HRPNUG

Harpoon User's Guide

Rev. 2.3 — 28 March 2023 User guide

Document information

Information	Content
Keywords	i.MX 8M, i.MX 93, 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 2.3 for i.MX 8M and i.MX 93 device family, using the Arm Cortex-A53/A55 processor (Armv8-A architecture).

1 Overview

This document presents the Harpoon release 2.3 for i.MX 8M and i.MX 93 device family, 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. This release includes an audio application, an industrial application and a real-time latency measurement application, all available both for FreeRTOS as well as Zephyr (some application feature limitations exist depending on the selected platform and RTOS).

This release supports the following software and hardware:

- NXP Linux Yocto
  - i.MX LF 6.1.1-1.0.0: For more information, see the i.MX Yocto Project User's Guide.
  - Real-time Edge SW v2.5: For more information, see the Real-time Edge Yocto Project v2.5 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 93 EVK (EAR support)
- Jailhouse hypervisor
- FreeRTOS V10.4.3 kernel
  - AARCH64 port, uniprocessor
  - Guest OS running on Jailhouse cell
- Zephyr RTOS 3.2.0
  - Cortex-A53 and Cortex-A55 port, SMP
  - Guest OS running on Jailhouse cell
- MCUXpresso SDK 2.12
  - GIC, Timer and MMU AARCH64 drivers
  - FlexCAN, ENET, ENET_QOS, GPT, I2C, SAI, and UART SoC drivers
  - Audio Codec drivers
  - PHY drivers
- RTOS applications
  - Audio reference application
  - Industrial reference application
  - Real-time latency measurement application

1.1 Architecture

The following figure shows the architecture of the Harpoon solution.

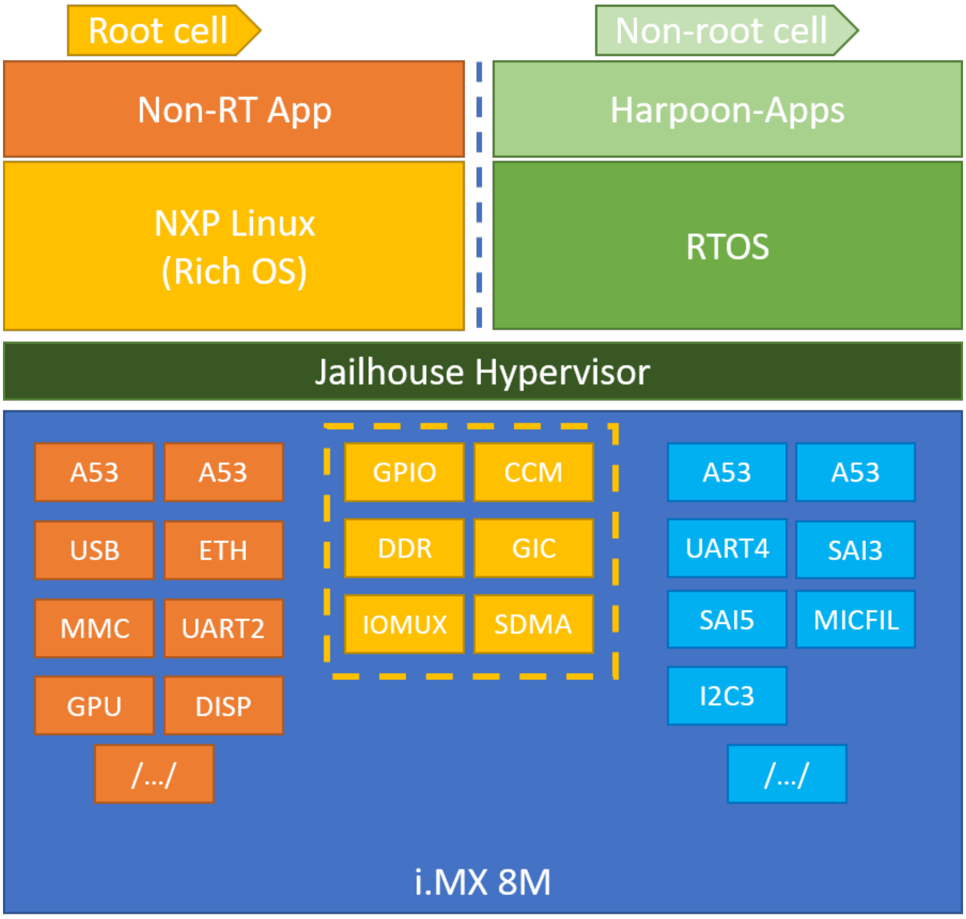


Figure 1. Harpoon solution architecture

The i.MX 8M box shows the hardware partitioning between Jailhouse cells.

The boxes in dark orange (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 the real-time application running on Jailhouse's inmate cell. It is built on top of Zephyr or FreeRTOS, using MCUXpresso drivers.

1.2 Hardware resource partitioning

Jailhouse hypervisor is used to run an RTOS in parallel with Linux: FreeRTOS and Zephyr are supported in this release.

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 RTOS (inmate) 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 cores 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 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/imx8m*-freertos.c` for the cell configuration of the FreeRTOS `hello_world` and `rt_latency` use case
- `configs/arm64/imx8m*-zephyr.c` for the cell configuration of the Zephyr `hello_world` and `rt_latency` use case
- `configs/arm64/imx8m*-freertos-audio.c` for the cell configuration of the FreeRTOS `audio` use case
- `configs/arm64/imx8m*-freertos-avb.c` for the cell configuration of the FreeRTOS `audio (AVB)` use case
- `configs/arm64/imx8m*-zephyr-audio.c` for the cell configuration of the Zephyr `audio` use case
- `configs/arm64/imx8m*-freertos-industrial.c` for the cell configuration of the FreeRTOS `industrial` use case
- `configs/arm64/imx8m*-zephyr-industrial.c` for the cell configuration of the Zephyr `industrial` use case
- `configs/arm64/imx93-freertos.c` for the cell configuration of the FreeRTOS `hello_world` use case
- `configs/arm64/imx93-zephyr.c` for the cell configuration of the Zephyr `hello_world` use case
- `configs/arm64/imx8m*.c` and `configs/arm64/imx93.c` for the root cell configuration

The CPU core allocated to the RTOS forms a bitmap in the `cpu` structure. Here, CPU core 3 is assigned to the cell:

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

For a multicore (SMP) cell, two cores can be used, for instance:

```
.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.

2 Building Harpoon Yocto images

As mentioned in the overview section, Harpoon is compatible with both i.MX Yocto and Real-Time Edge Yocto. Each distribution is addressed in a separate section below.

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-langdale -m
 imx-6.1.1-1.0.0_harpoon-v2.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

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 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 release.

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

This Harpoon release 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 https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/evaluation-kit-for-the-i-mx-8m-mini-applications-processor:8MMINILPD4-EVK*

3.1.2 i.MX 8M Nano EVK

Figure 3. i.MX 8M Nano EVK

Note: *For more information to order the board, see https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/evaluation-kit-for-the-i-mx-8m-nano-applications-processor:8MNANOD4-EVK.*

3.1.3 i.MX 8M Plus EVK

Figure 4. i.MX 8M Plus EVK

Note: *For more information to order the board, see https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-8m-plus-evaluation-kit-enabling-power-measurement:8MPLUSLPD4-PEVK.*

3.1.4 i.MX 93 EVK

Note: *i.MX 93 family pre-production samples and EVKs along with technical support available for early access customers through a paid program. For more info, see https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-9-processors/i-mx-93-applications-processor-family-arm-cortex-a55-ml-acceleration-power-efficient-mpu:i.MX93*

3.2 Audio use case hardware

Harpoon audio application uses the I2S HiFiBerry audio card *DAC+ ADC Pro*.

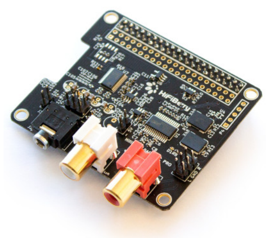


Figure 5. HiFiBerry DAC+ ADC Pro (picture from HiFiBerry's website)

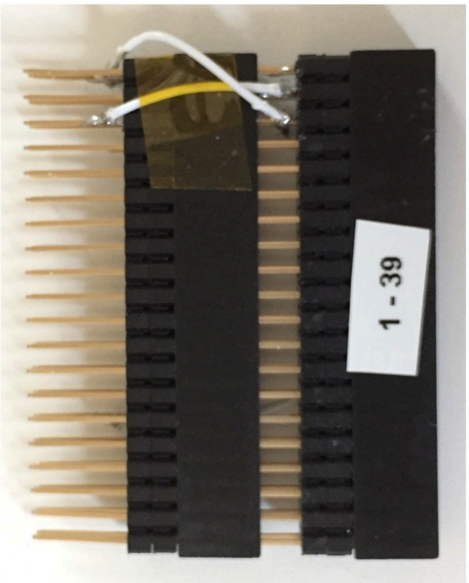
Note: *For more information to order the board, see https://www.hifiberry.com/shop/boards/hifiberry-dac-adc-pro/.*

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.

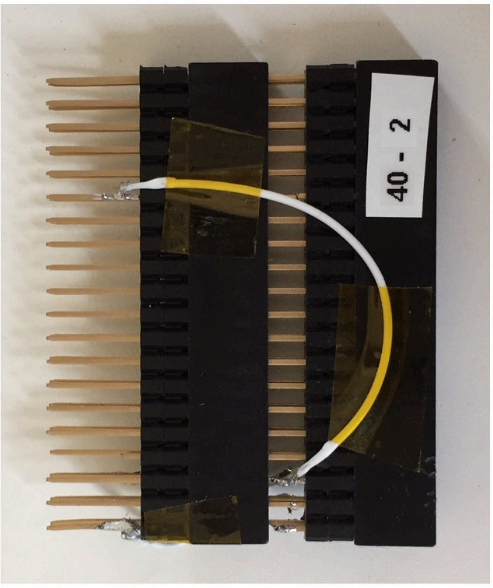
The following pins on the EVK's 40-pin connector must be connected to the following HiFiBerry's pins.

Table 1. 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
37	**35**	I2S word select for RX and TX
38	**38**	I2S RX



Inward



Outward

Figure 6. Handmade transposer

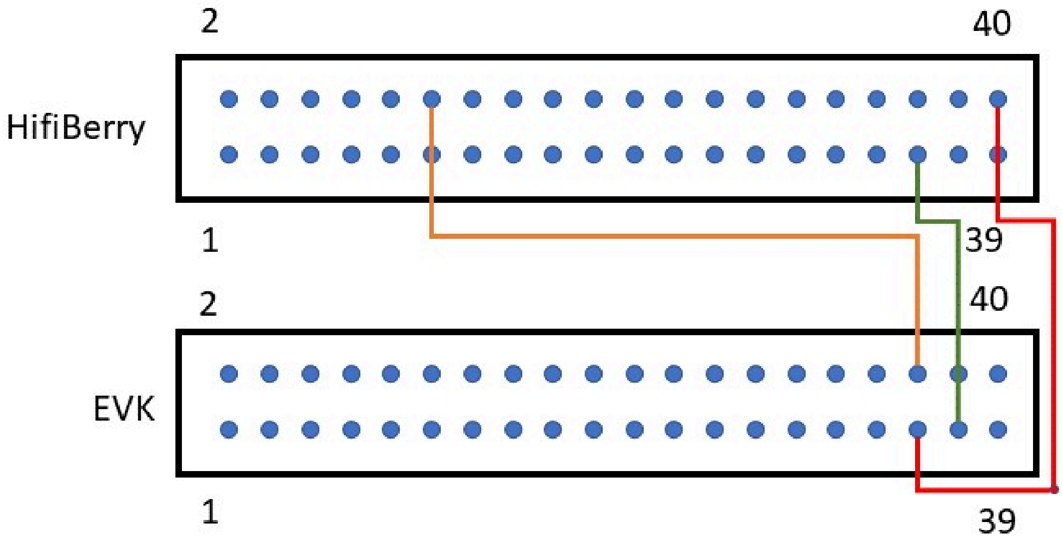


Figure 7. Transposer (Top View)

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

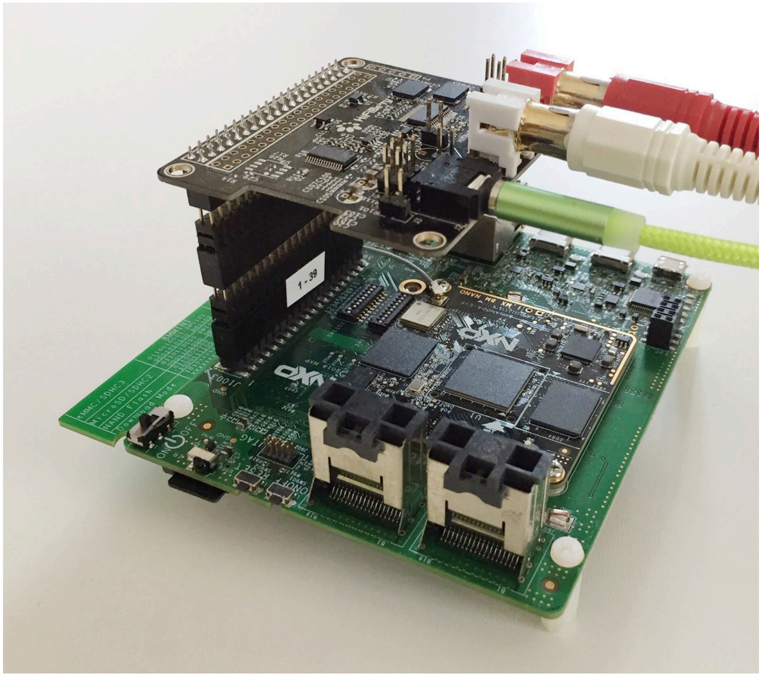


Figure 8. i.MX 8M Mini EVK with HiFiBerry audio card

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

Record 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**).

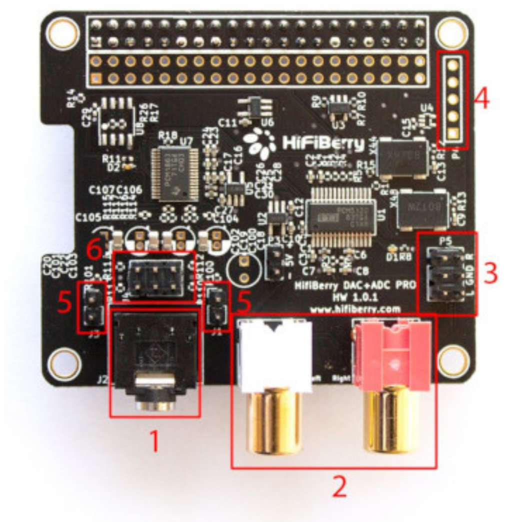


Figure 9. HiFiBerry audio connectors (picture from HiFiBerry's website)

Note: *For more information to order the board, see https://www.hifiberry.com/shop/boards/hifiberry-dac-adc-pro/.*

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.

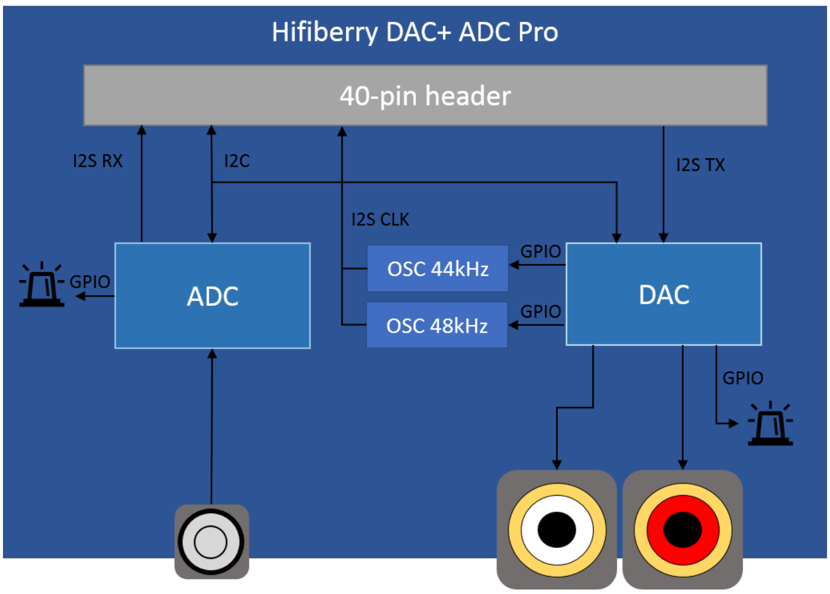


Figure 10. HiFiBerry architecture

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

3.3 Industrial use case hardware

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

Figure 11. LS1028A AVB/TSN network bridge

Note: *For more information to order the board, see https://www.nxp.com/design/qoriq-developer-resources/layerscape-ls1028a-reference-design-board:LS1028ARDB.*

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.

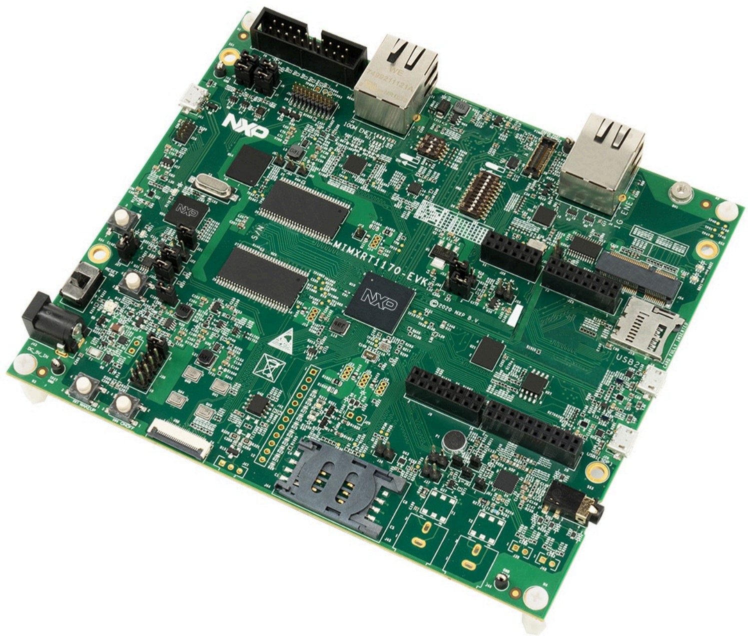


Figure 12. RT1170 TSN endpoint

Note: *For more information to order the board, see https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-rt1170-evaluation-kit:MIMXRT1170-EVK.*

The RT1170 is used as a TSN endpoint in a TSN network, exchanging packets with the i.MX 8M Plus board.

3.4 Virtio networking use case hardware

User needs to connect ENET port on i.MX 8M Mini 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

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

To do this, when U-Boot is executing, stop at 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 (hello_world, audio or rt_latency):

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

- For i.MX 8M Mini (hello_world or audio AVB):

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

- For i.MX 8M Mini (hello_world, industrial or rt_latency):

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

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

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

- For i.MX 8M Nano (hello_world, audio AVB):

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

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

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

- For i.MX 8M Plus (hello_world or audio AVB):

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

- For i.MX 8M Plus (hello_world, industrial or rt_latency):

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

- For i.MX 93 (hello_world):

```
u-boot => setenv jh_root_dtb imx93-11x11-evk-harpoon.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 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 FreeRTOS binary, create configuration:

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

To run Zephyr binary, create configuration:

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

Start 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 has been started, the following logs is shown in the inmate cell console:

FreeRTOS logs:

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

Zephyr logs:

```
*** Booting Zephyr OS build zephyr-v3.2.0-25-g106de0397317  ***
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: record sound from HiFiBerry's input and play it live through HiFiBerry's output.
- 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 (FreeRTOS only): implements a 3-stage pipeline with AVB input as a source that can be routed to different sinks (SAI outputs, AVTP sink).
- SMP Audio pipeline (Zephyr only): splits the above pipeline in two pieces to process them onto different cores.

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 64bit float format
- Audio playback to both SAI3 (on board codec/sound jack) and SAI5 (HiFiBerry)
- Audio capture from SAI5 (HiFiBerry)

Note: *Playback on SAI3: The i.MX 8M Plus EVK on board codec (WM8960) supports sample rates up to 48 kHz only. 88.2 kHz and above frequency settings will fail for this codec.*

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, the following command can be run to generate an appropriate configuration file:

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

To use the audio application, Jailhouse must be started first. To start Jailhouse and the audio application, run the Harpoon service with systemd:

```
# systemctl start harpoon
```

Once the Harpoon service has been 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)
                       Supporting 44100, 48000, 88200, 176400, 96000, 192000 Hz
                       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
        -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
        -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 2. Audio application latency

Sampling rate (kHz)	Audio latency (µs)				
	Audio buffer size (frames)				
	32	**16**	**8**	**4**	**2**
192	590	430	340	295	272
176.4	635	453	363	340	317
96	1,180	862	680	612	567
88.2	1,315	930	748	658	612
48	2,358	1,701	1,360	1,202	1,111
44.1	2,585	1,860	1,497	1,315	1,224

4.4.4 Running audio application: examples

4.4.4.1 Playing DTMF

To start DTMF playback with default parameters (48000 Hz sampling rate):

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

The DTMF is played both to the HiFiBerry RCA outputs as well as the onboard jack.

To run another audio use case, the playback must be stopped with the following command:

```
# harpoon_ctrl audio -s
```

4.4.4.2 Playing in loopback mode

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

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

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

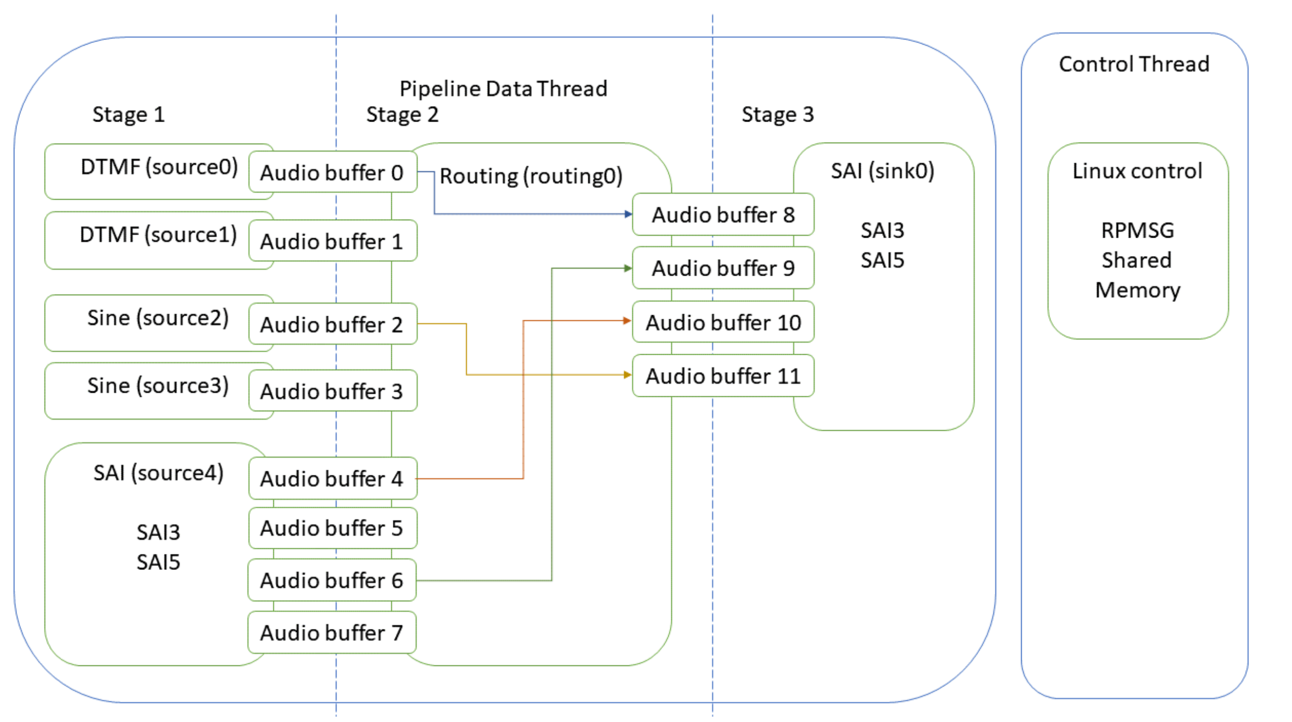
To run another audio use case, the playback must be stopped with the following command:

```
# harpoon_ctrl audio -s
```

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.



Audio pipeline with multiple sources/sinks and a routing element

Figure 13. Audio pipeline

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

Table 3. Indices of source elements

Index	Source element	Comment
0	DTMF, sequence 1	Software generated source
1	DTMF, sequence 2	Software generated source
2	Sine wave, 440 Hz	Software generated source
3	Sine wave, 880 Hz	Software generated source
4	SAI5, left channel	Hardware source
5	SAI5, right channel	Hardware source
6	SAI3, left channel	Hardware source
7	SAI3, right channel	Hardware source

Table 4. Indices of sink elements

Index	Sink element	Comment
0	SAI5, left channel	Hardware sink
1	SAI5, right channel	Hardware sink
2	SAI3, left channel	Hardware sink
3	SAI3, right channel	Hardware sink

This makes for a flexible pipeline. For instance, the following commands starts the pipeline and configures the routing element to have a loopback between SAI5 input and SAI3 output (i.e., sound recorded by the HiFiBerry card played by the EVK's internal codec) while a DTMF sequence is played on the left channel of SAI5's output and a 440 Hz sine wave on the right channel of SAI5's output (i.e., HiFiBerry's 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)
```

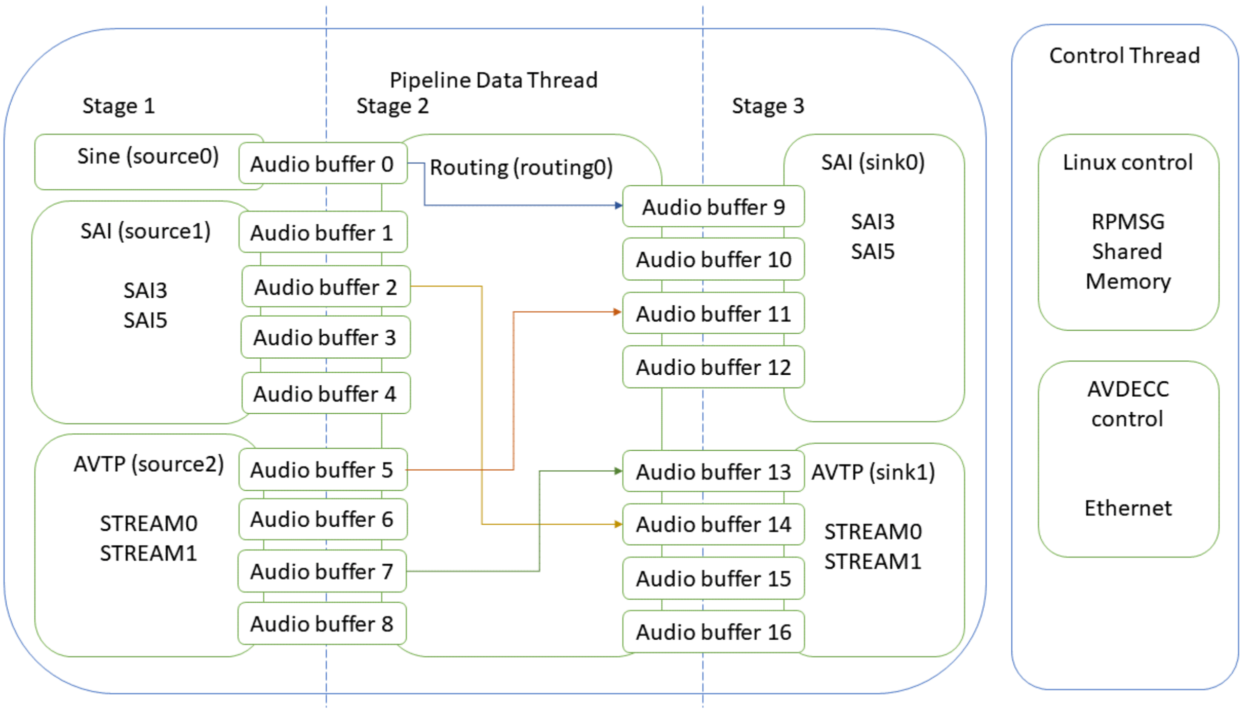
Note: *The pipeline dump also outputs the Audio Buffer Routing for an easier Buffer Routing through the "Routing element".*

4.4.4.4 Playing an AVB audio pipeline

The AVB audio pipeline embeds - as a source element - an AVB Listener, making use of the GenAVB/TSN stack streaming API’s. This element is only responsible of 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

It re-uses the audio application's pipeline framework for audio processing in which an AVTP Listener is added as a source.



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

Figure 14. 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 indices to connect elements.

Table 5. Indices of source elements

Index	Source element	Comment
0	Sine wave, 440 Hz	Software generated source
1	SAI5, left channel	Hardware source
2	SAI5, right channel	Hardware source
3	SAI3, left channel	Hardware source
4	SAI3, right channel	Hardware source
5	AVTP, stream#0 left channel	AVB source from network
6	AVTP, stream#0 right channel	AVB source from network
7	AVTP, stream#1 left channel	AVB source from network
8	AVTP, stream#1 right channel	AVB source from network

Table 6. Indices of sink elements

Index	Sink element	Comment
0	SAI5, left channel	Hardware sink
1	SAI5, right channel	Hardware sink
2	SAI3, left channel	Hardware sink
3	SAI3, right channel	Hardware sink

Table 6. Indices of sink elements...*continued*

Index	Sink element	Comment
4	AVTP, stream#0 left channel	AVB sink to network
5	AVTP, stream#0 right channel	AVB sink to network
6	AVTP, stream#1 left channel	AVB sink to network
7	AVTP, stream#1 right channel	AVB sink to network

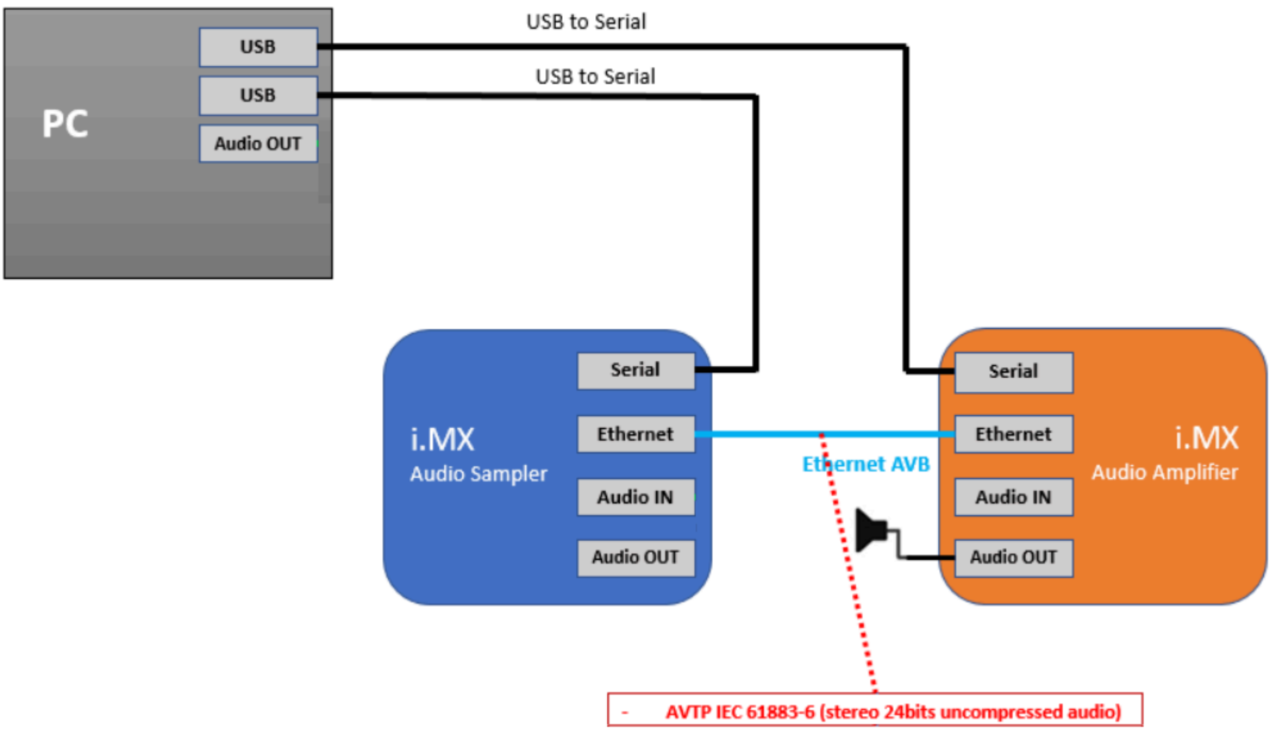
The sections below describe how to set up an (external) AVB Audio Media Server to enable the (Harpoon) AVB Listener and Talker.

4.4.4.4.1 AVB: Harpoon AVTP Listener

4.4.4.4.1.1 AVB setup preparation

An i.MX 8M Plus EVK with Real-time Edge SW v2.5 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 15. AVB Audio setup

4.4.4.4.1.2 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
    ```

4.4.4.4.1.3 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.

