IMXHPUG
i.MX Harpoon User's Guide

Rev. LF5.10.72_2.2.0 — 17 December 2021 User Guide

Contents

1 Overview

This document presents the Harpoon release for i.MX 8M device family.

Harpoon provides an environment for developing real-time demanding applications on an RTOS running on the Arm Cortex-A cores in parallel of a Linux distribution.

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

The hardware partitioning is configurable. In this release, an audio centric partitioning is done, with the aim to use the RTOS for audio tasks.

This release supports the following hardware:

- i.MX 8 Series
  - i.MX 8M Mini LPDDR4 EVKB
  - i.MX 8M Nano LPDDR4 EVK
  - i.MX 8M Nano DDR EVK

2 Build Instructions

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

```
mkdir yocto
cd yocto
repo init -u https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-hardknott -m
imx-5.10.72-2.2.0_harpoon.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 EVK board
- *imx8mn-ddr4-evk* for i.MX 8M Nano DDR4 EVK board
- *imx8mm-lpddr4-evk* for i.MX 8M Nano LPDDR4 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/imx-image-core-<machine>/`.

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

```
bzip2 -d -c imx-image-core-<machine>.wic.bz2 | sudo dd of=/dev/mmcblk0 bs=1M
```

The micro-SD card now contains the release.

3 Hardware Presentation

This release being audio centric, an i.MX 8M Mini/Nano EVK is used with an I2S HiFiBerry audio card.

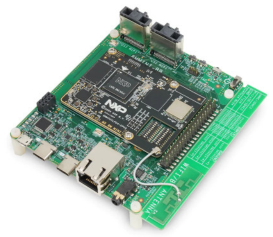

Figure 1. imx8mm-lpddr4-evk for i.MX 8M Mini EVK board

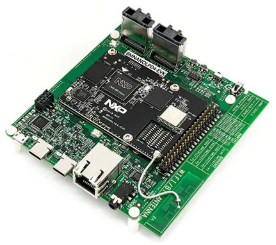

Figure 2. imx8mm-lpddr4-evk for i.MX 8M Nano LPDDR4 EVK board

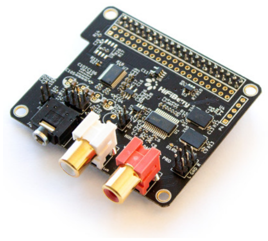

Figure 3. HiFiBerry DAC+ ADC Pro (picture from HiFiBerry's website)

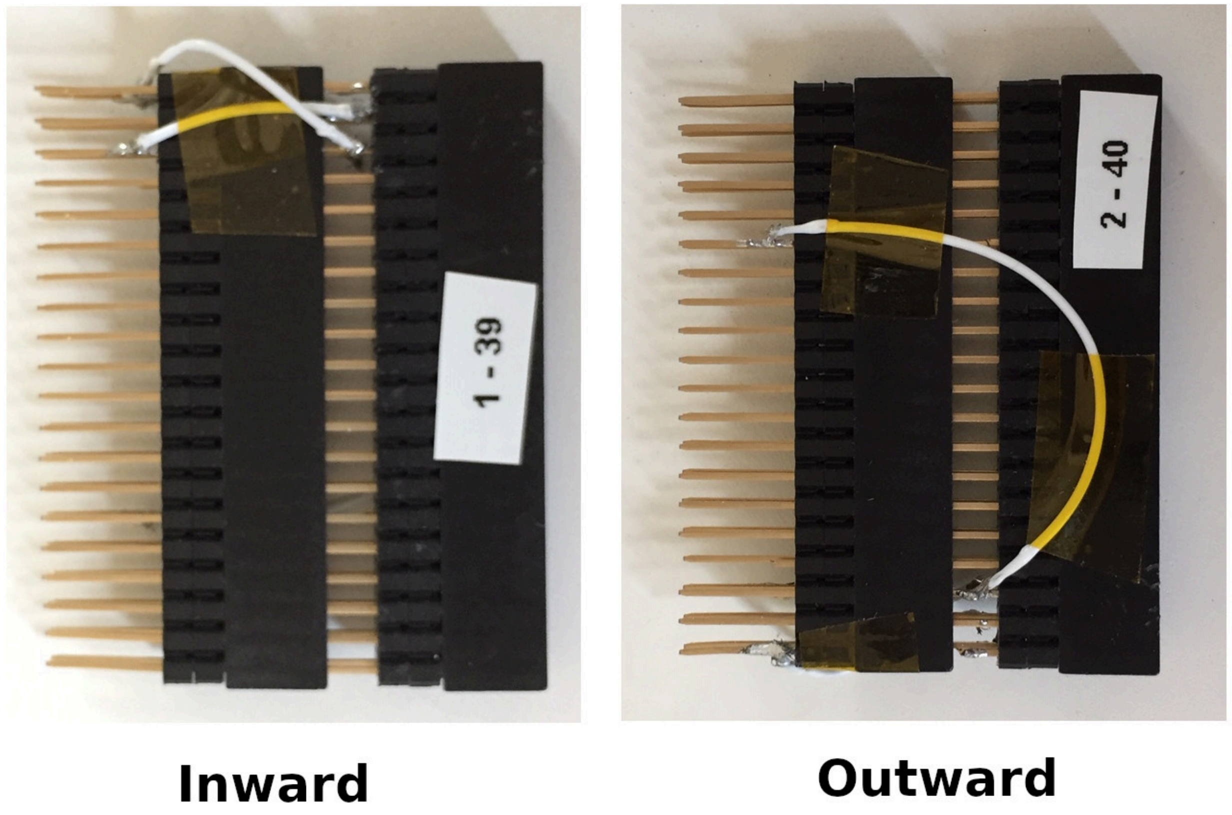

Figure 4. Hand-made transposer to connect HiFiBerry to EVK

The HiFiBerry DAC+ ADC Pro is an audio card designed for the Raspberry Pi but 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 and HiFiBerry pin connection

EVK	HiFiBerry	Function
2	2	5 V
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

The following figure shows a complete setup, with a handmade transposer to respect above pinout.

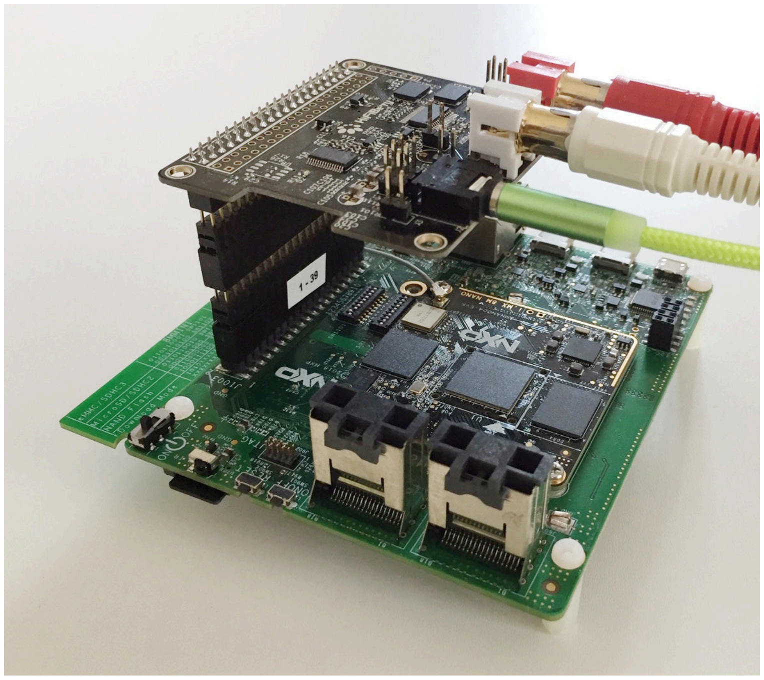

Figure 5. i.MX 8M EVK with Hifibery audio card

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**).

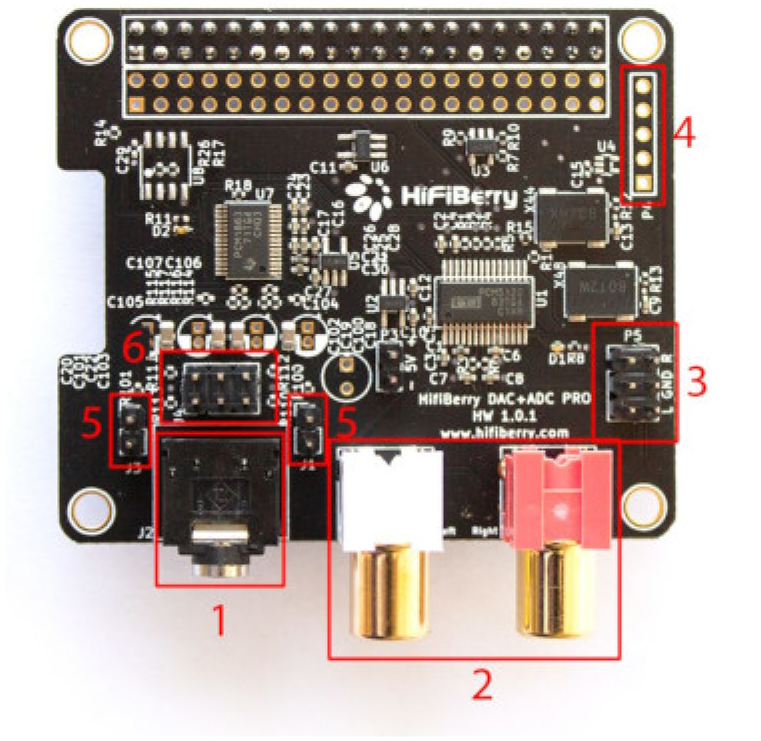

Figure 6. HiFiBerry audio connectors (picture from HiFiBerry's website)

- Control of the PCM1863 is done through I2C3, at address 0x4a.
