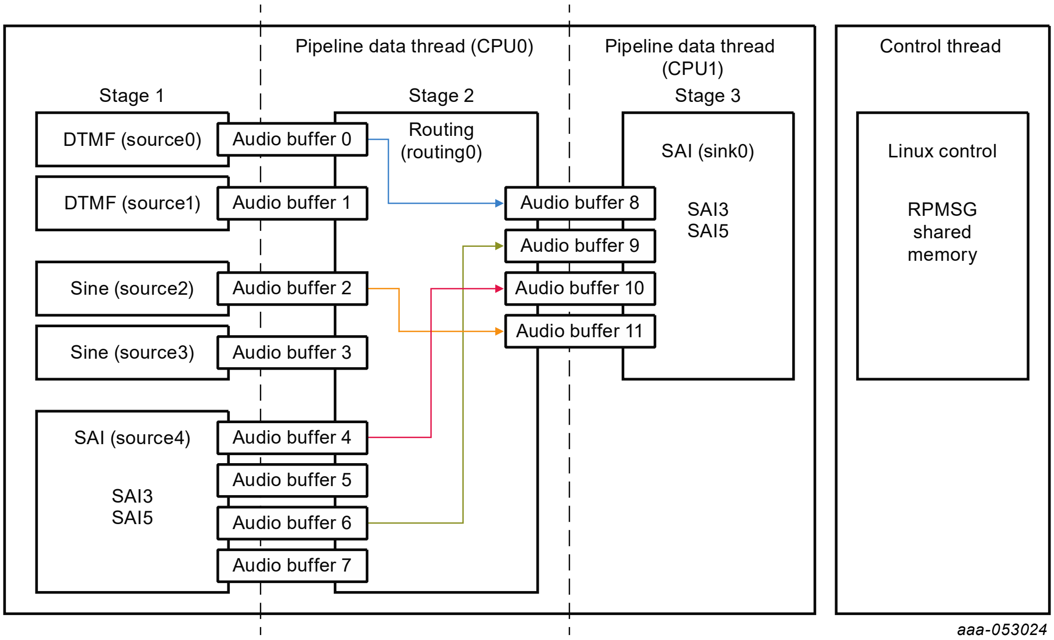


Figure 29. SMP Audio Pipeline with two threads and CPU cores

To run the Zephyr audio SMP pipeline application, run the following command to generate an appropriate configuration file:

```
# harpoon_set_configuration.sh zephyr audio_smp
```

Note: *Avoid changing the configuration while the Harpoon service is running (silent failure when restarting the service).*

Run the Harpoon service with Systemd to start Jailhouse.

```
# systemctl start harpoon
```

Then use the following command to run audio SMP pipeline testcase:

```
# harpoon_ctrl audio -r 5
```

You can then connect the provided sources to audio outputs:

```
# harpoon_ctrl routing -i 4 -o 2 -c   # SAI5's input to SAI3's output(L)
# harpoon_ctrl routing -i 5 -o 3 -c   # SAI5's input to SAI3's output(R)
```

To run another audio use case, stop the playback using the following command:

```
# harpoon_ctrl audio -s
```

4.4.4.5.2 Playing an SMP AVB audio pipeline

This pipeline runs with the same constraints as the SMP Full audio pipeline with the addition of the AVB feature. The GenAVB/TSN stack tasks are bound to the CPU Core 0, the AVTP source elements run on the CPU Core 0, and the AVTP Sink elements run on CPU Core 1.

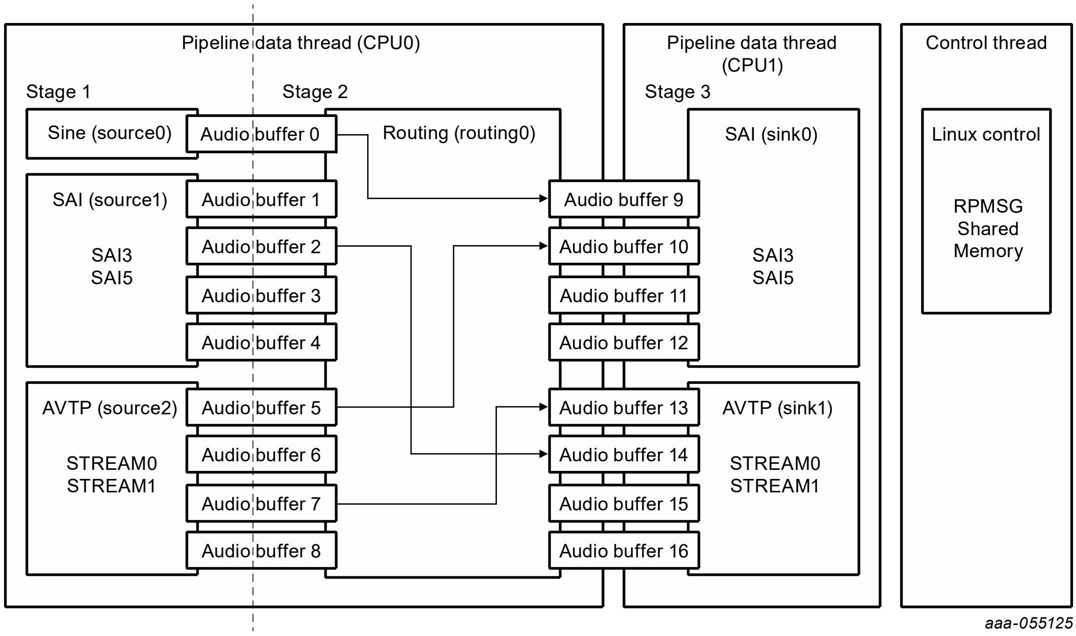


Figure 30. SMP AVB Audio Pipeline with two threads and CPU cores

To run the Zephyr audio AVB SMP pipeline application, run the following command to generate an appropriate configuration file:

```
# harpoon_set_configuration.sh zephyr avb
```

Run the Harpoon service with Systemd to start Jailhouse.

```
# systemctl start harpoon
```

Then use the following command to run the audio AVB SMP pipeline testcase:

```
# harpoon_ctrl audio -r 7
```

Or use the following command to run audio AVB SMP pipeline with Media Clock Recovery testcase:

```
# harpoon_ctrl audio -r 8
```

You can then follow the same steps as described in Playing an AVB audio Pipeline to connect streams and do the audio routing.

4.5 Industrial application

4.5.1 Features of the industrial application

The industrial application is available in the Harpoon share directory of the root file system:

```
/usr/share/harpoon/inmates/freertos/industrial.bin  # FreeRTOS binary
/usr/share/harpoon/inmates/zephyr/industrial.bin    # Zephyr binary
```

The different use cases are:

- FlexCAN-based communication (on i.MX 8M Plus EVK, i.MX 93 EVK, i.MX 95 15x15 EVK, and i.MX 95 19x19 EVK):
  - Two boards (nodes) are connected through their CAN bus connectors using a proper CAN bus cable. The latter can either be purchased or built following the CAN pinout standard.
  - Each node is configured to handle multiple message buffers, where a message buffer is configured either for transmit or receive.
  - Both nodes send/receive either CAN or CAN FD messages.
- Ethernet:
  - Simple MCUXpresso SDK API based application to send and receive packets through the ENET interface:
    - ENET application for FreeRTOS and Zephyr on i.MX 8M Mini/Nano EVK.
    - ENET_QoS application with or without internal loopback for Zephyr on i.MX 8M Plus EVK and i.MX 93 EVK.
  - Full TSN stack based application, running a gPTP stack and sending/receiving TSN packets on a TSN network:
    - Through the ENETC interface, acting as a controller/IO device on i.MX 95 15x15 LPDDR4x EVK and i.MX 95 19x19 LPDDR5 EVK.
    - Through the ENET_QOS interface, acting as a controller/IO device on i.MX 8M Plus EVK and i.MX 93 EVK.
    - Through the ENET_QOS interface, acting as a motor network controller on i.MX 8M Plus EVK and i.MX 93 EVK.
    - Through the ENET interface, acting as a controller/IO device on i.MX 8M Mini EVK and i.MX 8M Nano EVK.
      Note: *The ENET interface does not support 802.1Qbv. Packets are transmitted using basic, software based, strict priority scheduling.*

4.5.2 Starting the industrial application

To use the industrial application, Jailhouse must be started first. To start Jailhouse and the industrial application, create the corresponding Harpoon configuration file and run the harpoon service using `systemd`, for example:

```
# harpoon_set_configuration.sh freertos industrial
```

Or, to configure it for Zephyr:

```
# harpoon_set_configuration.sh zephyr industrial
```

The configuration file is stored under `/etc/harpoon/harpoon.conf` and the Harpoon systemd service uses it to start Jailhouse and the industrial application:

```
# systemctl start harpoon
```

Once the Harpoon service has been started, `harpoon_ctrl` is used to start or stop the industrial features with optional parameters. The different options for the industrial application are as follows:

```
Industrial FlexCAN options:
  -r <id>         run CAN mode id:
                  0 - Multiple Nodes and Messages Tx+Rx on the imx8mp and
                      the imx93
  -n <node_type> acting as node 'A' or 'B' (default 'A')
                  0 - node 'A'
                  1 - node 'B'
  -o <protocol>  use 'CAN' or 'CAN FD' protocol (default 'CAN')
                  0 - protocol is 'CAN'
                  1 - protocol is 'CAN FD'
  -s              stop FlexCAN based communication

Industrial ethernet options:
  -a <mac_addr>  set hardware MAC address (default 91:e0:f0:00:fe:70)
  -p <period_ns> set processing period in ns (default 100000)
  -r <id>         run ethernet mode id:
                  0 - genAVB/TSN stack with TSN application
                  1 - mcux-sdk API:
                         imx8m{m,n}: ENET on Zephyr and FreeRTOS
                         imx8mp, imx93: ENET_QoS on Zephyr
                  2 - mcux-sdk API with PHY loopback mode:
                         imx8mp, imx93: ENET_QoS on Zephyr
  -m <app_mode>  for genAVB/TSN: app mode (default 'NETWORK_ONLY', if not
 specified)
                  0 - mode is 'MOTOR_NETWORK'
                  2 - mode is 'NETWORK_ONLY'
  -i <role>       for genAVB/TSN: endpoint role (default 'controller', if not
 specified)
                  0 - role is 'IO device 0'
                  1 - role is 'IO device 1'
  -n <n_io_dev>  for 'NETWORK_ONLY' and 'MOTOR_NETWORK' app modes: number of
 connected io_devices (default is '1' if not specified. Max is '2')
                  max of two enpoints supported
  -c <ctrl_st>   for genAVB/TSN motor control: control strategy to be utilized
 (default is '0' if not specified)
                  0 - SYNCHRONIZED"
                  1 - FOLLOW"
                  2 - HOLD_INDEX"
                  3 - INTERLACED"
                  4 - STOP"
                  5 - IDENTIFY"
  -s              stop ethernet
```

4.5.3 Running the industrial application: examples

4.5.3.1 FlexCAN multiple nodes communication

4.5.3.1.1 Hardware setup

- On i.MX 8M Plus EVK:
  Connection is done through the CAN bus connector (J19) using a male DB9 adapter. The termination resistance should be added adequately.
- On i.MX 93 EVK:

Connection is done through the CAN bus connectors (J1101). The board has a DIP switch (S1101) to control the termination resistance.