The below steps must be executed to enable the AVB Listener on Harpoon side:

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 FreeRTOS audio application using the following command at the Linux prompt:

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

3. Start the AVB pipeline, connecting the AVTP source element (stream #0) to the SAI output (e.g.: HiFiBerry board)

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

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
```

4.4.4.4.1.4 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 <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 <control_type> <entity_id> <control_index>           Get a control
 value where control_type must be uint8 or utf8
        -l                                                      list discovered
 AVDECC entities
        -c <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id> <flags>      connect a stream between a talker and a
 listener
        -d <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id>              disconnect a stream between a talker and a
 listener
        -r <listener_entity_id> <listener_unique_id>            Get information
 about a listener sink
        -t <talker_entity_id> <talker_unique_id>                Get information
 about a talker source
        -s <talker_entity_id> <talker_unique_id> <index>        Get information
 from a talker about a given connection/stream
        -T <talker_entity_id> <talker_unique_id> <start|stop>          Send
 START_STREAMING or STOP_STREAMING command to a talker
```

```
        -L <listener_entity_id> <listener_unique_id> <start|stop>       Send
 START_STREAMING or STOP_STREAMING command to a listener
        -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: 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 command:

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

In the below 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
```

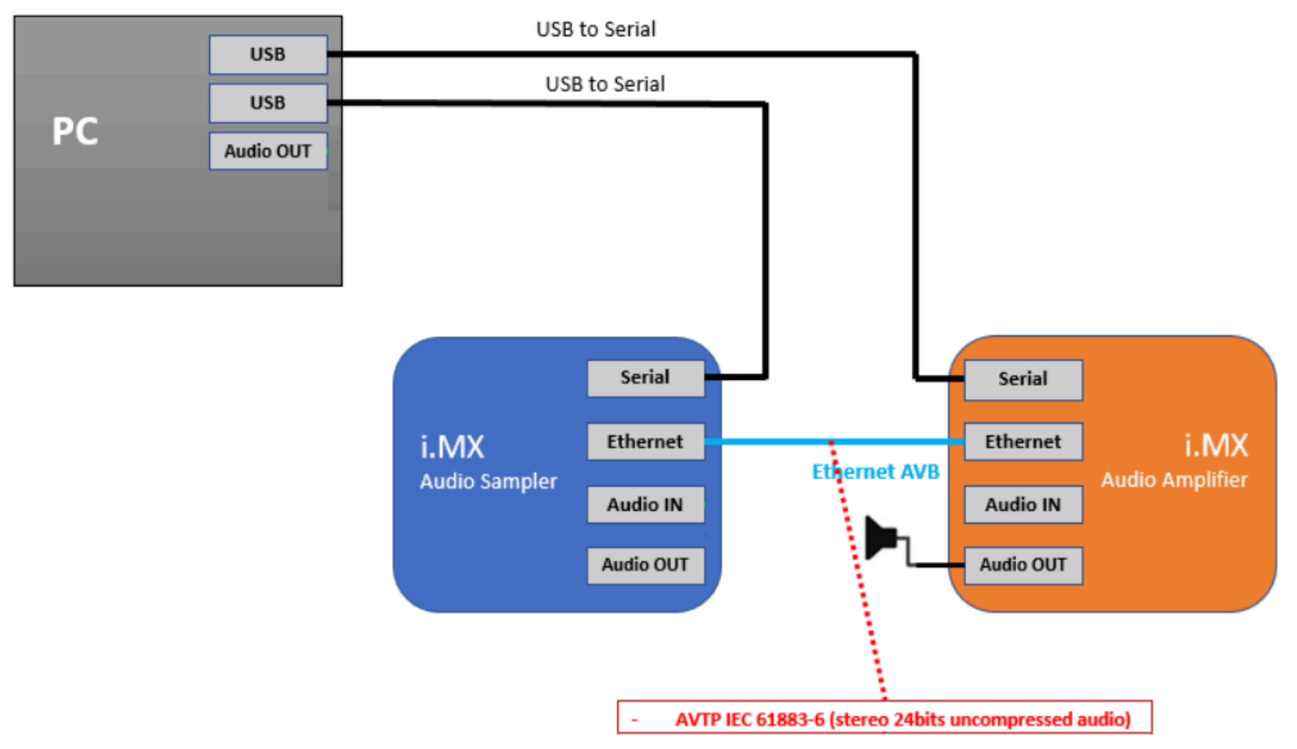
Once the stream is connected, the audio file can be heard on the SAI output lines.

4.4.4.4.2 AVB: Harpoon AVTP Talker

4.4.4.4.2.1 AVB setup preparation

An i.MX 8M Plus EVK with Real-time Edge SW v2.5 (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 16. AVB Audio setup

4.4.4.4.2.2 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 listenning 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
```

4.4.4.4.2.3 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.

The below steps must be executed to enable the AVB Talker on Harpoon side:

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 FreeRTOS audio application using the following command at the Linux prompt:

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

3. Start the AVB pipeline, connecting the SAI input (e.g.: HifiBerry board) to the AVTP sink element (stream #0)

```
# 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
```

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
```

4.4.4.4.2.4 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 Listener (Linux endpoint):

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

Usage:
app [options]

Options:
        -S <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 <control_type> <entity_id> <control_index>           Get a control
 value where control_type must be uint8 or utf8
        -l                                                      list discovered
 AVDECC entities
        -c <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id> <flags>      connect a stream between a talker and a
 listener
        -d <talker_entity_id> <talker_unique_id> <listener_entity_id>
 <listener_unique_id>              disconnect a stream between a talker and a
 listener
        -r <listener_entity_id> <listener_unique_id>            Get information
 about a listener sink
        -t <talker_entity_id> <talker_unique_id>                Get information
 about a talker source
        -s <talker_entity_id> <talker_unique_id> <index>        Get information
 from a talker about a given connection/stream
        -T <talker_entity_id> <talker_unique_id> <start|stop>           Send
 START_STREAMING or STOP_STREAMING command to a talker
        -L <listener_entity_id> <listener_unique_id> <start|stop>       Send
 START_STREAMING or STOP_STREAMING command to a listener
        -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 command:

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

In the below 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.

4.4.4.4.3 AVB Connect Harpoon Listener and Talker

The AVB Listener 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 AVB endpoints (i.MX 8M Plus LPDDR4 EVK)
- One AVB/TSN bridge (e.g LS1028ARDB)

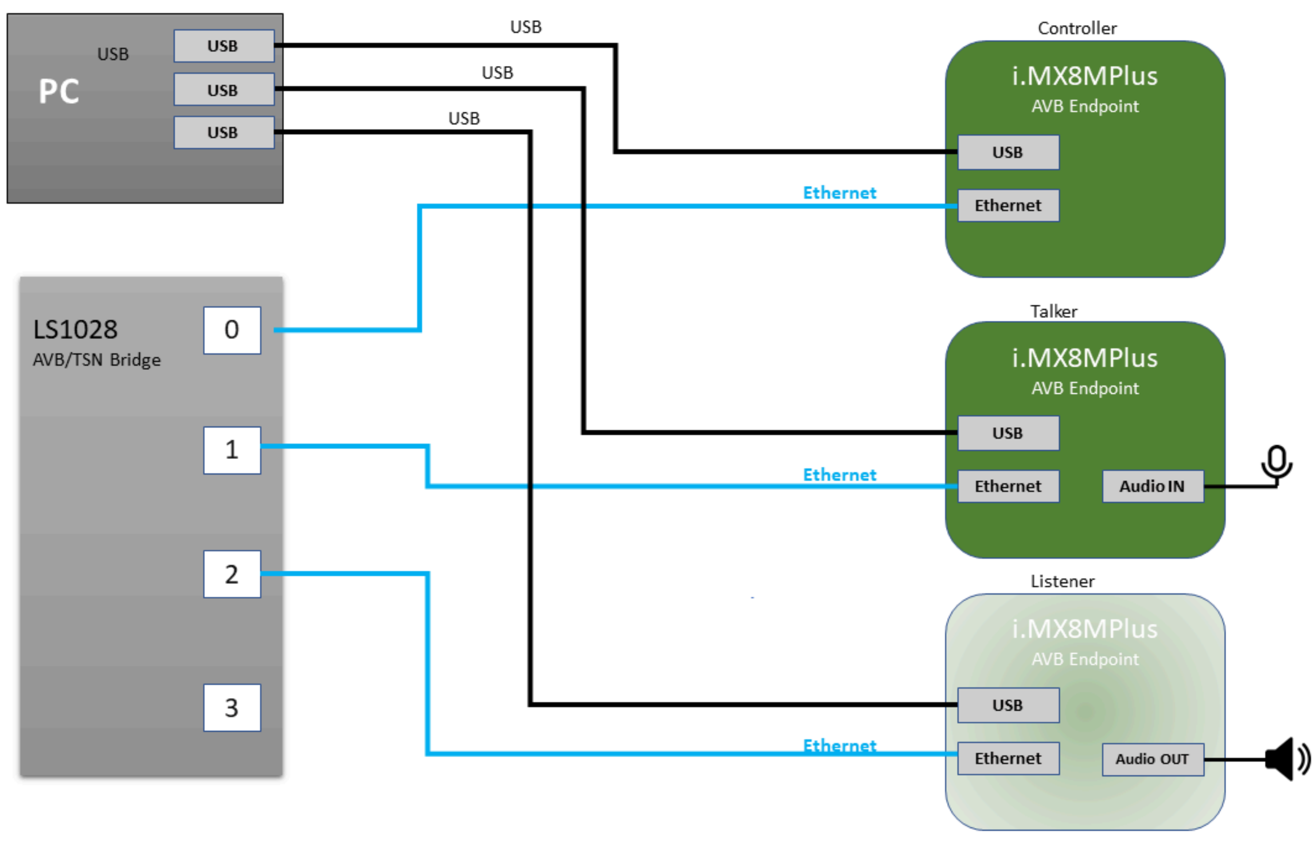


Figure 17. Harpoon AVTP Talker/Listener setup

The steps 1 and 2 from "AVB Listener configuration (Harpoon)" must be executed to enable the AVB Listener on Harpoon side, then:

1. Start the AVB pipeline, connecting the AVTP source element (stream #0) to the SAI output (e.g.: HifiBerry board)

```
# 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
```

2. 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
```

The steps 1 and 2 from "AVB Talker configuration (Harpoon)" must be executed to enable the AVB Talker on Harpoon side, then:

1. Start the AVB pipeline, connecting the SAI input (e.g.: HifiBerry board) to the AVTP sink element (stream #0)

```
# 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
```

2. 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
```

On the controller board, the AVDECC controller must be enabled:

1. Power on the i.MX board and let the boot process complete
2. Enable the GenAVB/TSN systemd service to start the stack automatically on next reboot:

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

3. Reboot the board. The change is saved across reboots, so this has only to be done once.
4. 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
```

On the controller board, the Talker and Listener AVTP streams must be connected:

1. Discover the AVB Endpoints using:

```
# genavbcontroller-app -l
```

2. Connect the discovered streams:
   In the below example, the Listener's stream #0 is connected to the Talker’s stream #0:

```
# genavb-controller-app -c 0x49fddbeef0000 0 0x49fddcafe0000 0 0
```

   Audio from the SAI input (e.g.: HifiBerry board) in the Talker's board should be heard at the SAI output (e.g.: HifiBerry board) in the Listener's board.

4.4.4.5 Playing an SMP full audio pipeline

The use case for SMP audio pipeline is only supported on Zephyr which runs SMP kernel on two CPU Cores, it will create and bind 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 be able to run pipelines that consume more than one single core CPU resources.

The main approach used is to split 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, that 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.

The reference audio application is splitting the pipeline used by "full audio pipeline" use case into two audio data pipelines, each pipeline runs on a dedicated thread binded to a dedicated CPU Core.

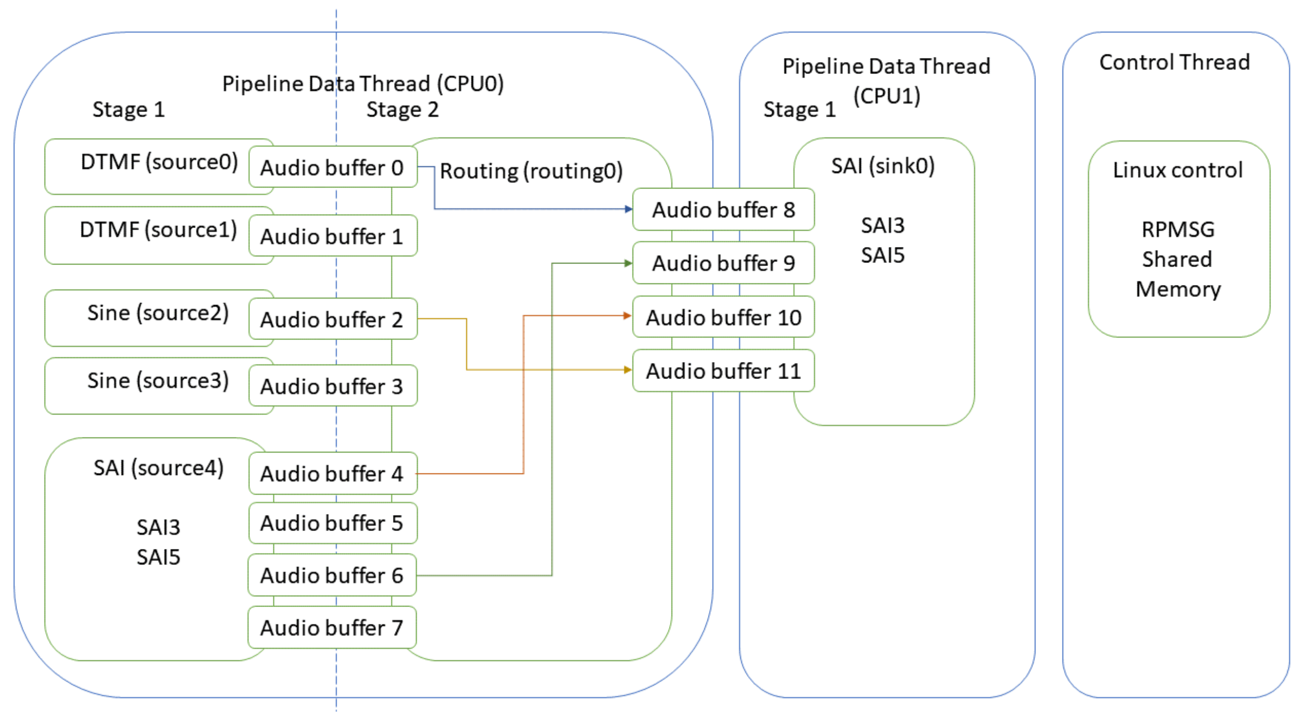


Figure 18. SMP Audio Pipeline with two threads and CPU cores

To run the Zephyr audio SMP pipeline application, the following command can be run 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).*

And run the harpoon service with systemd in order 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, the playback must be stopped with the following command:

```
# harpoon_ctrl audio -s
```

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 (only on i.MX 8M Plus EVK):
  - Loopback example: use of an internal loopback to send and receive CAN or CAN FD messages on the same board. In this example, no CAN bus cable is needed.
  - Ping-pong example: two boards are connected through their CAN bus connectors (J19) with a male-male DB9 CAN bus cable. The latter can either be purchased or built following the CAN pinout standard. Endpoint A (board A) sends CAN or CAN FD messages to Endpoint B (board B). Endpoint B uses two receiving queues to receive messages in turns, and prints the message content (and the receiving queue number) to the terminal after any queue is full.
- 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
  - Full TSN stack based application, running a gPTP stack and sending/receiving TSN packets on a TSN network:
    - Through the ENET_QOS interface, acting as a controller/IO device (i.MX 8M Plus EVK).
    - Through the ENET interface, acting as a controller/IO device (i.MX 8M Mini 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 instance:

```
# harpoon_set_configuration.sh freertos 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:

```
Industrial FlexCAN options:
        -r <id>        run CAN mode id:
                       0 - loopback
                       1 - interrupt
                       2 - pingpong
        -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)
        -r <id>        run ethernet mode id:
                       0 - genAVB/TSN stack (FreeRTOS only)
                       1 - mcux-sdk API (imx8m{m,n} ENET, imx8mp ENET_QoS on
 Zephyr)
                       2 - mcux-sdk API with PHY loopback mode (imx8mp ENET_QoS
 on Zephyr)
        -i <role>      for genAVB/TSN: endpoint role (default 'controller', if
 not specified)
                       0 - role is 'IO device 0'
                       1 - role is 'IO device 1'
        -s             stop ethernet
```

4.5.3 Running the industrial application: examples

4.5.3.1 FlexCAN use cases

To start a FlexCAN based communication, you can choose between the CAN or the CAN FD protocol, depending on your needs. To select the CAN protocol, you can add `-o 0` to the run command. To select the CAN FD protocol, you need to add `-o 1`. If nothing added, the selected protocol is the CAN one, as it is the default one.

Loopback

Execute the below command:

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

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

```
# harpoon_ctrl can -s
```

Ping-pong

One board must be chosen as node A and the other board as node B. (Note: Node B should start first) Data is sent from the node A to the node B.

Type this command to start receiving data on board B:

```
# harpoon_ctrl can -n 1 -r 2
```

Type this command on board A to start transferring data:

```
# harpoon_ctrl can -n 0 -r 2
```

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

```
# harpoon_ctrl can -s
```

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, one may 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.
   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.
   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 will be loopbacked and the port will received 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 as well as Time-Sensitive Networking (TSN) functionalities. Some functions for the latter do require special TSN hardware support, available in the i.MX 8M Plus SoC for instance.

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) runs in the Cortex-A53 FreeRTOS cell. The IO devices, which can be any TSN endpoint (i.MX 8M Plus, 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 LPDDR4 EVK, or optionally an i.MX RT1170 EVK)
- One TSN bridge (LS1028ARDB)

Note: *The second IO Device is optional.*

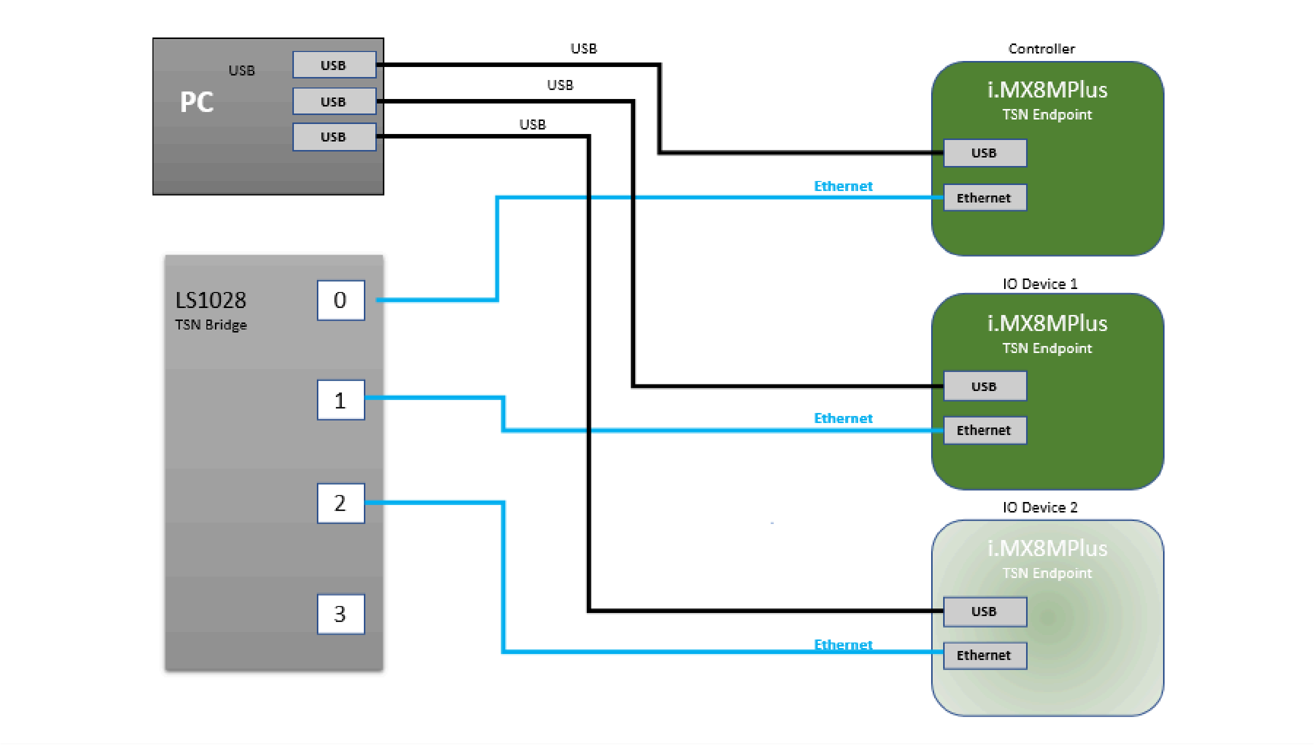


Figure 19. TSN endpoint sample application setup

4.5.3.3.2 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.

4.5.3.3.2.1 i.MX RT1170 TSN Endpoint - IO Device (Optional)

If using an i.MX RT1170 as the IO device, first flash the latest GenAVB/TSN Endpoint image (https://mcuxpresso.nxp.com/download/685e45c32bb022c898e4d11e6d914010).

Once the 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.

4.5.3.3.2.2 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. Enabling bridge interface is dependent on the board used.

TSN Bridge Configuration

Use the following commands to configure bridge on LS1028ARDB:

```
# ls /sys/bus/pci/devices/0000:00:00.5/net/
```

Get switch device interfaces for swp0, swp1, swp2, and swp3 as shown below:

```
ip link set 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*.

4.5.3.3.3 Running the TSN use case

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

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

The expected output in the inmate console is:

```
INFO: main_task             : Industrial application started!