- Control of the PCM5122 is done through I2C3, at address 0x4d.

The PCM1863 and PCM5122 use i.MX SAI5. The PCM5122 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, controlled by PCM5122 GPIOs.

The following diagram shows the HiFiBerry architecture.

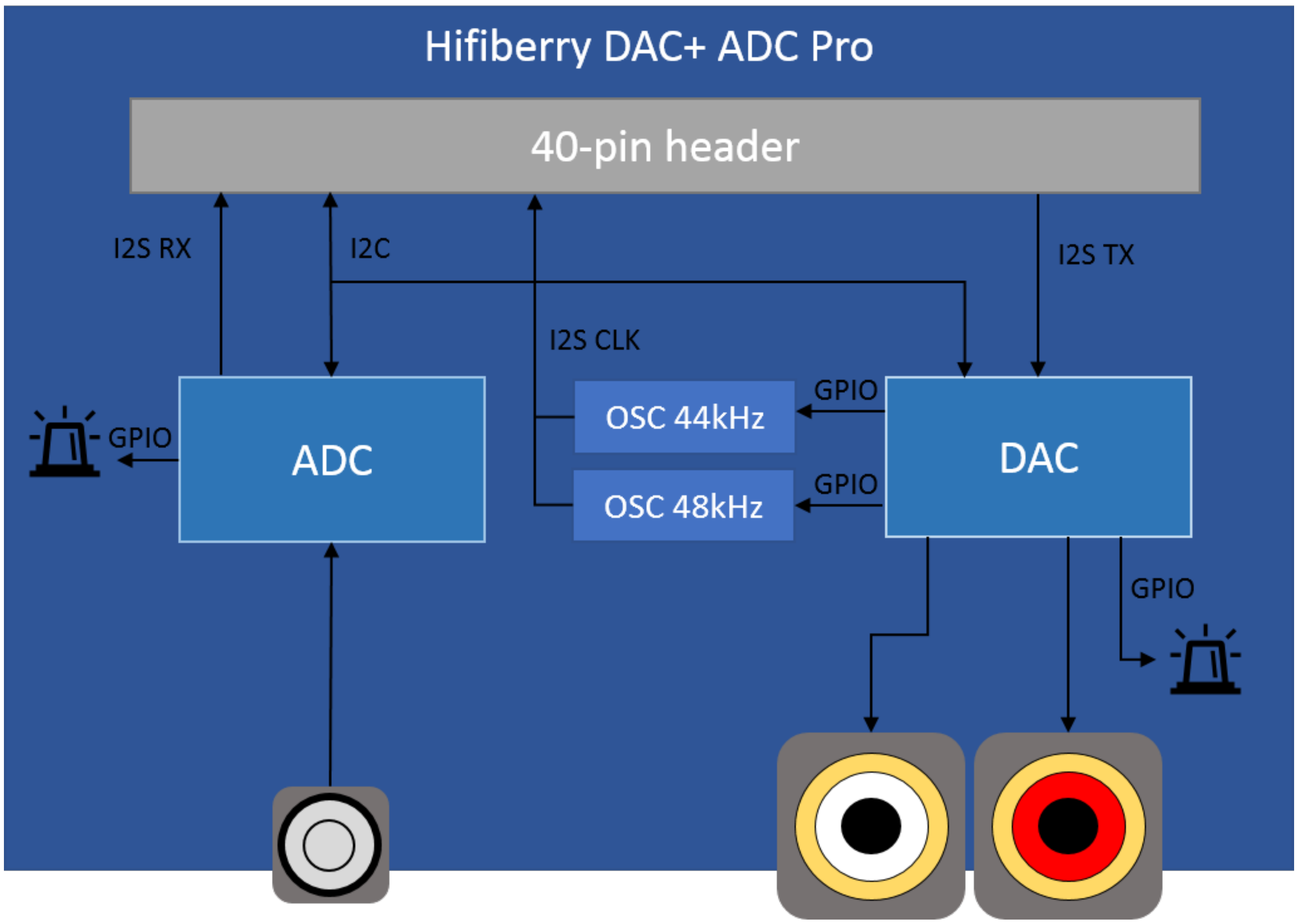

Figure 7. HiFiBerry architecture

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

Another way to play audio is through the EVK’s internal DAC by using the board’s jack connector.

4 Running the Demos

4.1 Basic setup

The EVK target exposes two serial ports through its USB debug interface.

One is used by Linux OS for its console, and the other is used by the guest RTOS Little Kernel.

To run the demos, open both serial port with terminal emulators, insert the micro-SD card on which the Yocto image has been flashed to the EVK and power up the board.

4.2 Starting Linux kernel

Linux kernel must be started with a 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 commands:

- For i.MX 8M Mini:

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

- For i.MX 8M Nano:

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

4.3 Starting Little Kernel

Log in to Linux OS by using the ‘root’ user name.

Then start the Little Kernel RTOS by using the helper script:

```
/usr/share/harpoon/scripts/jstart_lk_hifiberry.sh
```

The following Jailhouse trace is shown:

```
modprobe jailhouse
Enabling jailhouse root cell

Initializing Jailhouse hypervisor v0.12 (78-g6a4d89fc-dirty) on CPU 0
Code location: 0x0000ffffc0200800
Page pool usage after early setup: mem 39/992, remap 0/131072
Initializing processors:
 CPU 0... OK
 CPU 3... OK
 CPU 1... OK
 CPU 2... OK
Initializing unit: irqchip
Initializing unit: ARM SMMU v3
Initializing unit: ARM SMMU
No SMMU
Initializing unit: PVU IOMMU
Initializing unit: PCI
Adding virtual PCI device 00:02.0 to cell "imx8mm"
Adding virtual PCI device 00:03.0 to cell "imx8mm"
Adding virtual PCI device 00:00.0 to cell "imx8mm"
Adding virtual PCI device 00:01.0 to cell "imx8mm"
Page pool usage after late setup: mem 63/992, remap 144/131072
Activating hypervisor
[   15.522472] pci-host-generic bb800000.pci: host bridge /pci@0 ranges:
[   15.528977] pci-host-generic bb800000.pci:      MEM 0x00bb900000..0x00bb907fff -> 0x00bb900000
[   15.537707] pci-host-generic bb800000.pci: ECAM at [mem 0xbb800000-0xbb8fffff] for [bus 00]
[   15.546281] pci-host-generic bb800000.pci: PCI host bridge to bus 0001:00
[   15.553124] pci_bus 0001:00: root bus resource [bus 00]
[   15.558401] pci_bus 0001:00: root bus resource [mem 0xbb900000-0xbb907fff]
[   15.565336] pci 0001:00:00.0: [110a:4106] type 00 class 0xff0000
[   15.571389] pci 0001:00:00.0: reg 0x10: [mem 0x00000000-0x00000fff]
[   15.578164] pci 0001:00:01.0: [110a:4106] type 00 class 0xff0001
[   15.584367] pci 0001:00:01.0: reg 0x10: [mem 0x00000000-0x00000fff]
[   15.591273] pci 0001:00:02.0: [110a:4106] type 00 class 0xffc002
[   15.597564] pci 0001:00:02.0: reg 0x10: [mem 0x00000000-0x00000fff]
[   15.604570] pci 0001:00:03.0: [110a:4106] type 00 class 0xffc003
[   15.610780] pci 0001:00:03.0: reg 0x10: [mem 0x00000000-0x00000fff]
[   15.619636] pci 0001:00:00.0: BAR 0: assigned [mem 0xbb900000-0xbb900fff]
[   15.626519] pci 0001:00:01.0: BAR 0: assigned [mem 0xbb901000-0xbb901fff]
[   15.633364] pci 0001:00:02.0: BAR 0: assigned [mem 0xbb902000-0xbb902fff]
[   15.640183] pci 0001:00:03.0: BAR 0: assigned [mem 0xbb903000-0xbb903fff]
[   15.647320] ivshm 0001:00:00.0: enabling device (0000 -> 0002)
[   15.653237] ivshm 0001:00:00.0: Output section is defined, but currently not used by this version!
[   15.662288] ivshm 0001:00:00.0: name:register base: 0xbb900000 size: 0x1000
[   15.669275] ivshm 0001:00:00.0: name:shm base: 0xbbaf1000 size: 0x1ff000
[   15.676279] ivshm_pipe_thread: entry
[   15.676284] ivshm_pipe_init:931: IRQ: 0xc4