- On i.MX 95 15x15 EVK:
  Connection is done through the CAN bus connectors (J22). The termination resistance is already integrated in the board.
- On i.MX 95 19x19 EVK:
  Connection is done through the CAN bus connectors (J18). The termination resistance is already integrated in the board.

The used cables can be built (or purchased) following the CAN pinout standard. Each pin should be connected to its equivalent signal between two boards.

Table 22. CAN pinouts list

Signal	i.MX 93 J1101 pins	DB9 CAN bus cable pins	i.MX 95 19x19 J18 pins	i.MX 95 15x15 J22 pins
CAN_H	Pin 2	Pin 7	Pin 2	Pin 2
CAN_L	Pin 3	Pin 2	Pin 3	Pin 3
GND	Pin 4	Pin 3	Pin 4	Pin 4

4.5.3.1.2 Industrial CAN application overview

The industrial CAN application is configured to perform communication between two nodes. Each node has four message buffers (MB) used equally for transmit and receive (two MBs for transmit and two MBs for receive). The transmission is driven by a periodic timer (currently configured at 1200 us) and reception is driven by frame reception.

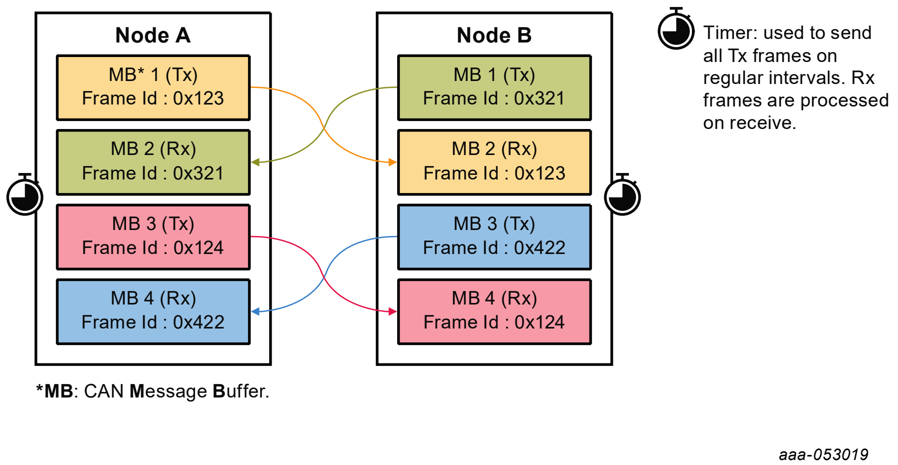


Figure 31. Industrial CAN application architecture

4.5.3.1.3 FlexCAN multiple nodes use case

To start the FlexCAN based communication:

- One board needs to be selected as Node A `-n 0` and the other as Node B `-n 1`.
- Select the same CAN protocol on both nodes: `-o 0` for CAN and `-o 1` for CAN FD. The default is the CAN protocol.

On board A, start CAN protocol multiple-node use case as Node A:

```
# harpoon_ctrl can -r 0 -n 0
```

On board B, start CAN protocol multiple-node use case as Node B:

```
# harpoon_ctrl can -r 0 -n 1
```

Type this command to stop the current use case (mandatory before starting a new use case):

```
# harpoon_ctrl can -s
```

During the execution of the application, reception and transmission logs are dumped on the console every 10 seconds.

Industrial CAN application logs example:

- Node A:

```
INFO: can_stats : |Mbit/s: 2|TX period µs: 1200|global irq: 5532306|
INFO: can_stats : |TX mb: 1, id: 123|==>|irq: 1383079|tx: 1383079|busy  : 2350|fail: 0|
INFO: can_stats : |RX mb: 2, id: 321|==>|irq: 1383074|rx: 1383074|ovrflw:    0|fail: 0|
INFO: can_stats : |TX mb: 3, id: 124|==>|irq: 1383079|tx: 1383079|busy  : 2350|fail: 0|
INFO: can_stats : |RX mb: 4, id: 422|==>|irq: 1383074|rx: 1383074|ovrflw:    0|fail: 0|
```

- Node B:

```
INFO: can_stats : |Mbit/s: 2|TX period µs: 1200|global irq: 5544926|
INFO: can_stats : |TX mb: 1, id: 321|==>|irq: 1389384|tx: 1389384|busy  :    0|fail: 0|
INFO: can_stats : |RX mb: 2, id: 123|==>|irq: 1383079|rx: 1383079|ovrflw:    0|fail: 0|
INFO: can_stats : |TX mb: 3, id: 422|==>|irq: 1389384|tx: 1389384|busy  :    0|fail: 0|
INFO: can_stats : |RX mb: 4, id: 124|==>|irq: 1383079|rx: 1383079|ovrflw:    0|fail: 0|
```

The definition of the log's key words is as follows:

- `Mbit/s`: bus baudrate. It is set to 2 Mbit/s for the CAN FD and to 1 Mbit/s for the CAN in the application example.
- `TX period µs`: transmission timer period. In this example it is set to 1200 us.
- `global irq`: global interrupts number. One global interruption may signal both RX and TX interruptions at the same time. It is possible that the global IRQ number is lower than the sum of TX and RX interruptions.
- `TX mb`: TX message buffer index.
- `RX mb`: RX message buffer index.
- `id`: frame id.
- `irq`: number of TX or RX interruptions:
  - A TX interruption is triggered when the application manages to send a message to the receiver.
  - An RX interruption is triggered when a message is received.
- `tx` and `rx`: number of reads and writes in the message buffer memory.
- `busy`: number of TX busy operations. It occurs when the application is not able to write another frame, because it is still waiting for the TX interruption from the previous one. This can also happen when the receiver is not in run mode or not configured properly.
- `ovrflw`: number of RX overflows. It occurs when the message buffer is busy and cannot receive the new frame.
- `fail`: number of reads and writes failures. It occurs when the application fails to read or write into message buffer memory.

4.5.3.2 Ethernet through MCUXpresso SDK API

A simple reference use case is given to exchange Ethernet packets using the SDK API.

1. Run the ENET test case on i.MX 8M Mini/Nano EVK.

```
# harpoon_ctrl ethernet -r 1
```

One possibility to verify that the use case is functional is to plug an Ethernet cable on the Ethernet connector on one end, and to a Linux host computer on the other end.
The expected output on the inmate cell console is as follows:

```
ENET test start.
ENET: Wait for PHY link up...
ENET: PHY link speed 1000M full-duplex
INFO: ethernet_sdk_enet_stat: not implemented
INFO: cpu_load_stats        : CPU load: 0.00%
ENET test result:
         TX: total = 100; succ = 100; fail = 0
         RX: total = 100; succ = 0; fail = 0; empty = 100
```

To verify that data are successfully received on the host side, use the `tcpdump` tool (sudo permissions may be required):

```
$ tcpdump -i <INTERFACE> -e
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp1s2, link-type EN10MB (Ethernet), capture size 262144 bytes
11:48:40.402104 00:04:9f:06:96:36 (oui Freescale) > 01:80:c2:00:00:0e (oui Unknown),
 ethertype LLDP (0x88cc), length 269: LLDP, length 255: imx8mp-lpddr4-evk
11:48:46.648227 00:00:00:00:00:00 (oui Ethernet) > Broadcast, 802.3, length 986: LLC,
 dsap Null (0x00) Individual, ssap Null (0x00) Response, ctrl 0x0302: Information,
 send seq 1, rcv seq 1, Flags [Final], length 986
 0x0000: 0001 0203 0405 0607 0809 0a0b 0c0d 0e0f ................
 0x0010: 1011 1213 1415 1617 1819 1a1b 1c1d 1e1f ................
 0x0020: 2021 2223 2425 2627 2829 2a2b 2c2d 2e2f .!"#$%&'()*+,-./
 0x0030: 3031 3233 3435 3637 3839 3a3b 3c3d 3e3f 0123456789:;<=>?
 0x0040: 4041 4243 4445 4647 4849 4a4b 4c4d 4e4f @ABCDEFGHIJKLMNO
 0x0050: 5051 5253 5455 5657 5859 5a5b 5c5d 5e5f PQRSTUVWXYZ[\]^_
 0x0060: 6061 6263 6465 6667 6869 6a6b 6c6d 6e6f `abcdefghijklmno
 0x0070: 7071 7273 7475 7677 7879 7a7b 7c7d 7e7f pqrstuvwxyz{|}~.
 0x0080: 8081 8283 8485 8687 8889 8a8b 8c8d 8e8f ................
 0x0090: 9091 9293 9495 9697 9899 9a9b 9c9d 9e9f ................
 0x00a0: a0a1 a2a3 a4a5 a6a7 a8a9 aaab acad aeaf ................
 0x00b0: b0b1 b2b3 b4b5 b6b7 b8b9 babb bcbd bebf ................
 0x00c0: c0c1 c2c3 c4c5 c6c7 c8c9 cacb cccd cecf ................
 0x00d0: d0d1 d2d3 d4d5 d6d7 d8d9 dadb dcdd dedf ................
 0x00e0: e0e1 e2e3 e4e5 e6e7 e8e9 eaeb eced eeef ................
 0x00f0: f0f1 f2f3 f4f5 f6f7 f8f9 fafb fcfd fe00 ................
 0x0100: 0102 0304 0506 0708 090a 0b0c 0d0e 0f10 ................
 0x0110: 1112 1314 1516 1718 191a 1b1c 1d1e 1f20 ................
 0x0120: 2122 2324 2526 2728 292a 2b2c 2d2e 2f30 !"#$%&'()*+,-./0
 0x0130: 3132 3334 3536 3738 393a 3b3c 3d3e 3f40 123456789:;<=>?@
 0x0140: 4142 4344 4546 4748 494a 4b4c 4d4e 4f50 ABCDEFGHIJKLMNOP
 0x0150: 5152 5354 5556 5758 595a 5b5c 5d5e 5f60 QRSTUVWXYZ[\]^_`
 0x0160: 6162 6364 6566 6768 696a 6b6c 6d6e 6f70 abcdefghijklmnop
 0x0170: 7172 7374 7576 7778 797a 7b7c 7d7e 7f80 qrstuvwxyz{|}~..
 0x0180: 8182 8384 8586 8788 898a 8b8c 8d8e 8f90 ................
 0x0190: 9192 9394 9596 9798 999a 9b9c 9d9e 9fa0 ................
 0x01a0: a1a2 a3a4 a5a6 a7a8 a9aa abac adae afb0 ................
<snip>
```