               [5835/93378]
INFO: ethernet_avb_tsn_init : ethernet_avb_tsn_init
INFO           0 app gavb_stack_init          : talker_entity_id
 0x0000000000000000
INIT           0.000000000 stack-freertos genavb_init                    :
 NXP's GenAVB/TSN stack version dev-d71ce4fc
INIT           0.000000000 stack-freertos hw_avb_timer_init              :
 hw_timer_init done
INIT           0.000000000 stack-freertos hw_clock_init                  :
 rate: 24000000, period: 100000000, mult(to ns): 699050667, shift(to ns): 24,
 mult(to cycles): 103079215, shift(to cycles): 32
INIT           0.000000000 stack-freertos hw_clock_register              : hw
 clock id: 1 registered
INIT           0.000000000 stack-freertos hw_timer_register              :
 hw_timer(C0600080) of clock id: 1 registered
INIT           0.000000000 stack-freertos hw_timer_register              :
 hw_timer(C06000C8) of clock id: 1 registered
INIT           0.000000000 stack-freertos hw_timer_register              :
 hw_timer(C0600110) of clock id: 1 registered
INFO           0.000000000 stack-freertos hw_avb_timer_register_device   :
 dev(C06003D0) , ref clock 24000000 Hz, min delay cycles 240
INFO           0.000000000 stack-freertos gpt_hw_timer_set_period        :
 gpt_dev (C06003A0) set period 125(us), 3000(cycles)
```

```
INIT            0.000000000 stack-freertos gpt_init                            :
 gpt_init : registered AVB HW timer(C06003D0) channel: 0, prescale: 1
ERR             0.000000000 stack-freertos gpt_init                            :
 gpt_init : failed to register GPT media clock recovery
INIT            0.000000000 stack-freertos __port_init                         :
 port(0): C0604DF8
INIT            0.000000000 stack-freertos enet_qos_init                       :
 port(0) enet(0) core clock: 125000000 Hz, ptp ref clock: 100000000 Hz, ptp/
system clock: 80000000 Hz
INIT            0.000000000 stack-freertos enet_qos_init                       :
 port(0) enet(0) num TX queue: 5, num RX queue: 4
INIT            0.000000000 stack-freertos hw_clock_init                       :
 rate: 1000000000, period: 3b9ac9ffc4653600, mult(to ns): 1, shift(to ns): 0,
 mult(to cycles): 1, shift(to cycles): 0
INIT            0.000000000 stack-freertos hw_clock_register                   : hw
 clock id: 2 registered
INIT            0.000000000 stack-freertos hw_timer_register                   :
 hw_timer(C0605930) of clock id: 2 registered
INIT            0.000000000 stack-freertos hw_timer_register                   :
 hw_timer(C0605970) of clock id: 2 registered
INIT            0.000000000 stack-freertos hw_timer_register                   :
 hw_timer(C06059B0) of clock id: 2 registered, pps support
INIT            0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 0 success, flags: 0
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 1 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 2 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 3 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 4 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 5 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 6 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 7 has no hw clock
INIT            0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 8 success, flags: 1
INIT            0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 9 success, flags: 0
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 10 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 11 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 12 has no hw clock
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 13 has no hw clock
INIT            0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 14 success, flags: 4
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 15 has no hw clock

                    [6599/94179]
ERR             0.000000000 stack-freertos _os_clock_init                      :
 clock ID: 16 has no hw clock
INFO            0.000000000 stack-freertos net_qos_map_traffic_class_to_hw_ :
 port(0) num tc: 5, num sr: 2, num hw queues: 5
```

```
INFO          0.000000000 stack-freertos net_qos_map_traffic_class_to_hw_ : num
 hw queues: 5, num cbs: 2
INFO          0.000000000 stack-freertos net_qos_map_traffic_class_to_hw_ :
 tc(0)->hw_queue_id: 0, flags: 2, hw queue prop: 1
INFO          0.000000000 stack-freertos net_qos_map_traffic_class_to_hw_ :
 tc(1)->hw_queue_id: 1, flags: 2, hw queue prop: 1
INFO          0.000000000 stack-freertos net_qos_map_traffic_class_to_hw_ :
 tc(2)->hw_queue_id: 2, flags: 2, hw queue prop: 1
INFO          0.000000000 stack-freertos net_qos_map_traffic_class_to_hw_ :
 tc(3)->hw_queue_id: 3, flags: 1, hw queue prop: 2
INFO          0.000000000 stack-freertos net_qos_map_traffic_class_to_hw_ :
 tc(4)->hw_queue_id: 4, flags: 1, hw queue prop: 2
INFO          0.000000000 stack-freertos hw_timer_request                 :
 hw_timer(C0600110)
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0626B78), queue: 0
INIT          0.000000000 stack-freertos net_tx_task                      :
 networking(C060F2A8) tx task started
INIT          0.000000000 stack-freertos net_rx_task                      :
 networking(C0626A38) rx task started
INIT          0.000000000 stack-freertos net_task_init                    :
 networking started
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0605A68), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0605A68), queue: 0
INIT          0.000000000 stack-freertos hw_avb_timer_start               :
 hw_timer_start done
INIT          0.000000000 stack-freertos management_task                  :
 management task started
INFO          0.000000000 stack-freertos __net_tx_init                    :
 socket(C0658680) port_id(0)
INIT          0.000000000 stack-freertos ipc_tx_init                      :
 ipc(C06039C8, C06588A0) success
INIT          0.000000000 stack-freertos ipc_tx_init                      :
 ipc(C0603AF8, C0658A40) success
INIT          0.000000000 stack-freertos ipc_rx_init                      :
 ipc(C0603898, C0658BE0) success
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C06585A0), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C06585A0), queue: C06583A0
INIT          0.000000000 management mac_service_init                 :
 mac(C06585D8) done
INIT          0.000000000 management management_init                  :
 management(C0658550) done
INIT          0.000000000 stack-freertos management_task                  :
 started
INIT          0.000000000 stack-freertos phy_task                         :
 phy(1) task started
INIT          0.000000000 stack-freertos management_task_init             :
 management main completed
INIT          0.000000000 stack-freertos gptp_task                        :
 gptp task started
INFO          0.000000000 gptp   gptp_check_config               : gptp config
 is valid
INIT          0.000000000 gptp   gptp_init                       :
 gptp(C065A150) (profile 0 - rsync 0 - num ports = 1 - force_2011 = 0)
INIT          0.000000000 stack-freertos ipc_tx_init                      :
 ipc(C0602B88, C0665F40) success
```



```
INIT          0.000000000 stack-freertos ipc_tx_init                     :
 ipc(C0602CB8, C06660E0) success
INFO          0.000000000 stack-freertos __net_rx_init                   :
 socket(C0666280)
INFO          0.000000000 stack-freertos __net_tx_init                   :
 socket(C06664A0) port_id(0)
INFO          0.000000000 stack-freertos net_add_multi                   :
 port(0) 01:80:c2:00:00:0e
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C0665530), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                 :
 os_timer(C0665530), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C0665598), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                 :
 os_timer(C0665598), queue: C0659FA0
INIT          0.000000000 gptp   gptp_cmlds_init                 : CMLDS link
 port (0) initialized
INFO          0.000000000 gptp   dump_priority_vector            : domain(0,
 0) system priority vector: root identity 00bbccfffeddee12
INFO          0.000000000 gptp   dump_priority_vector            : system
 priority vector: priority1 255      priority2 248
INFO          0.000000000 gptp   dump_priority_vector            : system
 priority vector: class 255  accuracy 255
INFO          0.000000000 gptp   dump_priority_vector            : system
 priority vector: variance 17258
INFO          0.000000000 gptp   dump_priority_vector            : system
 priority vector: source port identity 00bbccfffeddee12, port number 0
INFO          0.000000000 gptp   dump_priority_vector            : system
 priority vector: port number 0 steps removed 0
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C0665600), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                 :
 os_timer(C0665600), queue: C0659FA0
INIT          0.000000000 gptp   gptp_instance_init              : Configuring
 Port(0) (C0663B30) domain(0, 0) delayMechanism(P2P)
INIT          0.000000000 gptp   gptp_port_init_timers           : Port(0)
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C0665668), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                 :
 os_timer(C0665668), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C06656D0), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                 :
 os_timer(C06656D0), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C0665738), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                 :
 os_timer(C0665738), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C06657A0), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                 :
 os_timer(C06657A0), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C0665808), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                 :
 os_timer(C0665808), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create             :
 os_timer(C0665870), queue: 0
```

```
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665870), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C06658D8), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C06658D8), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665940), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665940), queue: C0659FA0
ERR           0.000000000 common timer_start                      :
 timer(C0663FD8) 0ms period
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C06659A8), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C06659A8), queue: C0659FA0
INIT          0.000000000 gptp   gptp_instance_init               :
 instance(C06635F0) domain(0, 0) is enabled (gm capable 0)
INFO          0.000000000 gptp   dump_priority_vector             : domain(1,
 -1) system priority vector: root identity 00bbccfffeddee12
INFO          0.000000000 gptp   dump_priority_vector             : system
 priority vector: priority1 255      priority2 248
INFO          0.000000000 gptp   dump_priority_vector             : system
 priority vector: class 255  accuracy 255
INFO          0.000000000 gptp   dump_priority_vector             : system
 priority vector: variance 17258
INFO          0.000000000 gptp   dump_priority_vector             : system
 priority vector: source port identity 00bbccfffeddee12, port number 0
INFO          0.000000000 gptp   dump_priority_vector             : system
 priority vector: port number 0 steps removed 0
INIT          0.000000000 gptp   gptp_instance_init               : Configuring
 Port(0) (C0664AB0) domain(1, -1) delayMechanism(COMMON_P2P)
INIT          0.000000000 gptp   gptp_port_init_timers            : Port(0)
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665A10), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665A10), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665A78), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665A78), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665AE0), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665AE0), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665B48), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665B48), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665BB0), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665BB0), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665C18), queue: 0
INFO          0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665C18), queue: C0659FA0
INFO          0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665C80), queue: 0
```

```
INFO           0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665C80), queue: C0659FA0
INFO           0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665CE8), queue: 0
INFO           0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665CE8), queue: C0659FA0
ERR            0.000000000 common timer_start                      :
 timer(C0664F58) 0ms period
INFO           0.000000000 stack-freertos timer_system_create              :
 os_timer(C0665D50), queue: 0
INFO           0.000000000 stack-freertos os_timer_create                  :
 os_timer(C0665D50), queue: C0659FA0
INIT           0.000000000 gptp   gptp_instance_init               :
 instance(C0664570) domain(1, -1) is disabled (gm capable 0)
INFO           0.000000000 gptp   gptp_link_down                   : Port(0):
 link is DOWN
INIT           0.000000000 stack-freertos ipc_rx_init                      :
 ipc(C06039C8, C0666810) success
INIT           0.000000000 stack-freertos ipc_tx_init                      :
 ipc(C0603898, C06669B0) success
INIT           0.000000000 stack-freertos ipc_rx_init                      :
 ipc(C0602A58, C0666DC0) success
INFO           0.000000000 gptp   gptp_link_down                   : Port(0):
 link is DOWN
INIT           0.159357387 stack-freertos gptp_task_init                   :
 gptp main completed
INIT           0.159357387 stack-freertos srp_task                         : srp
 task started
INIT           0.159357387 stack-freertos ipc_rx_init                      :
 ipc(C06039C8, C066B5E0) success
INIT           0.159357387 stack-freertos ipc_tx_init                      :
 ipc(C0603898, C066B780) success
INFO           0.159357387 stack-freertos __net_rx_init                    :
 socket(C066B920)
INFO           0.159357387 stack-freertos __net_tx_init                    :
 socket(C066BB40) port_id(0)
INIT           0.159357387 stack-freertos ipc_rx_init                      :
 ipc(C06014F8, C066BF00) success
INIT           0.159357387 stack-freertos ipc_tx_init                      :
 ipc(C0601628, C066C0A0) success
INIT           0.159357387 stack-freertos ipc_tx_init                      :
 ipc(C0601758, C066C240) success
INIT           0.159357387 srp    msrp_map_init                    : done
INFO           0.159357387 stack-freertos timer_system_create              :
 os_timer(C066B388), queue: 0
INFO           0.159357387 stack-freertos os_timer_create                  :
 os_timer(C066B388), queue: C0667BF0
INFO           0.159357387 stack-freertos timer_system_create              :
 os_timer(C066B3F0), queue: 0
INFO           0.159357387 stack-freertos os_timer_create                  :
 os_timer(C066B3F0), queue: C0667BF0
INIT           0.159357387 srp    mrp_init                         :
 mrp_app(C066AD40) done
INIT           0.159357387 srp    msrp_port_init                   : port(0)
 done
INFO           0.159357387 srp    msrp_create_domain               : port(0)
 domain(6, 3, 2) created, num domains 1
INFO           0.159357387 stack-freertos timer_system_create              :
 os_timer(C066B458), queue: 0
```