[   15.676292] ivshm_pipe_init:933: IO Mapping: 00000000c148a5e6
[   15.689752] ivshm_pipe_init:936: SHM Mapping: [0000000070b40eed-00000000efab8363]
[   15.697257] ivshm_pipe_init:937: IVpos: 0
[   15.704840] ivshmem-net 0001:00:01.0: enabling device (0000 -> 0002)
[   15.711351] ivshmem-net 0001:00:01.0: TX memory at 0x00000000bbd02000, size 0x000000000007f000
[   15.720006] ivshmem-net 0001:00:01.0: RX memory at 0x00000000bbd81000, size 0x000000000007f000
[   15.730373] ivshm 0001:00:02.0: enabling device (0000 -> 0002)
[   15.736571] ivshm 0001:00:02.0: Output section is defined, but currently not used by this version!
[   15.745671] ivshm 0001:00:02.0: name:register base: 0xbb902000 size: 0x1000
[   15.752713] ivshm 0001:00:02.0: name:shm base: 0xbba01000 size: 0xdf000
[   15.759696] ivshm_pipe_thread: entry
[   15.759702] ivshm_pipe_init:931: IRQ: 0xc6
[   15.759712] ivshm_pipe_init:933: IO Mapping: 00000000ef470369
[   15.773334] ivshm_pipe_init:936: SHM Mapping: [000000000c6bbcdb-000000001a0db116]
[   15.780872] ivshm_pipe_init:937: IVpos: 0
[   15.788873] ivshm 0001:00:03.0: enabling device (0000 -> 0002)
[   15.794847] ivshm 0001:00:03.0: Output section is defined, but currently not used by this version!
[   15.803884] ivshm 0001:00:03.0: name:register base: 0xbb903000 size: 0x1000
[   15.810883] ivshm 0001:00:03.0: name:shm base: 0xbbae1000 size: 0xf000
[   15.817645] ivshm_pipe_init:931: IRQ: 0xc7
[   15.817647] ivshm_pipe_thread: entry
[   15.825540] ivshm_pipe_init:933: IO Mapping: 0000000071713072
[   15.831334] ivshm_pipe_init:936: SHM Mapping: [0000000046756cf1-000000003f9052cc]
[   15.838865] ivshm_pipe_init:937: IVpos: 0
[   15.844664] The Jailhouse is opening.
Creating jailhouse lk cell
[   15.880755] IRQ 14: no longer affine to CPU0
[   15.881307] CPU0: shutdown
[   15.888320] psci: CPU0 killed (polled 0 ms)
[   15.925267] CPU1: shutdown
[   15.927987] psci: CPU1 killed (polled 0 ms)
Adding virtual PCI device 00:00.0 to cell "lk"
Shared memory connection established, peer cells:
 "imx8mm"
Created cell "lk"
Page pool usage after cell creation: mem 84/992, remap 144/131072
[   15.952298] Created Jailhouse cell "lk"
Loading jailhouse lk cell
Cell "lk" can be loaded
Starting lk cell
Started cell "lk"
[   16.166983] ivshm_pipe_fsm:715: Initialization IVSHM services...
[   16.173744] ivshm_endpoint_push:160: No endpoint found for id 2
[   16.180425] ivshm_endpoint_thread:386: buf rpmsg_console: [0000000039ad8706-00000000a60f70df]
```

In the terminal emulator connected to the other serial port, the Little Kernel trace is shown:

```
[00000.000000] idle 0 > appargs_init:1437: Valid Device Tree Blob found @ 0xffff000000000000
[00000.000000] idle 0 > detected GICv3
[00000.000000] idle 0 > Enabling cpu 1 returned 0
[00000.000000] idle 0 > Enabling cpu 2 returned fffffffd
[00000.000000] idle 0 > Enabling cpu 3 returned fffffffd
```

```
[00026.106650] idle 0 > Initialize memory arena: memsize 0x1FC00000
[00026.111126] idle 0 >
[00026.111127] idle 0 > welcome to lk/MP
[00026.111128] idle 0 >
[00026.111131] idle 0 > boot args 0x0 0x0 0x0 0x0
[00026.111135] idle 0 > INIT: cpu 0, calling hook 0xffff000000049e48 (version) at level 0x3ffff,
flags 0x1
[00026.111137] idle 0 > version:
[00026.111137] idle 0 >         arch:     ARM64
[00026.111138] idle 0 >         platform: IMX8
[00026.111139] idle 0 >         target:   IMX8MM_DTS
[00026.111140] idle 0 >         project:  IMX8MM_DTS
[00026.111141] idle 0 >         buildid:  K5B9K_LOCAL
[00026.111141] idle 0 >         LK version:
[00026.111142] idle 0 >           imx8 rev: F20A6FE
[00026.111143] idle 0 >           lk   rev: D2E6942F
[00026.111146] idle 0 > INIT: cpu 0, calling hook 0xffff00000004e628 (vm_preheap) at level 0x3ffff,
flags 0x1
[00026.111716] idle 0 > initializing heap
[00026.111740] idle 0 > calling constructors
[00026.111744] idle 0 > INIT: cpu 0, calling hook 0xffff00000004e680 (vm) at level 0x50000, flags 0x1
[00026.111753] idle 0 > INIT: cpu 0, calling hook 0xffff00000004aea0 (ivshmem) at level 0x50001,
flags 0x1
[00026.111924] idle 0 > IVSHMEM: ABI revision: 2
[00026.111926] idle 0 > IVSHMEM: VPCI region mapping: 1048576@bb800000
[00026.111927] idle 0 > IVSHMEM: IRQ base: 108
[00026.111980] idle 0 > Found 110a:4106 at 00:00.0
[00026.111985] idle 0 > IVSHMEM: bar0 is at 0xffffffffff000000/ff000000
[00026.111987] idle 0 > IVSHMEM: bar1 is at 0xffffffffff001000/ff001000
[00026.112033] idle 0 > IVSHMEM: ID is 1
[00026.112034] idle 0 > IVSHMEM: max. peers is 1
[00026.112039] idle 0 > IVSHMEM: state table: 4096@0xffffffffbbaf0000
[00026.112041] idle 0 > IVSHMEM: R/W section: 2093056@0xffffffffbbaf1000
[00026.112042] idle 0 > IVSHMEM: [NOT USED] input section: 4096@0xffffffffbbcf0000
[00026.112044] idle 0 > IVSHMEM: [NOT USED] output section: 4096@0xffffffffbbcf1000
[00026.112103] idle 0 > INIT: cpu 0, calling hook 0xffff00000004c2f0 (ivshm_pipe) at level 0x50002,
flags 0x1
[00026.112257] idle 0 > ivshm_pipe_init:932: IRQ: 0x6c
[00026.112259] idle 0 > ivshm_pipe_init:934: IO Mapping: 0xffffffffff000000
[00026.112261] idle 0 > ivshm_pipe_init:936: SHM Mapping: [0xffffffffbbaf1000-0xffffffffbbceffff]
[00026.112263] idle 0 > ivshm_pipe_init:938: IVpos: 1
[00026.112295] idle 0 > initializing mp
[00026.112295] idle 0 > initializing threads
[00026.112297] idle 0 > initializing timers
[00026.112297] idle 0 > initializing ports
[00026.112301] idle 0 > INIT: cpu 0, calling hook 0xffff000000042c10 (debuglog) at level 0x6ffff,
flags 0x1
[00026.112328] idle 0 > Debug log buffer size 131072
[00026.112329] idle 0 > creating bootstrap completion thread
[00026.112342] ivshm-pipe > ivshm_pipe_thread: entry
[00026.112350] bootstrap2 > top of bootstrap2()
[00026.112354] bootstrap2 > INIT: cpu 0, calling hook 0xffff000000043428 (libdpc) at level 0x70000,
flags 0x1
[00026.112401] bootstrap2 > creating bootstrap completion thread for cpu 1
[00026.112426] bootstrap2 > creating bootstrap completion thread for cpu 2
[00026.112431] bootstrap2 > creating bootstrap completion thread for cpu 3
[00026.112457] bootstrap2 > initializing platform
[00026.112460] bootstrap2 > INIT: cpu 0, calling hook 0xffff000000041720 (appargs) at level 0x8fffe,
flags 0x1
[00026.112475] unknown > INIT: cpu 1, calling hook 0xffff0000000263c8
```