2. Run the ENET_QoS test case on i.MX 8M Plus EVK or i.MX 93 EVK.
This use case is only supported on Zephyr.

```
# harpoon_ctrl ethernet -r 1
```

One possibility to verify that the use case is functional is to plug an Ethernet cable on the Ethernet connector on one end, and to a Linux host computer on the other end. Use the `tcpdump` tool on the Linux host to verify that the packets are received correctly.
The expected output on the inmate cell console is as follows:

```
INFO: main_task : Industrial application started!
INFO: industrial_set_hw_addr: 00:bb:cc:dd:ee:14
INFO: enet_qos_init : enet_qos_init
INFO: ethernet_sdk_enet_run :
INFO: ethernet_sdk_enet_run : ####################
INFO: ethernet_sdk_enet_run : # #
INFO: ethernet_sdk_enet_run : # enet_qos_app #
INFO: ethernet_sdk_enet_run : # #
INFO: ethernet_sdk_enet_run : ####################
INFO: ethernet_sdk_enet_run : Wait for PHY init...
INFO: ethernet_sdk_enet_run : PHY setup was finalized
INFO: ethernet_sdk_enet_run :
30 frames ----> will be sent in 3 queues, and frames will be received in 3
 queues.
INFO: ethernet_sdk_enet_run : The frames transmitted from the ring 0, 1, 2 is
 10, 10, 10, total 30 frames!
INFO: ethernet_sdk_enet_run : The frames received from the ring 0, 1, 2 is 0,
 0, 0, total 0 frames!
INFO: ethernet_sdk_enet_run : ENET QOS TXRX Test Done0
```

3. Run the ENET_QoS Loopback test case on i.MX 8M Plus EVK or i.MX 93 EVK:
   This use case is only supported on Zephyr.

```
# harpoon_ctrl ethernet -r 2
```

   For this test case, the PHY internal loopback is enabled, so the packets sent out by the ENET_QoS port are looped back and the port receives these packets transmitted.
   The expected output on the inmate cell console is as follows:

```
INFO: main_task : Industrial application started!
INFO: industrial_set_hw_addr: 00:bb:cc:dd:ee:14
INFO: enet_qos_init : enet_qos_init
INFO: ethernet_sdk_enet_run :
INFO: ethernet_sdk_enet_run : ####################
INFO: ethernet_sdk_enet_run : # #
INFO: ethernet_sdk_enet_run : # enet_qos_app #
INFO: ethernet_sdk_enet_run : # #
INFO: ethernet_sdk_enet_run : ####################
INFO: ethernet_sdk_enet_run : Wait for PHY init...
INFO: ethernet_sdk_enet_run : PHY setup was finalized
INFO: ethernet_sdk_enet_run :
30 frames ----> will be sent in 3 queues, and frames will be received in 3
 queues.
INFO: ethernet_sdk_enet_run : The frames transmitted from the ring 0, 1, 2 is
 10, 10, 10, total 30 frames!
INFO: ethernet_sdk_enet_run : The frames received from the ring 0, 1, 2 is
 10, 10, 10, total 30 frames!
INFO: ethernet_sdk_enet_run : ENET QOS TXRX Loopback Test PASSED0
```

4.5.3.3 Ethernet with GenAVB/TSN stack

A more complex Ethernet use case uses the GenAVB/TSN Stack, which provides advanced implementation for AVB and Time-Sensitive Networking (TSN) features. Some functions for the latter do require special TSN hardware support, available in the i.MX 8M Plus, i.MX 93, and i.MX 95 SoCs.

The following sections give some details on the hardware requirements, setup preparation, and test execution.

As far as the Harpoon demonstration goes, the controller (i.MX 8M Plus, i.MX 93, i.MX 95) runs in the Cortex-A53/A55 FreeRTOS or Zephyr cell. The IO devices, which can be any TSN endpoint (i.MX 8M Plus, i.MX 93, i.MX 95, i.MX RT1170, etc.) and the TSN bridge complete the TSN network environment for this use case.

4.5.3.3.1 Requirements

- Two TSN endpoints (i.MX 8M Plus EVK, i.MX 93 EVK, i.MX 95 15x15 LPDDR4x EVK, i.MX 95 19x19 LPDDR5 EVK, or optionally an i.MX RT1170 EVK)
- One TSN bridge (LS1028ARDB)

Note: *The second IO Device is optional.*

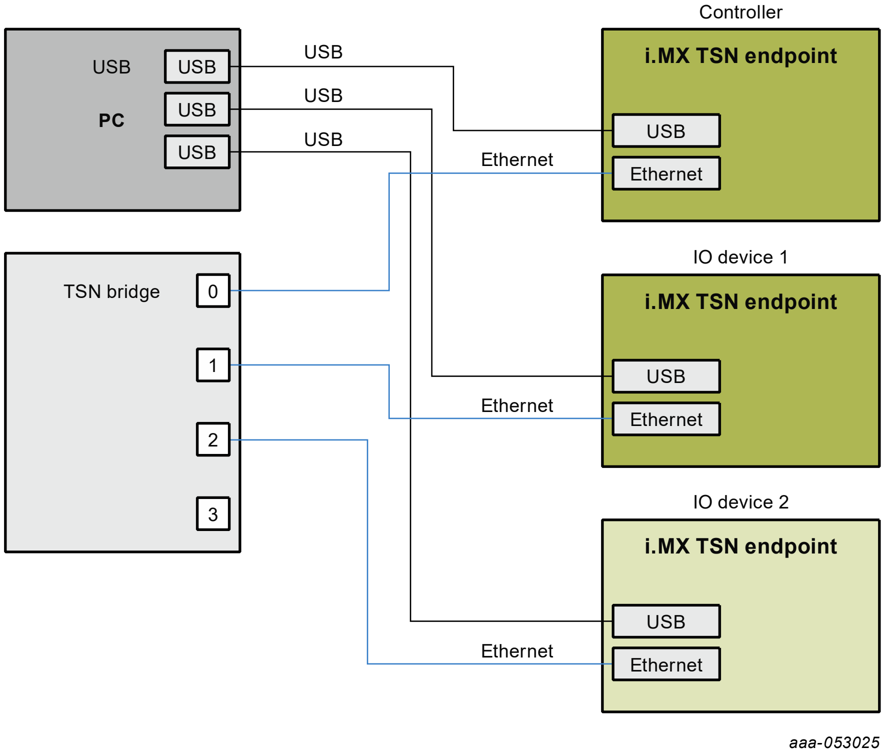


Figure 32. TSN endpoint sample application setup

4.5.3.3.2 Hardware setup

- On i.MX 8M Plus EVK:
  Connection is done through the ENET1 port (J8) using an RJ45 Ethernet cable.
- On i.MX 93 EVK:
  Connection is done through the ENET1 port (J501) using an RJ45 Ethernet cable.
- On i.MX 95 15x15 EVK:
  Connection is done through the ENET1 port (J8), which corresponds to the ENETC0 interface, using an RJ45 Ethernet cable.
- On i.MX 95 19x19 EVK:
  Connection is done through the ENET1 port (J11), which corresponds to the ENETC0 interface, using an RJ45 Ethernet cable.

4.5.3.3.3 Setup preparation

One of the TSN endpoint needs to be configured as “controller” and the other one as “IO device”. Both endpoints are connected to the TSN bridge.

4.5.3.3.3.1 i.MX RT1170 TSN Endpoint - IO device (optional)

To use an i.MX RT1170 as the IO device, first flash the latest GenAVB/TSN Endpoint image from GenAVB_TSN_RFP_7_0_0 for i.MXRT1170/RT1050 on MCUXpresso-SDK_24_12_0.

When the i.MX RT1170 is flashed, press **Insert** and set the following parameters:

```
IO_DEVICE_0>>write tsn_app/role 1
IO_DEVICE_0>>write tsn_app/period_ns 100000
```

Press **Insert** to exit the configuration mode and reboot.

4.5.3.3.3.2 TSN Bridge

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

By default, LS1028ARDB does not forward packets if no bridge interface is configured under Linux OS. Enabling bridge interface is dependent on the board used.

TSN Bridge configuration

Use the following commands to configure bridge on LS1028ARDB (with Real-time Edge SW v3.3 or above):

```
# 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 dev eno2 up
# ip link add name br0 type bridge
# ip link set br0 up
# ip link set master br0 swp0 up
# ip link set master br0 swp1 up
# ip link set master br0 swp2 up
# ip link set master br0 swp3 up
```

Then, start gPTP:

```
# tsn.sh start
```

TSN Bridge logging

Logs are stored in `/var/log/tsn-br`.

- Linux command:

```
# tail -f /var/log/tsn-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), *Link status*, *AS capability* and *Port Role* are printed for each port.

```
Port(0): domain(0, 0): Role: Master   Link: Up  asCapable: Yes neighborGptpCapable: Yes
 delayMechanism: P2P
Port(0): Propagation delay (ns): 334.29                 min    329 avg    333 max    342 variance
 17
Port(1): domain(0, 0): Role: Disabled Link: Down asCapable: No neighborGptpCapable: No
 delayMechanism: P2P
Port(2): domain(0, 0): Role: Master   Link: Up  asCapable: Yes neighborGptpCapable: Yes
 delayMechanism: P2P
Port(2): Propagation delay (ns): 386.54                 min    380 avg    385 max    390 variance
  9
Port(3): domain(0, 0): Role: Disabled Link: Down asCapable: No neighborGptpCapable: No
 delayMechanism: P2P
Port(4): domain(0, 0): Role: Disabled Link: Down asCapable: No neighborGptpCapable: No
 delayMechanism: P2P
```

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*.

4.5.3.3.4 Running the TSN use case

To start the Ethernet use case from the inmate cell (acting as a TSN Endpoint - Controller), run the following command:

```
# harpoon_ctrl ethernet -r 0
```

To start the Ethernet use case from the inmate cell (acting as a TSN Endpoint - IO Device), run the following command:

```
# harpoon_ctrl ethernet -r 0 -i 0
```

The expected initialization output in the inmates consoles is:

```
INFO: main_task             : Industrial application started!
INFO: rpmsg_init            : RPMSG init ...
INFO: rpmsg_init            : RPMSG link up
INFO: industrial_set_hw_addr: 00:bb:cc:dd:ee:14
INFO: ethernet_avb_tsn_init : ethernet_avb_tsn_init
INFO  0 app gavb_stack_init          : talker_entity_id 0x0000000000000000
INIT  0.000000000 os     genavb_init            : NXP's GenAVB/TSN stack version XXXXX

[...]