```
INFO           0.159357387 stack-freertos os_timer_create                  :
 os_timer(C066B458), queue: C0667BF0
INFO           0.159357387 srp    mrp_alloc_attribute              :
 mrp_app(C066AD40) port(0) attr(C066BD60, MSRP_ATTR_TYPE_DOMAIN)
INFO           0.159357387 srp    mrp_mad_join_request             :
 mrp_app(C066AD40) port(0) attr(C066BD60, MSRP_ATTR_TYPE_DOMAIN) new(1)
INFO           0.159357387 srp    msrp_create_domain               : port(0)
 domain(5, 2, 2) created, num domains 2
INFO           0.159357387 srp    mrp_alloc_attribute              :
 mrp_app(C066AD40) port(0) attr(C066C3E0, MSRP_ATTR_TYPE_DOMAIN)
INFO           0.159357387 srp    mrp_mad_join_request             :
 mrp_app(C066AD40) port(0) attr(C066C3E0, MSRP_ATTR_TYPE_DOMAIN) new(1)
INFO           0.159357387 srp    msrp_domain_update_boundary_port : port(0)
 class(0), srp boundary 1
INFO           0.159357387 srp    msrp_domain_update_boundary_port : port(0)
 class(1), srp boundary 1
INFO           0.159357387 stack-freertos net_add_multi                    :
 port(0) 01:80:c2:00:00:0e
INFO           0.159357387 srp    msrp_port_enable                 : port(0)
 enabled
INFO           0.159357387 srp    msrp_enable                      :
 msrp(C066ABB8) enabled
INIT           0.159357387 srp    msrp_init                        :
 msrp(C066ABB8) done
INIT           0.159357387 stack-freertos ipc_rx_init                      :
 ipc(C0601C18, C066C470) success
INIT           0.159357387 stack-freertos ipc_tx_init                      :
 ipc(C0601D48, C066C610) success
INIT           0.159357387 stack-freertos ipc_tx_init                      :
 ipc(C0601E78, C066C7B0) success
INIT           0.159357387 srp    mvrp_map_init                    : done
INFO           0.159357387 stack-freertos timer_system_create              :
 os_timer(C066B4C0), queue: 0
INFO           0.159357387 stack-freertos os_timer_create                  :
 os_timer(C066B4C0), queue: C0667BF0
INFO           0.159357387 stack-freertos timer_system_create              :
 os_timer(C066B528), queue: 0
INFO           0.159357387 stack-freertos os_timer_create                  :
 os_timer(C066B528), queue: C0667BF0
INFO           0.159357387 stack-freertos timer_system_create              :
 os_timer(C066B590), queue: 0
INFO           0.159357387 stack-freertos os_timer_create                  :
 os_timer(C066B590), queue: C0667BF0
INIT           0.159357387 srp    mrp_init                         :
 mrp_app(C066B078) done
INFO           0.159357387 stack-freertos net_add_multi                    :
 port(0) 01:80:c2:00:00:21
INIT           0.159357387 srp    mvrp_port_init                   : port(0)
 done
INIT           0.159357387 srp    mvrp_init                        :
 mvrp(C066B010) done
INIT           0.159357387 srp    mmrp_init                        :
 mmrp(C066AB38) done
INIT           0.159357387 srp    srp_init                         :
 srp(C0667DA0) done
INIT           0.159357387 stack-freertos srp_task                         :
 started
INFO           0.159357387 srp    msrp_port_status                 :
 msrp(C066ABB8) port(0) operational (0)
```

```
INFO           0.159357387 srp    mvrp_port_status                 :
 mvrp(C066B010) port(0) operational (0)
INIT           0.159357387 stack-freertos phy_task                        :
 phy(1) initialized
INIT           0.159357387 stack-freertos 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
INFO           0.159357387 stack-freertos hw_timer_request                :
 hw_timer(C06059B0) pps
INFO           0.159357387 stack-freertos os_timer_create                 :
 os_timer(C06532B0), queue: 0
INFO           0 app gavb_pps_init           : success, clk_id: 1
INFO           0 app cyclic_task_init        : cyclic task type: 0, id: 0
INFO           0 app cyclic_task_init        : task params
INFO           0 app cyclic_task_init        : task_period_ns        : 100000
INFO           0 app cyclic_task_init        : task_period_offset_ns : 0
INFO           0 app cyclic_task_init        : transfer_time_ns      : 50000
INFO           0 app cyclic_task_init        : sched_traffic_offset  : 35000
INFO           0 app cyclic_task_init        : use_fp                : 0
INFO           0 app cyclic_task_init        : use_st                : 1
INFO           0.159357387 stack-freertos __net_rx_init                   :
 socket(C066EAD0)
INFO           0.159357387 stack-freertos net_add_multi                   :
 port(0) 91:e0:f0:00:fe:71
INFO           0.159357387 stack-freertos __net_tx_init                   :
 socket(C066ECF0) port_id(0)
INFO           0.159357387 stack-freertos hw_timer_request                :
 hw_timer(C0605970)
INFO           0.159357387 stack-freertos os_timer_create                 :
 os_timer(C066FB30), queue: 0
INFO           0 app cyclic_task_init        : success
INFO           0 app tsn_net_st_config_enable : scheduled traffic config enabled
INFO           0.159357387 stack-freertos __net_rx_init                   :
 socket(C06719B0)
INFO           0.159357387 stack-freertos net_add_multi                   :
 port(0) 91:e0:f0:00:fe:a0
INFO           0 app alarm_task_monitor_init  : success
INFO           0.159357387 srp    msrp_vector_add_event            : port(0)
 domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_NEW
INFO           0.159357387 srp    msrp_vector_add_event            : port(0)
 domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_NEW
INFO           0.159357387 srp    msrp_vector_add_event            : port(0)
 domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_NEW
INFO           0.159357387 srp    msrp_vector_add_event            : port(0)
 domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_NEW
INFO           0.159357387 srp    msrp_vector_add_event            : port(0)
 domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
INFO           0.159357387 srp    msrp_vector_add_event            : port(0)
 domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN MRP_ATTR_EVT_JOINMT
INFO           2.161474650 stack-freertos port_up                         :
 port(0) up, speed:2, duplex:1
```

```
INFO          2.161474650 gptp   gptp_link_up                     : Port(0):
 link is UP
```

To stop the Ethernet use case (to eventually restart it), the previous run 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 GPT timer and relative to the hardware IRQ event time, with sub-microsecond precision.

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 v2.3.0
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 the `rt_latency` application is shown:

```
Latency options:
        -r <id>        run latency test case id
        -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:

```
---
Running test case 1:
benchmark_task: running
stats(C0601260) irq delay (ns) min 1041 mean 1169 max 3250 rms^2 1375559
 stddev^2 8797 absmin 1041 absmax 3250
n_slot 21 slot_size 200
0 0 0 0 0 499 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
stats(C06016C0) irq to sched (ns) min 2916 mean 3265 max 6125 rms^2 10698499
 stddev^2 37779 absmin 2916 absmax 6125
n_slot 21 slot_size 1000
0 0 5 491 2 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

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

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

Table 7. 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	Min	Average	Max	Stddev

Table 7. Real-time latencies measured on i.MX 8M Plus/FreeRTOS (in ns)*...continued*

No system load	708	720	3,041	11,055	2,571	2,571	5,375	246,547
Low priority task CPU load	708	733	3,041	11,329	2,458	2,522	4,791	98,168
Low priority IRQ load	10,916	11,184	15,791	396,895	12,833	12,833	18,041	476,218
Low priority task CPU load, mutex	708	724	3,458	540,485	2,416	2,563	5,416	221,287
Linux CPU + memory load	708	736	3,041	12,140	2,458	2,458	2,458	2,458
RTOS cold cache	708	765	3,416	65,735	2,458	2,750	6,458	542,859

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

Description	i.MX 8M Mini IRQ Latency (ns)				i.MX 8M Mini Task Latency (ns)			
	Min	Average	Max	Stddev	Min	Average	Max	Stddev
No system load	1,166	1,229	3,625	46,935	3,000	3,091	6,583	131,222
Low priority task CPU load	1,166	1,213	3,708	23,623	3,000	3,059	5,875	51,385
Low priority IRQ load	11,083	11,639	16,625	155,293	12,958	13,481	18,708	174,034
Low priority task CPU load, mutex	1,166	1,218	3,250	24,982	3,000	3,069	5,583	67,273
Linux CPU + memory load	1,166	1,214	4,791	34,997	3,000	3,057	7,625	7,625
RTOS cold cache	1,166	1,218	3,500	31,464	3,000	3,142	6,833	172,301

Table 9. 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	Min	Average	Max	Stddev
No system load	750	776	3,375	58,221	2,583	2,626	5,250	93,990
Low priority task CPU load	750	761	3,208	22986	2,583	2,620	5,125	77,623
Low priority IRQ load	875	884	1,208	744	11,375	11,514	16,291	227,942
Low priority task CPU load, mutex	750	763	3,083	19,989	2,583	2,606	4,916	41,151
Linux CPU + memory load	750	761	3,250	23,783	2,583	2,608	5,125	49,572
RTOS cold cache	750	942	3,625	396,498	2,583	3,047	6,666	923,414

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: *In the current release, the virtio_net application is only supported under FreeRTOS on i.MX 8M Mini EVK for Yocto Real-time Edge SW (i.MX BSP Yocto not supported).*

This application will start a Virtio networking backend on Jailhouse inmate cell, Linux will run Virtio networking frontend which provide a virtual network interface, backend owns physical ENET port and share with frontend by using Virtio communication between frontend and backend.

4.7.2 Running the Virtio Networking application

To use the virtio_net 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 instance:

```
# 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 industrial application:

```
# systemctl start harpoon
```

Once the harpoon service has been started, virtio_net backend application will be started with the following log in console of inmate cell:

```
Starting Virtio networking backend...
virtio network device initialization succeed!
ENET: Wait for PHY link up...
ENET: PHY link speed 1000M full-duplex
Switch enabled with enet remote port
Done
```

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 we can find "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 00:01:02:03:04:05  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 will get IP address by DHCP, otherwise need to set IP address by using "ifconfig" command.