INIT  0.000000000 os     phy_task               : started
INIT  0.000000000 os     net_tx_task            : networking(C0624B70) tx task started
INIT  0.000000000 os     net_rx_task            : networking(C0624850) rx task started
INIT  0.000000000 os     net_task_init          : networking started
INIT  0.000000000 os     management_task        : management task started
INIT  0.000000000 os     management_task        : started
INIT  0.000000000 os     management_task_init   : management main completed
INIT  0.000000000 os     gptp_task              : gptp task started
INIT  0.006209075 os     gptp_task_init         : gptp main completed
INIT  0.006209075 os     srp_task               : srp task started
INIT  0.006209075 os     srp_task               : started
INIT  0.006209075 os     srp_task_init          : srp main completed

[...]

INFO: ethernet_avb_tsn_run  : tsn_app config
INFO: ethernet_avb_tsn_run  : mode             : NETWORK_ONLY
INFO: ethernet_avb_tsn_run  : role             : 0
```

```
INFO: ethernet_avb_tsn_run  : num_io_devices   : 1
INFO: ethernet_avb_tsn_run  : motor_offset     : 0
INFO: ethernet_avb_tsn_run  : control_strategy : 0
INFO: ethernet_avb_tsn_run  : app period       : 100000
INFO: ethernet_avb_tsn_run  : BUILD_MOTOR disabled, MOTOR_NETWORK and MOTOR_LOCAL modes cannot be
 used
```

After a few seconds, TSN Endpoints should be synchronized through gPTP and exchanging packets at the rate of 10000 packets per second. To observe this behavior, check the logs. If an endpoint has gPTP running correctly, the following log should appear:

```
Port(0): domain(0, 0): Role: Slave    Link: Up  asCapable: Yes
 neighborGptpCapable: Yes delayMechanism: P2P
Port(0): Propagation delay (ns): 340.13   min    331 avg    339 max    347
 variance    25
```

If the endpoint is grand master, the role field should be “Master”; otherwise, it should be “Slave”. If the application socket is correctly receiving packets, “link up” should be shown.

```
socket_stats_print        : link up
```

Between two appearances of the following log, the number of valid frames should be incremented by 50000 (10000 pps for 5 seconds):

```
socket_stats_print        : cyclic rx socket(C0605A80) net_sock(C0666820) peer
 id: 1
socket_stats_print        : valid frames  : XXXXX
socket_stats_print        : err id        : 0
socket_stats_print        : err ts        : XXXXX
socket_stats_print        : err underflow : XXXXX
```

To stop the Ethernet use case (to eventually restart it), the previous commands must be stopped with the following command:

```
# harpoon_ctrl ethernet -s
```

4.5.3.4 Ethernet motor control with GenAVB/TSN stack

The industrial reference application also provides a TSN Motor Control use case, using the GenAVB/TSN Stack capabilities on multiple i.MX platforms. The *GenAVB/TSN Stack FreeRTOS Evaluation User's Guide* (GAVBFREVALUG) from GenAVB_TSN_RFP_7_0_0 for i.MXRT1170/RT1050 on MCUXpresso-SDK_24_12_0 provides detailed description about the TSN Distributed Motor Control Application and the required configurations (and hardware rework) for i.MX RT endpoints used as TSN Motor IO Devices.

The following sections describe the use of Harpoon i.MX Endpoints (i.MX 8M Plus or i.MX 93) as TSN Motor Controller.

4.5.3.4.1 Requirements

- One TSN endpoint acting as Controller (i.MX 93 EVK)
- Two TSN endpoints acting as IO-devices (i.MX RT1170 EVK)
- One TSN bridge (LS1028ARDB)

Note: *The second IO Device is optional.*

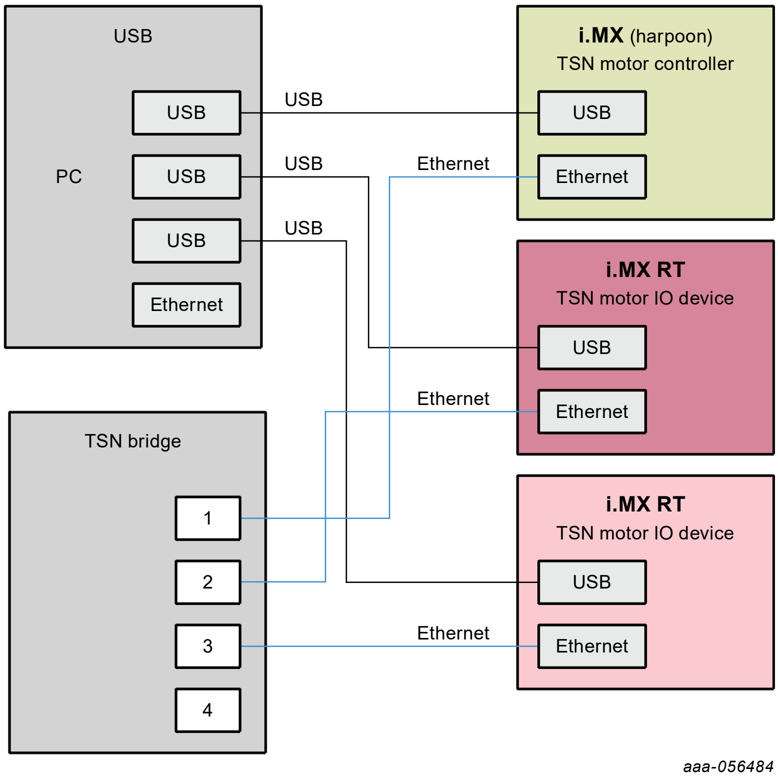


Figure 33. TSN motor control endpoints sample application setup

4.5.3.4.2 Hardware setup

- On i.MX 8M Plus EVK:
  Connection is done through the ENET1 port (J8) using an RJ45 Ethernet cable.
- On i.MX 93 EVK:
  Connection is done through the ENET1 port (J501) using an RJ45 Ethernet cable.

4.5.3.4.3 Setup preparation

The i.MX 93 or i.MX 8M Plus TSN endpoint needs to be configured as “Motor controller” and the other ones (i.MX RT1170) as “Motor IO devices”. All endpoints are connected through the TSN bridge.

4.5.3.4.3.1 i.MX RT1170 TSN Endpoint - Motor IO device

To use an i.MX RT1170 as the IO device, first flash the latest GenAVB/TSN Endpoint from GenAVB_TSN_RFP_7_0_0 for i.MXRT1170/RT1050 on MCUXpresso-SDK_24_12_0.

When the i.MX RT1170 is flashed, press the **Insert** key and set the following parameters:

```
IO_DEVICE_0>>write tsn_app/role 1
IO_DEVICE_0>>write tsn_app/mode 2
IO_DEVICE_0>>write tsn_app/period_ns 100000
```

If two endpoints are used, for the second one, provide the same parameters, **except for the role, which must be '2'**.

Press the **Insert** key to exit the configuration mode and reboot.

Note: *For more details about the i.MX RT configuration, see the GenAVB/TSN Stack FreeRTOS Evaluation User's Guide (GAVBFREVALUG), which is part of the GenAVB/TSN MCUXpresso SDK tarball.*

4.5.3.4.3.2 TSN Bridge

Use the same configuration as in TSN Bridge

4.5.3.4.4 Running the TSN Motor Control use case

To start the Ethernet motor control use case from the inmate cell (acting as a TSN Endpoint - Controller in MOTOR_NETWORK mode with a single IO Device), run the following command:

```
# harpoon_ctrl ethernet -r 0 -m 2
```

Note: *Additional and different configuration parameters may be provided, namely, the number of IO-devices and the motor control strategy to be employed.*

The expected initialization output in the inmates consoles is:

```
INFO: main_task             : Industrial application started!
INFO: rpmsg_init            : RPMSG init ...
INFO: rpmsg_init            : RPMSG link up
INFO: industrial_set_hw_addr: 00:bb:cc:dd:ee:14
INFO: ethernet_avb_tsn_init : ethernet_avb_tsn_init
INFO          0 app gavb_stack_init         : talker_entity_id 0x0000000000000000
INIT          0.000000000 os     genavb_init                      : NXP's GenAVB/TSN stack version
 dev-d71ce4fc

[...]

INIT          0.000000000 os     phy_task                         : started
INIT          0.000000000 os     net_tx_task                      : networking(C0624B70) tx task
 started
INIT          0.000000000 os     net_rx_task                      : networking(C0624850) rx task
 started
INIT          0.000000000 os     net_task_init                    : networking started
INIT          0.000000000 os     management_task                  : management task started
INIT          0.000000000 os     management_task                  : started
INIT          0.000000000 os     management_task_init             : management main completed
INIT          0.000000000 os     gptp_task                        : gptp task started
INIT          0.006209075 os     gptp_task_init                   : gptp main completed
INIT          0.006209075 os     srp_task                         : srp task started
INIT          0.006209075 os     srp_task                         : started
INIT          0.006209075 os     srp_task_init                    : srp main completed

[...]

INFO: ethernet_avb_tsn_run  : tsn_app config
INFO: ethernet_avb_tsn_run  : mode             : MOTOR_NETWORK
INFO: ethernet_avb_tsn_run  : role             : 0
INFO: ethernet_avb_tsn_run  : num_io_devices   : 2
INFO: ethernet_avb_tsn_run  : motor_offset     : 0
INFO: ethernet_avb_tsn_run  : control_strategy : 3
INFO: ethernet_avb_tsn_run  : app period       : 100000
INFO          0.002281937 os     hw_timer_request                 : hw_timer(D0605C58) pps
INFO          0.002281937 os     os_timer_create                  : os_timer(D06562D0), queue: 0
INFO          0 app gavb_pps_init           : success, clk_id: 1
INFO          0 app control_strategy_context_init: Strategy successfuly initialized
INFO          0 app motor_params_dump        : motor1:0 params:
INFO          0 app motor_params_dump        : vel max: 4000.000000 (rpm)
INFO          0 app motor_params_dump        : acc max: 12000.000000 (rpm/s)
INFO          0 app motor_params_dump        : J:       0.000150 (A.s/rpm)
INFO          0 app motor_params_dump        : b:       0.000110 (A/rpm)
INFO          0 app motor_params_dump        : Tm:      0.065000 (A)
INFO          0 app control_strategy_register_motor: Registered motor with :
INFO          0 app control_strategy_register_motor:     IO device ID : 1
INFO          0 app control_strategy_register_motor:     Motor ID     : 0
INFO          0 app control_strategy_register_motor:     Internal ID  : 256
```

```
INFO          0 app motor_params_dump        : motor2:0 params:
INFO          0 app motor_params_dump        : vel max: 4000.000000 (rpm)
INFO          0 app motor_params_dump        : acc max: 12000.000000 (rpm/s)
INFO          0 app motor_params_dump        : J:       0.000150 (A.s/rpm)
INFO          0 app motor_params_dump        : b:       0.000110 (A/rpm)
INFO          0 app motor_params_dump        : Tm:      0.065000 (A)
INFO          0 app control_strategy_register_motor: Registered motor with :
INFO          0 app control_strategy_register_motor:     IO device ID : 2
INFO          0 app control_strategy_register_motor:     Motor ID     : 0
INFO          0 app control_strategy_register_motor:     Internal ID  : 512
INFO          0 app cyclic_task_init         : cyclic task type: 0, id: 0
```

After a few seconds, TSN Endpoints should be synchronized through gPTP and exchanging packets at the rate of 10000 packets per second. To observe this behavior, check the logs. If an endpoint has gPTP running correctly, the following log should appear:

```
Port(0): domain(0, 0): Role: Slave    Link: Up  asCapable: Yes
 neighborGptpCapable: Yes delayMechanism: P2P
Port(0): Propagation delay (ns): 340.13   min    331 avg    339 max    347
 variance    25
```

If the endpoint is grand master, the role field should be “Master”; otherwise, it should be “Slave”. If the application socket is correctly receiving packets, “link up” should be shown.

```
socket_stats_print       : link up
```

Between two appearances of the following log, the number of valid frames should be incremented by 50000 (10000 pps for 5 seconds):

```
socket_stats_print       : cyclic rx socket(C0605A80) net_sock(C0666820) peer
 id: 1
socket_stats_print       : valid frames  : XXXXX
socket_stats_print       : err id        : 0
socket_stats_print       : err ts        : XXXXX
socket_stats_print       : err underflow : XXXXX
```

Furthermore, if the motor control application has been properly configured, the following logs should be shown periodically:

```
INFO  1718811593 app controller_stats_print   : current state           :
 CONTROL
INFO  1718811593 app controller_stats_print   : state control           : 58131
INFO  1718811593 app controller_stats_print   : state io_device missing : 61871
INFO  1718811593 app controller_stats_print   : state standby           : 0
INFO  1718811593 app controller_stats_print   : errors msg id: 0, src id: 0,
 motor id: 0, empty data: 0
INFO  1718811593 app controller_stats_print   : errors strat loop: 0, strat
 next: 0
INFO  1718811593 app control_strategy_stats_print: ctx(d0627e48):
INFO  1718811593 app control_strategy_stats_print:   state              : STRATEGY
INFO  1718811593 app control_strategy_stats_print:   current strategy :
 INTERLACED
INFO  1718811593 app control_strategy_stats_print:   old strategy     :
 INTERLACED
INFO  1718811593 app control_strategy_motor_stats_print: ctx(d0668550) io_device
 id: 2, motor id: 0
INFO  1718811593 app control_strategy_motor_stats_print:   startup offset : 0
INFO  1718811593 app control_strategy_motor_stats_print:   pos real       :
 0.219500
```

```
INFO  1718811593 app control_strategy_motor_stats_print:   pos target     :
 0.219698
INFO  1718811593 app control_strategy_motor_stats_print:   speed real     :
 7.344162
INFO  1718811593 app control_strategy_motor_stats_print:   errors margin: 0,
 margin stop: 0
INFO: stats_print           : stats(D06686F8) pos err min 0 mean 1 max 20 rms^2
 12 stddev^2 9 absmin 0 absmax 28
INFO: hist_print            : n_slot 181 slot_size 1
INFO: hist_print            : 34909 358 51 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0
INFO  1718811593 app control_strategy_motor_stats_print: ctx(d0667a50) io_device
 id: 1, motor id: 0
INFO  1718811593 app control_strategy_motor_stats_print:   startup offset : 0
INFO  1718811593 app control_strategy_motor_stats_print:   pos real       :
 0.249750
INFO  1718811593 app control_strategy_motor_stats_print:   pos target     :
 0.250000
INFO  1718811593 app control_strategy_motor_stats_print:   speed real     :
 -5.097555
INFO  1718811593 app control_strategy_motor_stats_print:   errors margin: 1,
 margin stop: 0
INFO: stats_print           : stats(D0667BF8) pos err min 0 mean 4 max 70 rms^2
 57 stddev^2 33 absmin 0 absmax 70
INFO: hist_print            : n_slot 181 slot_size 1
INFO: hist_print            : 33720 1046 201 142 98 30 80 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0
INFO  1718811593 app control_strategy_motor_stats_print:   last margin error:
 45.000000
```

To stop the Ethernet motor control use case (to eventually restart it), the previous commands must be stopped with the following command:

```
# harpoon_ctrl ethernet -s
```

4.6 rt_latency application

The `rt_latency` application is a simple benchmark application for real-time OS that measures the latency (Time delta, in nanoseconds) between hardware IRQ events and software actions:

- `irq delay`: time to enter in the software IRQ handler after a hardware IRQ occurs (hardware + hypervisor + IRQ vector latency)
- `irq to sched`: time to enter in an RTOS task, scheduled by the IRQ handler (`irq delay` + RTOS scheduler)

All measurements are done using a hardware timer (GPT on i.MX 8M or TPM on i.MX 9) and relative to the hardware IRQ event time, with sub-microsecond precision.

Since Harpoon 2.4, the timer sampling frequency has been increased to better reflect real-time constraints: The hardware timer is now scheduled every **100 us**.

When running, the `rt_latency` application prints regular statistics, based on the measurements taken, to help characterize the system real-time latency.

The `rt_latency` application is available in the Harpoon share directory of the root file system:

```
/usr/share/harpoon/inmates/freertos/rt_latency.bin  # FreeRTOS binary
/usr/share/harpoon/inmates/zephyr/rt_latency.bin    # Zephyr binary
```

To use the `rt_latency` application, Jailhouse must be started first. To start Jailhouse and the `rt_latency` application, create an appropriate Harpoon configuration file and run the Harpoon service with `systemd`. For instance:

```
# harpoon_set_configuration.sh freertos latency
```

```
# systemctl start harpoon
```

The Harpoon service uses the `/etc/harpoon/harpoon.conf` configuration file that contains the RTOS and the application to run. By default, the configuration file points to the FreeRTOS audio application. To run the `rt_latency` application, we have generated a corresponding configuration file. This step needs to be run only once.

Once the Harpoon service has been started, the following `rt_latency` trace is shown in the terminal emulator connected to the other serial port:

```
Harpoon vX.Y.Z
main_task: running
```

After booting, the `rt_latency` application waits for commands to be received. A list of available commands is shown using the command `harpoon_ctrl`:

```
# harpoon_ctrl -h
```

The usage for the `rt_latency` application is shown:

```
Latency options:
        -r <id>        run latency test case id
        -q             quiet testing (Do not dump stats regularly, but only once
 on test case stop)
        -s             stop running test case
```

Examples:

To stop the `rt_latency` application's current test case:

```
# harpoon_ctrl latency -s
```

To run a test case:

It is possible to engage some CPU load and/or IRQ load to measure their impact on the latency. To do so, different test cases (TC) can be executed, by specifying the test case ID with the "`-r`" option:

```
# harpoon_ctrl latency -r <TC_ID>
```

TC_ID:

- 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 test case)
- 6: extra CPU load + cache flush (instruction cache only for this release)

To execute test case 1:

```
# harpoon_ctrl latency -r 1
```

When running, latency statistics are printed every 10 seconds:

```
---
INFO: start_test_case       : Running test case 1:

INFO: benchmark_task        : running

INFO: stats_print           : stats(C0601B30) irq delay (ns) min 625 mean 792 max 3625
 rms^2 629985 stddev^2 1510 absmin 625 absmax 3625


INFO: hist_print            : n_slot 21 slot_size 1000

INFO: hist_print            : 99890 76 22 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INFO: stats_print           : stats(C0601F90) irq to sched (ns) min 2583 mean 2587 max
 8291 rms^2 6702537 stddev^2 6329 absmin 2583 absmax 8291