Then use "ping" command to check whether 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
```

5 Known Issues

Table 10. Known issues

ID	Description	Workarounds
HRPN-245	Linux cannot access eMMC.	Store root file system on SD card or NFS.
HRPN-447	Audio glitches on i.MX 8M Nano EVK.	-
HRPN-448	RTOS crashes on Ethernet use case restart.	Restart the Jailhouse cell.
HRPN-680	FlexCAN communication does not work consistently on Zephyr. Both CAN and CAN FD protocols present issues.	Investigation in progress.
HRPN-483	Audio glitches on all boards for combination of high frequency and low frame size.	Do not use combinations of the following parameters: • Frame size: 2, 4 • Frequency: 176.4kHz, 192kHz

Table 10. Known issues*...continued*

ID	Description	Workarounds
HRPN-632	Occurrences of command timeout for frame size 2 for Audio SMP pipeline.	-
HRPN-739	When using a third-party AVDECC controller for AVTP stream connections, restarting the pipeline leads to a crash.	Restart the harpoon service
HRPN-755	AVTP Talker/Listener audio streaming has occasional missing audio samples.	Do not use the following parameters: • Frame size: 2, 4
HRPN-779	i.MX 8M Plus: Lower audio volume for on-board Jack output compared to HiFiBerry's output.	-
HRPN-808	When running Audio SMP, you may run into unstabilities: sometimes the pipeline does not emit sound.	-
HRPN-810	Switching from Zephyr to FreeRTOS application may cause harpoon_ctrl command failure.	Refresh the hash table maintained by bash to ensure the use of /usr/local/bin/harpoon_ctrl symlink: `hash -r`

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* and *mcux-sdk* repositories are required to build FreeRTOS based applications and *zephyr* and *hal_nxp* repositories are required to build Zephyr based applications. Additionally, repository *GenAVB_TSN* is needed to build the industrial application. The `west` tool is used to fetch those repositories, along with harpoon-apps Git tree.

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

You need to have both `git` and `west` 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_2.3.0 hww
$ cd hww
$ 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 10.3-2021.07.

To download the toolchain and install it:

```
$ wget https://developer.arm.com/-/media/Files/downloads/gnu-a/10.3-2021.07/
binrel/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf.tar.xz
tar -C /opt/ -xvf gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf.tar.xz
```

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

```
$ export ARMGCC_DIR=/opt/gcc-arm-10.3-2021.07-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` or `rt_latency`.
- `BOARD` is `evkmimx8mm` for i.MX 8M Mini, `evkmimx8mn` for i.MX 8M Nano, `evkmimx8mp` for i.MX 8M Plus, `evkmimx93` for i.MX 93 EVK.
  Note: *In this release, i.MX 93 EVK only supports the `hello_world` application.*
- 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 cross-compile toolchain first, and then set the cross-compile environment:

```
$ export ARMGCC_DIR=/opt/gcc-arm-10.3-2021.07-x86_64-aarch64-none-elf/
```

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, and `evkmimx8mp` for i.MX 8M Plus, `evkmimx93` for i.MX 93 EVK.
  Note: *In this release, i.MX 93 EVK only supports the `hello_world` application.*
- 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 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:

```
$ bitbake meta-toolchain
```

This generates a toolchain installer in 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, the installer name is `fsl-imx-xwayland-glibc-x86_64-meta-toolchain-armv8a-imx8mm-lpddr4-evk-toolchain-6.1-langdale.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/6.1-langdale`.

When the toolchain is installed, different cross-compile variables must be set. This is done by sourcing script `environment-setup-cortexa53-crypto-poky-linux`. For example with default installation path:

```
$ . /opt/fsl-imx-xwayland/6.1-langdale/environment-setup-armv8a-poky-linux
```

The Harpoon control application can then be built:

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

The build generates two artefacts, `harpoon_ctrl` and `harpoon_ctrl_rpmsg` in the same directory and can be used on target.

The Linux root cell uses the Inter-VM Shared Memory (ivshmem) device to communicate with Zephyr inmate cell. `harpoon_ctrl` binary implements this device, and should be used when a Zephyr inmate cell is used.

The Linux root cell uses the Remote Processor Messaging (RPMsg) device to communicate with FreeRTOS inmate cell. `harpoon_ctrl_rpmsg` binary implements this device, and should be used when a FreeRTOS inmate cell is used.

When using `systemd`, the selection of the needed binary is handled. When RPMsg device is needed, a symbolic link `harpoon_ctrl` is created and targets `harpoon_ctrl_rpmsg` binary, which gets deleted once the service is stopped. Hence, you only need to call `harpoon_ctrl` to communicate with FreeRTOS and Zephyr inmate 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.

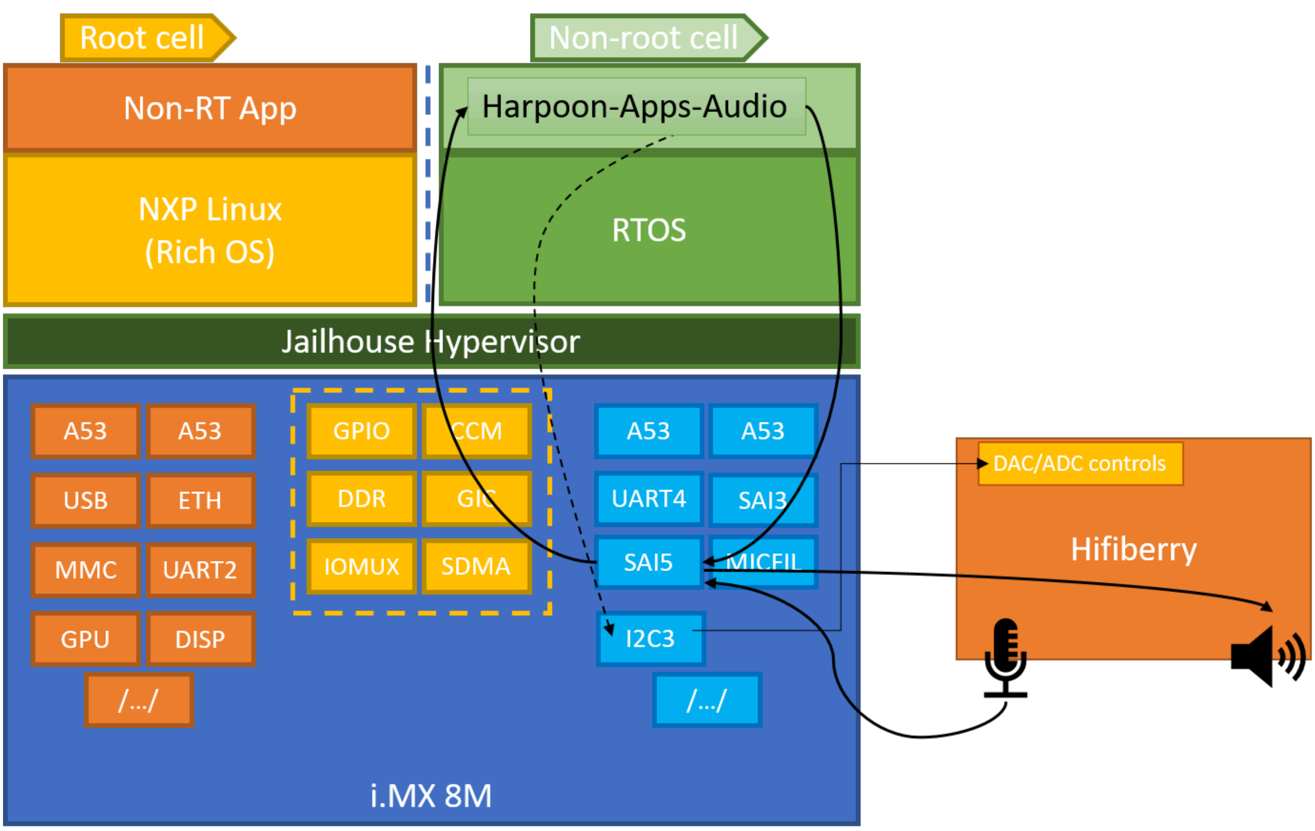


Figure 20. Architecture of audio application

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 chapter gives 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 repository `harpoon-apps`.

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

```
audio/
├── common
│   ├── audio.c
│   ├── audio.h
│   ├── audio_buffer.c
│   ├── audio_buffer.h
│   ├── audio_element.c
│   ├── audio_element.h
│   ├── audio_element_avtp_sink.c
│   ├── audio_element_avtp_sink.h
```

```
│       ├── audio_element_avtp_source.c
│       ├── audio_element_avtp_source.h
│       ├── audio_element_dtmf.c
│       ├── audio_element_dtmf.h
│       ├── audio_element_pll.c
│       ├── audio_element_pll.h
│       ├── audio_element_routing.c
│       ├── audio_element_routing.h
│       ├── audio_element_sai_sink.c
│       ├── audio_element_sai_sink.h
│       ├── audio_element_sai_source.c
│       ├── audio_element_sai_source.h
│       ├── audio_element_sine.c
│       ├── audio_element_sine.h
│       ├── audio_entry.h
│       ├── audio_format.h
│       ├── audio_pipeline.c
│       ├── audio_pipeline.h
│       ├── avb_config.c
│       ├── boards
│       │   ├── evkmimx8mm
│       │   │   ├── app_board.h
│       │   │   ├── avb_hardware.c
│       │   │   ├── clock_config.c
│       │   │   ├── codec_config.c
│       │   │   ├── 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_clock_config.h
│       │       └── sai_config.h
│       ├── pipeline_config.c
│       ├── play_pipeline.c
│       ├── sai_drv.c
│       └── sai_drv.h
│   ├── freertos
│   │   ├── boards
│   │   │   ├── evkmimx8mm
│   │   │   │   ├── app_mmu.h
│   │   │   │   └── armgcc_aarch64
│   │   │   │       ├── CMakeLists.txt
│   │   │   │       ├── build_ddr_debug.sh
│   │   │   │       ├── build_ddr_release.sh
│   │   │   │       └── clean.sh
│   │   │   ├── evkmimx8mn
│   │   │   [...]
│   │   │   └── evkmimx8mp
│   │   │       [...]
│   │   └── main.c
│   └── zephyr
│       ├── CMakeLists.txt
│       ├── boards
│       │
```

```
│       ├── evkmimx8mm
│       │   └── armgcc_aarch64
│       │       ├── build_singlecore.sh
│       │       ├── build_smp.sh
│       │       └── clean.sh
│       ├── evkmimx8mn
│       │   [...]
│       ├── evkmimx8mp
│       │   [...]
│       ├── mimx8mm_evk_a53.conf
│       ├── mimx8mm_evk_a53_smp.conf
│       ├── mimx8mn_evk_a53.conf
│       ├── mimx8mn_evk_a53_smp.conf
│       ├── mimx8mp_evk_a53.conf
│       └── mimx8mp_evk_a53_smp.conf
│   ├── main.c
│   └── prj.conf
```

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

RTOS specific code goes to directory `audio/freertos` and `audio/zephyr`.

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

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

6.3.3 Board specific code

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

Table 11. 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.
`flags.cmake`	CFLAGS and LDFLAGS definitions for building the application.

6.3.4 Controlling application from Linux side

Linux side can control the Harpoon application by sending messages through the ivshmem communication channel provided by Jailhouse or RPMsg communication channel. The mailbox API is used for communication.

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

For ivshmem channel, RTOS is prepared to work with the ivshmem memory:

```
err = ivshmem_transport_init(0, &audio_ctx->mem, &tp, &cmd, &resp);
```

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

```
err = rpmsg_transport_init(RL_BOARD_RPMSG_LINK_ID, EPT_ADDR, "rpmsg-raw",  &tp,
 &cmd, &resp);
```

Then mailbox is initialized:

```
mailbox_init(&audio_ctx->mb, cmd, resp, false, tp);
```

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

```
do {
        command_handler(&m, &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:

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 12. Revision history

Revision number	Date	Substantive changes
2.3	28 March 2023	• Support for AVB Talker in FreeRTOS audio • Support for RPMsg control (FreeRTOS, all boards) • Support for Virtual Ethernet

Table 12. Revision history...*continued*

Revision number	Date	Substantive changes
		• Support for i.MX 93 (preview: hello_world)
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
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.
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
EAR 2.0.1	29 March 2022	Full integration to NXP Real-Time Edge.
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.

9 Legal information

9.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

9.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

9.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

Date of release: 28 March 2023
Document identifier: HRPNUG