INFO: hist_print            : n_slot 21 slot_size 1000

INFO: hist_print            : 0 0 99673 233 68 24 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
INFO: print_stats           : late alarm scheduling: 0
```

Both the `irq delay` and the `irq to sched` statistics are shown:

- `min/mean/max`: minimum, average, or maximum latency value measured within the last period of time.
- `absmin/absmax`: minimum or maximum latency value measured since the beginning of the test.
- A histogram is also shown to give an idea of the repartition of the measured latency values.

Table 23. Real-time latencies measured on i.MX 943/FreeRTOS (in ns)

Description	i.MX 943 IRQ Latency (ns)				i.MX 943 Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	541	791	6,416	657	1,958	2,003	8,000	2,709
Low priority task CPU load	833	901	1,583	1,583	2,166	2,421	4,041	4,946
Low priority IRQ load	833	1,079	1,708	7,298	6,875	9,166	11,541	9,788
Low priority task CPU load, mutex	875	905	1,541	1,825	2,250	2,428	4,125	8,087
Linux cpu + memory load	500	792	1,541	210	1,916	1,988	3,791	2,782
RTOS cold cache	500	792	1,500	207	1,958	1,992	3,833	1,978

Table 24. Real-time latencies measured on i.MX 95 19x19/FreeRTOS (in ns)

Description	i.MX 95 19x19 IRQ Latency (ns)				i.MX 95 19x19 Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	500	791	1,750	156	1,958	2,002	5,208	2,435
Low priority task CPU load	500	791	1,666	136	1,916	1,976	4,666	2,889
Low priority IRQ load	541	909	1,666	907	6,875	9,161	12,791	10,098
Low priority task CPU load, mutex	500	791	1,708	184	1,958	2,002	5,083	2,300
Linux cpu + memory load	500	791	1,666	159	1,916	1,975	4,291	2,693
RTOS cold cache	500	791	6,500	184	1,916	1,975	10,125	2,865

Table 25. Real-time latencies measured on i.MX 95 15x15/FreeRTOS (in ns)

Description	i.MX 95 15x15 IRQ Latency (ns)				i.MX 95 15x15 Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	541	791	2,125	187	1,958	2,003	5,875	1,933
Low priority task CPU load	500	792	2,375	187	1,916	1,992	6,125	1,795
Low priority IRQ load	750	911	1,666	338	6,791	9,196	13,250	10,758
Low priority task CPU load, mutex	541	791	4,500	297	1,958	2,004	7,208	2,939
Linux cpu + memory load	500	792	2,208	176	1,916	1,992	5,708	2,024
RTOS cold cache	500	792	1,958	186	1,916	1,988	10,625	2,876

Table 26. Real-time latencies measured on i.MX 93/FreeRTOS (in ns)

Description	i.MX 93 IRQ Latency (ns)				i.MX 93 Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	541	751	2,916	66	2,000	2,055	6,000	4,861
Low priority task CPU load	500	757	2,583	266	1,958	2,043	5,083	2,625
Low priority IRQ load	500	781	1,416	2,251	6,625	9,568	16,375	8,833
Low priority task CPU load, mutex	541	750	2,666	54	2,000	2,055	4,416	4,889
Linux cpu + memory load	541	757	2,666	262	1,958	2,044	5,166	2,737
RTOS cold cache	541	758	2,583	308	1,958	2,044	4,291	2,862

Table 27. Real-time latencies measured on i.MX 8M Plus/FreeRTOS (in ns)

Test description	i.MX 8M Plus IRQ Latency (ns)				i.MX 8M Plus Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	583	875	6,583	1,054	2,708	2,755	8,541	8,516
Low priority task CPU load	583	872	6,000	1,234	2,708	2,751	9,625	8,529
Low priority IRQ load	666	982	1,875	2,349	10,458	11,684	20,416	9,865
Low priority task CPU load, mutex	583	875	3,041	480	2,708	2,754	8,333	4,354
Linux cpu + memory load	583	871	4,458	668	2,708	2,751	6,333	5,224
RTOS cold cache	583	872	3,416	1,383	2,666	2,751	8,458	9,993

Table 28. Real-time latencies measured on i.MX 943/Zephyr (in ns)

Description	i.MX 943 IRQ Latency (ns)				i.MX 943 Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	875	898	1,583	727	1,916	2,352	3,791	7,989
Low priority task CPU load	875	895	1,625	650	1,958	2,584	4,916	26,984
Low priority IRQ load	666	900	1,625	874	7,500	8,697	10,083	6,488
Low priority task CPU load, mutex	875	899	1,458	721	2,041	2,353	3,750	8,611
Linux cpu + memory load	875	891	1,500	634	2,083	2,580	4,125	28,228
RTOS cold cache	875	895	2,000	706	1,958	2,509	4,166	40,777

Table 29. Real-time latencies measured on i.MX 95 19x19/Zephyr (in ns)

Description	i.MX 95 19x19 IRQ Latency (ns)				i.MX 95 19x19 Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	541	791	1,583	156	1,750	1,956	3,958	5,337
Low priority task CPU load	500	791	1,541	151	1,750	2,114	4,166	16,995
Low priority IRQ load	625	827	1,708	806	7,458	8,889	10,375	5,740
Low priority task CPU load, mutex	541	791	1,666	149	1,750	1,956	4,000	5,348
Linux cpu + memory load	541	791	1,625	123	1,750	2,113	4,208	16,860
RTOS cold cache	541	792	2,041	192	1,750	2,059	4,291	25,359

Table 30. Real-time latencies measured on i.MX 95 15x15/Zephyr (in ns)

Description	i.MX 95 15x15 IRQ Latency (ns)				i.MX 95 15x15 Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	541	791	1,625	132	1,750	1,974	4,208	5,451
Low priority task CPU load	541	791	1,666	154	1,750	2,142	4,250	19,549
Low priority IRQ load	625	836	1,708	1,514	7,458	8,830	12,458	10,053
Low priority task CPU load, mutex	541	791	1,541	134	1,750	1,975	4,083	5,575
Linux cpu + memory load	541	791	1,541	148	1,750	2,142	4,125	19,576
RTOS cold cache	541	792	2,041	230	1,750	2,080	4,125	28,441

Table 31. Real-time latencies measured on i.MX 93/Zephyr (in ns)

Description	i.MX 93 IRQ Latency (ns)				i.MX 93 Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	541	750	1,458	69	1,708	1,935	3,291	4,356
Low priority task CPU load	541	750	1,541	62	1,750	2,079	3,250	15,769
Low priority IRQ load	541	858	1,416	905	7,875	9,210	10,541	1,108
Low priority task CPU load, mutex	541	750	1,666	57	1,750	1,935	3,125	4,370
Linux cpu + memory load	541	750	1,541	64	1,750	2,076	3,291	15,582
RTOS cold cache	541	750	2,083	133	1,750	2,018	3,666	23,215

Table 32. Real-time latencies measured on i.MX 8M Plus/Zephyr (in ns)

Description	i.MX 8M Plus IRQ Latency (ns)				i.MX 8M Plus Task Latency (ns)			
	Min	Average	Max	Stddev^2	Min	Average	Max	Stddev^2
No system load	708	890	6,125	436	2,750	2,867	8,166	2,242
Low priority task CPU load	708	885	1,750	374	2,750	3,013	4,666	9,954
Low priority IRQ load	625	1,150	1,791	841	9,625	11,516	15,666	8,186
Low priority task CPU load, mutex	708	890	3,791	439	2,750	2,866	5,791	2,230
Linux cpu + memory load	708	885	2,000	374	2,750	3,013	4,708	9,861
RTOS cold cache	708	885	6,208	885	2,750	2,971	10,583	15,879

4.7 Virtio Networking application

4.7.1 Features of the Virtio Networking application

The `virtio_net` application is available in the Harpoon share directory of the root file system:

```
/usr/share/harpoon/inmates/freertos/virtio_net.bin  # FreeRTOS binary
```

Note: *The virtio_net application is only supported under FreeRTOS on i.MX 8M Mini EVK, i.MX 8M Plus EVK or i.MX 93 EVK for Yocto Real-time Edge SW (i.MX BSP Yocto not supported).*

This application starts a Virtio networking back end on Jailhouse inmate cell. Linux OS runs Virtio networking front end, which provides a virtual network interface. The back end owns physical ENET port and shares with the front end by using Virtio communication between the front end and back end.

4.7.2 Running the Virtio Networking application

To use the `virtio_net` application, Jailhouse must be started first. To start Jailhouse and the Virtio Networking application, create the corresponding Harpoon configuration file and run the Harpoon service using `systemd`, for example:

```
# harpoon_set_configuration.sh freertos virtio_net
```

Note: *Avoid changing the configuration while the Harpoon service is running (silent failure when restarting the service).*

The configuration file is stored under `/etc/harpoon/harpoon.conf` and the Harpoon systemd service uses it to start Jailhouse and the Virtio Networking application:

```
# systemctl start harpoon
```

When the Harpoon service has been started, `virtio_net` back end application is started with the following login console of inmate cell:

```
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-duplexx
```

Then in Linux console of root cell, use `ifconfig` and `ethtool` to check whether virtual networking interface is available. The driver used by virtual networking interface is "`virtio_net`", so from the following log, "`eth1`" is Virtio virtual networking interface.

```
root@imx8mm-lpddr4-evk:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 16384
        ether fa:6f:22:ce:31:6b  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

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.193.20.30  netmask 255.255.255.0  broadcast 10.193.20.255
        inet6 fe80::201:2ff:fe03:405  prefixlen 64  scopeid 0x20<link>
        ether 00:04:9f:00:01:02  txqueuelen 1000  (Ethernet)
```

```
        RX packets 17  bytes 3897 (3.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 41  bytes 7309 (7.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 99  bytes 8926 (8.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 99  bytes 8926 (8.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@imx8mm-lpddr4-evk:~# ethtool -i eth1
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
```

If the interface is connected to a DHCP service, it gets the IP address by DHCP. Otherwise, set the IP address by using the `ifconfig` command.

Then use the `ping` command to check whether the virtual networking interface works or not.

```
root@imx8mm-lpddr4-evk:~# ping 10.193.20.18
PING 10.193.20.18 (10.193.20.18) 56(84) bytes of data.
64 bytes from 10.193.20.18: icmp_seq=1 ttl=64 time=3.65 ms
64 bytes from 10.193.20.18: icmp_seq=2 ttl=64 time=1.83 ms
64 bytes from 10.193.20.18: icmp_seq=3 ttl=64 time=1.84 ms
64 bytes from 10.193.20.18: icmp_seq=4 ttl=64 time=1.83 ms
64 bytes from 10.193.20.18: icmp_seq=5 ttl=64 time=1.84 ms
64 bytes from 10.193.20.18: icmp_seq=6 ttl=64 time=1.84 ms
```

Use the following command to change the MAC address of `virtio_net`:

```
root@imx8mm-lpddr4-evk:~# ifconfig eth1 hw ether 00:04:9f:00:01:03
```

5 Known Issues

Table 33. Known issues

ID	Description	Workarounds
HRPN-245	Linux cannot access eMMC.	Store the root file system on SD card or NFS.
HRPN-448	RTOS crashes on Ethernet TSN use case stress restarts.	Restart the Jailhouse cell.
HRPN-483	Audio glitches on all boards for a combination of high frequency and low frame size.	Do not use combinations of the following parameters: • Frame size: 2, 4 • Frequency: 176.4 kHz, 192 kHz

Table 33. Known issues...*continued*

ID	Description	Workarounds
HRPN-632	Occurrences of command timeout for frame size 2 for Audio SMP pipeline.	-
HRPN-872/873	When running Audio SMP, you may run into instabilities: For a combination of high frequency and low frame size, you might have audio sample drop.	-
HRPN-1190	Spurious latency peak when stopping the RT Latency application using the `harpoon_ctrl` tool.	Disregard the last statistics print.
HRPN-1226	i.MX 8M Mini EVK Rev. E (Baseboards with RTL Ethernet PHYs) has PHY setup failures.	Currently, only boards with AR8031 PHYs are supported.
HRPN-1309	Occasional crash of the V4L2 driver on the Linux OS when starting Jailhouse on i.MX 8M Plus EVK.	None. No impact on the RTOS application and Linux console.

6 Technical Details on Harpoon Applications

6.1 Description

Harpoon reference applications are embedded in a repository named *harpoon-apps*.

Several RTOS applications are embedded in this repository, which may run in Jailhouse cells, based on an RTOS (currently using FreeRTOS and Zephyr) and leveraging the MCUXpresso SDK. As a consequence, *FreeRTOS-Kernel*, *mcu-sdk-cmsis*, *mcuxsdk-core*, *mcux-component*, and *mcux-devices-imx* repositories are required to build FreeRTOS-based applications, and *nxp-zephyr* and *hal_nxp* repositories are required to build Zephyr-based applications. Additionally, repositories *GenAVB_TSN*, *rtos-abstraction-layer*, *rtos-apps* (containing common RTOS application files shared with other teams), and *rpmsg-lite* (RPMsg Lite middleware) are needed to build the industrial and audio applications. The `west` tool (from the Zephyr project) is used to fetch those repositories, along with the `harpoon-apps` Git tree. The *mcuxsdk-manifests* repository provides a west-based manifest to fetch all required repositories efficiently.

To manage Linux-RTOS communication, a control application running in the Linux root cell is used. This application is to be compiled with the Yocto toolchain.

The next section explains how to build binaries (RTOS application and Linux control application).

Related information

https://docs.zephyrproject.org/latest/guides/west/index.html

6.2 Manual build

6.2.1 Setting up the environment

Both `git` and `west` should be installed to fetch the source code for Harpoon-apps, FreeRTOS, Zephyr, MCUXpresso SDK, etc.:

```
$ west init -m https://github.com/NXP/harpoon-apps --mr harpoon_3.5.0 hww
$ cd hww
$ west config commands.allow_extensions true # Allow the usage of west
 extensions provided by MCUXpresso SDK
$ west update
```

6.2.2 Building the RTOS application for the RTOS cell

6.2.2.1 Building FreeRTOS based applications

FreeRTOS applications for Armv8-A must be compiled with a compatible toolchain.

The reference toolchain is the GNU Arm cross-toolchain for the A-profile cores GCC 14.2.Rel1.

To download the toolchain and install it:

```
$ 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 -C /opt/ -xvf arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-elf.tar.xz
```

If starting from a fresh console, the cross-compiler variable must be set:

```
$ export ARMGCC_DIR=/opt/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-elf
```

Then, build an RTOS application:

```
$ cd harpoon-apps/<RTOS_APP>/freertos/boards/<BOARD>/armgcc_aarch64
$ ./build_ddr_release.sh
```

Where:

- `RTOS_APP` is `hello_world`, `audio`, `industrial`, `rt_latency`, or `virtio_net`.
- `BOARD` is `evkmimx8mm` for i.MX 8M Mini, `evkmimx8mn` for i.MX 8M Nano, `evkmimx8mp` for i.MX 8M Plus, `mcimx93evk` for i.MX 93 EVK, `imx943evk` for i.MX 943 EVK, `imx95lp4xevk15` for i.MX 95 15x15 EVK, and `imx95lpd5evk19` for i.MX 95 19x19 EVK.
- Build artefacts are available in the directory `ddr_release/`.
- The artefact to be used on target is the RTOS application binary: <RTOS_APP>.bin.

6.2.2.2 Building Zephyr-based applications

Install the cross-compile toolchain first, and then set the cross-compile environment and the Zephyr kernel directory variable:

```
$ export ARMGCC_DIR=/opt/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-elf
$ export Zephyr_DIR=/path/to/hww/zsdk/zephyr
```

Then, build a Single Core Zephyr application:

```
$ cd harpoon-apps/<RTOS_APP>/zephyr/boards/<BOARD>/armgcc_aarch64
$ ./build_singlecore.sh
```

Or build an SMP Zephyr application:

```
$ cd harpoon-apps/<RTOS_APP>/zephyr/boards/<BOARD>/armgcc_aarch64
$ ./build_smp.sh
```

Where,

- `RTOS_APP` is `hello_world`, `audio`, `industrial`, or `rt_latency`.
- BOARD is `evkmimx8mm` for i.MX 8M Mini, `evkmimx8mn` for i.MX 8M Nano, `evkmimx8mp` for i.MX 8M Plus, `mcimx93evk` for i.MX 93 EVK, `imx943evk` for i.MX 943 EVK, `imx95lp4xevk15` for i.MX 95 15x15 EVK, and `imx95lpd5evk19` for i.MX 95 19x19 EVK.

- Build artefacts are available in the directory `build_singlecore/zephyr/` or `build_smp/zephyr/`.
- The artefact to be used on target is the RTOS application binary: <RTOS_APP>.bin for singlecore application or <RTOS_APP>_smp.bin for the SMP application.

6.2.3 Building the Linux control application for the root cell

The Linux control application for Armv8-A must be compiled with a compatible toolchain.

The reference toolchain is the Poky Arm cross-toolchain built with Yocto.

To generate this toolchain for a specific Yocto image:

```
$ bitbake <yocto_image_name> -c populate_sdk
```

This generates a toolchain installer in the directory `tmp/deploy/sdk`. The installer name depends on the `DISTRO` and `MACHINE` variables and on the image name of the current build. For instance, for an i.MX build of `imx-image-core`, the installer name is `fsl-imx-xwayland-glibc-x86_64-meta-toolchain-armv8a-imx8mm-lpddr4-evk-toolchain-6.12-walnascar.sh`.

When executed, the installer prompts for a directory where to put the toolchain. The default location for the i.MX toolchain is `/opt/fsl-imx-xwayland/6.12-walnascar`.

When the toolchain is installed, different cross-compile variables must be set. This is done by sourcing the script `environment-setup-armv8a-poky-linux`. For example, with the default installation path:

```
$ . /opt/fsl-imx-xwayland/6.12-walnascar/environment-setup-armv8a-poky-linux
```

The Harpoon control application can then be built:

```
$ cd harpoon-apps/ctrl
$ ./build_ctrl.sh
```

The build generates one binary: `harpoon_ctrl` in the same directory and can be used on target.

The Linux root cell uses the Remote Processor Messaging (RPMsg) device to communicate with FreeRTOS and Zephyr inmate cells. `harpoon_ctrl` binary implements this device, and should be used to communicate with RTOS cells.

6.3 Developing a Harpoon Application

Harpoon-apps is the basis to create a Harpoon application. It links with (at least) MCUXpresso drivers and an RTOS (FreeRTOS and Zephyr).

A Harpoon application has its own directory in the root folder of the Harpoon-apps repository. Examples include `audio`, the audio reference application, `industrial`, the industrial reference application and `rt_latency`, the real-time benchmark application.

6.3.1 Architecture of the audio application

The audio application, which serves as an example for this chapter, has the following architecture.

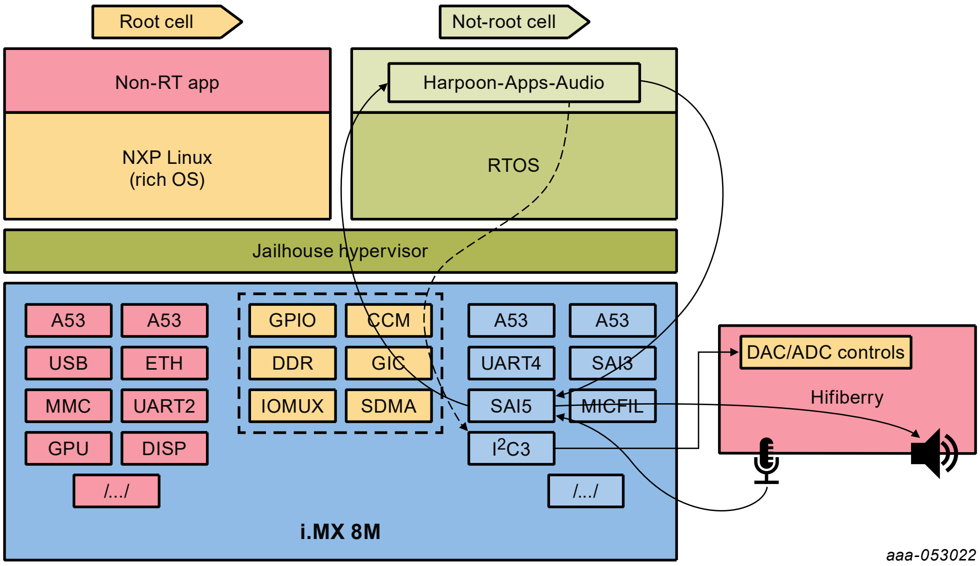


Figure 34. Architecture of audio application

The DAC and ADC on the HiFiBerry card are controlled by the audio application. Control is done through I2C3 and data throughput through SAI5.

6.3.2 Source file creation

This section provides some information on how to develop an application for Harpoon by using the `audio` application as an example.

First, the application directory must be created in the root directory of the repository `harpoon-apps`.

This directory contains the source code for the application, a CMake configuration file listing the files to be compiled. The source file can be common to all RTOS and platforms, RTOS-dependent and/or platform-dependent. Helper scripts are provided to build the application for each RTOS/platform combination.

```
audio
├── common
│   ├── audio_app.c
│   ├── audio_app.h
│   ├── avb_config.c
│   ├── boards
│   │   ├── evkmimx8mm
│   │   │   ├── app_board.h
│   │   │   ├── avb_hardware.c
│   │   │   ├── clock_config.c
│   │   │   ├── codec_config.c
│   │   │   ├── genavb_sdk.h
│   │   │   ├── pin_mux.c
│   │   │   ├── sai_clock_config.c
│   │   │   └── sai_config.c
│   │   ├── evkmimx8mn
│   │   │   [...]
│   │   ├── evkmimx8mp
│   │   │   [...]
│   │   ├── include
│   │   │   ├── avb_hardware.h
│   │   │   ├── clock_config.h
│   │   │   ├── codec_config.h
```

```
│    │    │    ├── pin_mux.h
│    │    │    └── sai_config.h
│    │    └── mcimx93evk
│    │    │    [...]
│    ├── pipeline_config.c
│    └── pipeline_config_single_sai.c
├── freertos
│    ├── boards
│    │    ├── evkmimx8mm
│    │    │    ├── app_mmu.h
│    │    │    ├── armgcc_aarch64
│    │    │    │    ├── build_ddr_debug.sh
│    │    │    │    ├── build_ddr_release.sh
│    │    │    │    └── clean.sh
│    │    │    ├── prj.conf
│    │    │    ├── reconfig.cmake
│    │    │    └── variable.cmake
│    │    ├── evkmimx8mn
│    │    │    ├── app_mmu.h
│    │    │    ├── armgcc_aarch64
│    │    │    │    ├── build_ddr_debug.sh
│    │    │    │    ├── build_ddr_release.sh
│    │    │    │    └── clean.sh
│    │    │    ├── prj.conf
│    │    │    ├── reconfig.cmake
│    │    │    └── variable.cmake
│    │    ├── evkmimx8mp
│    │    │    ├── app_mmu.h
│    │    │    ├── armgcc_aarch64
│    │    │    │    ├── build_ddr_debug.sh
│    │    │    │    ├── build_ddr_release.sh
│    │    │    │    └── clean.sh
│    │    │    ├── prj.conf
│    │    │    ├── reconfig.cmake
│    │    │    └── variable.cmake
│    │    └── mcimx93evk
│    │         ├── app_mmu.h
│    │         ├── armgcc_aarch64
│    │         │    ├── build_ddr_debug.sh
│    │         │    ├── build_ddr_release.sh
│    │         │    └── clean.sh
│    │         ├── prj.conf
│    │         ├── reconfig.cmake
│    │         └── variable.cmake
│    ├── CMakeLists.txt
│    ├── genavb_sdk.cmake
│    ├── Kconfig
│    ├── main.c
│    └── prj.conf
└── zephyr
     ├── boards
     │    ├── evkmimx8mm
     │    │    ├── app_mmu.c
     │    │    ├── app_mmu.h
     │    │    └── armgcc_aarch64
     │    │         ├── build_singlecore.sh
     │    │         ├── build_smp.sh
     │    │         └── clean.sh
     │    ├── evkmimx8mn
     │    │    [...]
```

```
    │     ├── evkmimx8mp
    │     │   [...]
    │     ├── imx8mm_evk_mimx8mm6_a53.conf
    │     ├── imx8mm_evk_mimx8mm6_a53.overlay
    │     ├── imx8mm_evk_mimx8mm6_a53_smp.conf
    │     ├── imx8mm_evk_mimx8mm6_a53_smp.overlay
    │     ├── imx8mn_evk_mimx8mn6_a53.conf
    │     ├── imx8mn_evk_mimx8mn6_a53.overlay
    │     ├── imx8mn_evk_mimx8mn6_a53_smp.conf
    │     ├── imx8mn_evk_mimx8mn6_a53_smp.overlay
    │     ├── imx8mp_evk_mimx8ml8_a53.conf
    │     ├── imx8mp_evk_mimx8ml8_a53.overlay
    │     ├── imx8mp_evk_mimx8ml8_a53_smp.conf
    │     ├── imx8mp_evk_mimx8ml8_a53_smp.overlay
    │     ├── imx93_evk_mimx9352_a55.conf
    │     ├── imx93_evk_mimx9352_a55.overlay
    │     └── mcimx93evk
    │     │   [...]
    ├── CMakeLists.txt
    ├── genavb_sdk.cmake
    ├── main.c
    └── prj.conf
```

The application starts in the function `main()`, and defined in the file `main.c`.

RTOS-specific code goes to the directories `audio/freertos` and `audio/zephyr`.

Board specific code (clock configuration, hardware description, MMU configuration) goes to the directories `audio/<rtos>/boards/<boardid>` and `audio/boards/<boardid>`.

OS-agnostic code goes to the directory `audio/common`.

6.3.3 Board specific code

Board specific code and header files for the audio application include:

Table 34. Board specific code

`app_board.h`	Definition of SAI and I2C instances used for the demo. I2C addresses of HiFiBerry's DAC and ADC. SAI configuration. Audio samples format.
`app_mmu.h`	Device memory to map with MMU (includes SAI and I2C).
`sai_clock_config.c`	Configuration of Audio PLLs, Audiomix (for i.MX 8M Plus) and SAI clocks.
`sai_config.c`	Define configuration of each SAI instance.
`codec_config.c`	Helper functions to open, configure and close DAC and ADC drivers.
`pin_mux.c`	Functions to set IOMux for the application use case.
`CMakeLists.txt`	CMake configuration file that includes all necessary MCUXpresso drivers.

6.3.4 Controlling application from Linux side

Linux side can control the Harpoon application by sending messages through the RPMsg communication channel.

The audio application leverages this in function `audio_app_ctrl_init()`, defined in `audio/common/audio_app.c`.

For RPMsg channel, RTOS creates a RPMsg endpoint with service name "rpmsg-raw" for communication:

```
void *ctrl_handle = (void *)rpmsg_transport_init(RL_BOARD_RPMSG_LINK_ID,
 EPT_ADDR, "rpmsg-raw");
```

Finally, the application's main thread periodically looks for incoming control messages:

```
do {
        audio_command_handler(&ctx);
        […]
} while (1);
```

The Linux user space application that sends control messages is located in the directory `ctrl` of the `harpoon-apps` repository.

7 Note About the Source Code in the Document

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

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision History

The following table provides the revision history for this document.

Table 35. Revision history

Document ID	Release date	Description
UG10170 v.3.5	12 December 2025	• Support for Hello World and RT Latency on i.MX 943 EVK • Support for Software-based AVB Media Clock Recovery for i.MX 8M Mini, i.MX 8M Nano, and i.MX 93 EVKs • Migrated common RTOS application components into rtos-apps as a library • Migrated to major MCUX SDK release from 2.16.0 to 25.09.00 • Updated Zephyr from v4.1.0 to v4.3.0 • Updated FreeRTOS Kernel from 11.0.1 to 11.1.0

Table 35. Revision history*...continued*

Document ID	Release date	Description
		• Upgraded the Linux Kernel from 6.12.20 to 6.12.34 • Updated to new Arm GCC 14.2 Rel1 Toolchain
UG10170 v.3.4	28 July 2025	• Support for Industrial TSN on i.MX 95 15x15 LPDDR4x EVK and i.MX 95 19x19 LPDDR5 EVK • Updated Zephyr from v3.7.0 to v4.1.0 • Upgraded the Linux Kernel from 6.6.36 to 6.12.20
UG10170 v.3.4	26 March 2025	• Support for Hello World and RT Latency on i.MX 95 15x15 LPDDR4x EVK • Support for Industrial FlexCAN on i.MX 95 15x15 LPDDR4x EVK and i.MX 95 19x19 LPDDR5 EVK • Support Interrupt Nesting on FreeRTOS
UG10170 v.3.2	18 December 2024	• Support for Hello World and RT Latency on i.MX 95 EVK • Improved performance on RT Latency for i.MX 8M device family • Updated MCUX SDK from 2.14.0 to 2.16.0 • Updated FreeRTOS Kernel from 10.5.0 to 11.0.1 • Updated Zephyr from v3.5.0 to v3.7.0 • Updated to new Arm GCC 13.2 Rel1 Toolchain • Support for TSN Motor Controller mode in industrial application on i.MX 8M Plus EVK • Bug Fix: FPU Context save at Context Switch on FreeRTOS that caused crashes on new ARM GCC Toolchain
HRPNUG_3.1	26 July 2024	• Support for AVB Milan Mode • Support for multi-channel audio usecase on MX93AUD-HAT • Support for TSN Motor Controller mode in industrial application on i.MX 93 EVK
HRPNUG v.3.0	29 March 2024	• Support for AVB SMP pipeline in Zephyr audio application • Support for MX93AUD-HAT audio expansion board on i.MX 93 EVK • Improve support for AVB Talker and Listener on Zephyr • Improve support for TSN industrial application on Zephyr
HRPNUG v.2.5	15 December 2023	• Support for audio and industrial applications on i.MX 93 EVK • Initial support for AVB Talker and Listener on Zephyr • Initial support for TSN industrial application on Zephyr
HRPNUG v.2.4	28 July 2023	• Full Support for RPMsg control (all OSes, all boards) • Support for RT Latency on i.MX 93 EVK • Support for Virtual Ethernet on i.MX 8M Plus and i.MX 93 EVK • Support for AVB Listener Media Clock Recovery on i.MX 8M Plus EVK • Support for AVB Listener Synchronization
HRPNUG v.2.3	28 March 2023	• Support for AVB Talker in FreeRTOS audio • Support for RPMsg control (FreeRTOS, all boards) • Support for Virtual Ethernet • Support for i.MX 93 (preview: hello_world)
HRPNUG v.2.2	16 December 2022	• Support for AVB listener in FreeRTOS audio • Support for SMP pipeline in Zephyr audio • Support for RPMsg control (preview) • Support for ENET, ENET_QoS in Zephyr industrial
HRPNUG v.EAR 2.1.0	28 July 2022	Minor changes to Section 4 and Section 5. Compatible with Real-Time Edge Software Rev 2.3 release

Table 35. Revision history*...continued*

Document ID	Release date	Description
HRPNUG v.EAR 2.1.0	30 June 2022	• New industrial application in harpoon-apps • Implementation of flexible audio pipeline in harpoon-apps • Support for i.MX 8M Nano EVK for i.MX Yocto • Support for EVK's internal audio codecs • Support for systemd • Support for Zephyr • Drivers for FlexCAN, ENET, ENET_QOS
HRPNUG v.EAR 2.0.1	29 March 2022	Full integration to NXP Real-Time Edge
HRPNUG v.EAR 2.0.0	14 January 2022	Introduction of `harpoon-apps`. Support of FreeRTOS Support of both i.MX BSP and Real-Time Edge SW

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.

Amazon Web Services, AWS, the Powered by AWS logo, and FreeRTOS — are trademarks of Amazon.com, Inc. or its affiliates.

I2C-bus — logo is a trademark of NXP B.V.

Layerscape — is a trademark of NXP B.V.

QorIQ — is a trademark of NXP B.V.

Contents

1	**Overview**	**2**
1.1	Supported features for i.MX 8M Family	3
1.2	Supported features for i.MX 9 Family	4
1.3	Architecture	5
1.4	Hardware resource partitioning	5
1.5	i.MX 943 and i.MX 95 System Manager	7
1.6	Acronym Table	7
2	**Building Harpoon Yocto Images**	**8**
2.1	i.MX Yocto	8
2.2	Real-Time Edge Yocto	8
3	**Hardware Setup**	**9**
3.1	i.MX reference boards	9
3.1.1	i.MX 8M Mini EVK	9
3.1.2	i.MX 8M Mini EVKB	9
3.1.3	i.MX 8M Nano EVK	10
3.1.4	i.MX 8M Plus EVK	10
3.1.5	i.MX 93 EVK	11
3.1.6	i.MX 943 EVK	11
3.1.7	i.MX 95 15x15 LPDDR4x EVK and i.MX 95 19x19 LPDDR5 EVK	11
3.2	Audio use case hardware	11
3.2.1	HiFiBerry setup	12
3.2.2	MX93AUD-HAT setup	15
3.3	Industrial use case hardware	16
3.4	Virtio networking use case hardware	18
4	**Running Harpoon Reference Applications**	**18**
4.1	Basic setup	18
4.2	Starting Linux kernel	18
4.3	hello_world application	21
4.4	Audio application	22
4.4.1	Features of the audio application	22
4.4.2	Starting the audio application	23
4.4.3	Audio latency in loopback mode	24
4.4.4	Running audio application: examples	25
4.4.4.1	Playing DTMF	25
4.4.4.2	Playing in loopback mode	25
4.4.4.3	Playing a full audio pipeline	27
4.4.4.4	Playing an AVB audio pipeline	30
4.4.4.5	Playing an SMP audio pipeline	55
4.5	Industrial application	57
4.5.1	Features of the industrial application	58
4.5.2	Starting the industrial application	58
4.5.3	Running the industrial application: examples	59
4.5.3.1	FlexCAN multiple nodes communication	59
4.5.3.2	Ethernet through MCUXpresso SDK API	62
4.5.3.3	Ethernet with GenAVB/TSN stack	63
4.5.3.4	Ethernet motor control with GenAVB/TSN stack	67
4.6	rt_latency application	71
4.7	Virtio Networking application	77
4.7.1	Features of the Virtio Networking application	77
4.7.2	Running the Virtio Networking application	77
5	**Known Issues**	**78**
6	**Technical Details on Harpoon Applications**	**79**
6.1	Description	79
6.2	Manual build	79
6.2.1	Setting up the environment	79
6.2.2	Building the RTOS application for the RTOS cell	80
6.2.2.1	Building FreeRTOS based applications	80
6.2.2.2	Building Zephyr-based applications	80
6.2.3	Building the Linux control application for the root cell	81
6.3	Developing a Harpoon Application	81
6.3.1	Architecture of the audio application	81
6.3.2	Source file creation	82
6.3.3	Board specific code	84
6.3.4	Controlling application from Linux side	84
7	**Note About the Source Code in the Document**	**85**
8	**Revision History**	**85**
	Legal information	**88**

Date of release: 12 December 2025
Document identifier: UG10170