```
(arm_generic_timer_init_secondary_cpu) at level 0x6ffff, flags 0x2
[00026.112505] idle 1 > entering scheduler on cpu 1
[00026.119456] ivshm-pipe > ivshm_pipe_fsm:715: Initialization IVSHM services...
[00026.119669] console > ivshm_endpoint_thread:386: buf
console: [0xffff0000006fe3f0-0xffff0000007003ef]
[00026.125966] bootstrap2 > INIT: cpu 0, calling hook 0xffff000000041720 (appargs_core) at level
0x8ffff, flags 0x1
[00026.125975] bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(1)...
[00026.126159] binary-514 > ivshm_endpoint_thread:386: buf
binary-514: [0xffff0000007033d0-0xffff0000007035cf]
[00026.126177] bootstrap2 > imx_gpio_init: entry
[00026.126212] bootstrap2 > imx_gpr_init: entry
[00026.126222] bootstrap2 > imx_gpr_init:102: 4 GPR register to initialize..
[00026.126226] bootstrap2 > imx_gpr_init:108: Set register 0xffffffff30340008 to 0
[00026.126230] bootstrap2 > imx_gpr_init:108: Set register 0xffffffff30340018 to 0
[00026.126234] bootstrap2 > imx_gpr_init:108: Set register 0xffffffff3034001c to 0
[00026.126237] bootstrap2 > imx_gpr_init:108: Set register 0xffffffff30340020 to f0f0000
[00026.126349] bootstrap2 > imx_sdma_init:677: sdma ccb: va: 0xffff000020000000 pa: 94309000
[00026.126375] bootstrap2 > imx_sdma_init:716: sdma bd0: va: 0xffff000020001000 pa: 9430b000
[00026.126401] bootstrap2 > imx_sdma_init:748: sdma context: va: 0xffff000020002000 pa: 9430c000
[00026.126433] bootstrap2 > fw version 4.8
[00026.126497] bootstrap2 > imx_sdma_init:801: Loaded firmware to SDMA 3 RAM
[00026.126506] bootstrap2 > initializing target
[00026.126512] bootstrap2 > INIT: cpu 0, calling hook 0xffff000000041720 (appargs_platform) at level
0x90000, flags 0x1
[00026.127251] bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(8)...
[00026.127489] bootstrap2 > pca6416_init:294: PCA6416 on i2c bus 3, IRQ 96
[00026.128982] bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(10)...
[00026.129031] bootstrap2 > PDM base address: 0xffffffff30080000
[00026.129035] bootstrap2 >     CTRL_1: 0x00000000
[00026.129038] bootstrap2 >     CTRL_2: 0x0c000000
[00026.129042] bootstrap2 >     FIFO_CTRL: 0x00000003
[00026.129045] bootstrap2 >     :DC_CTRL 0x00000000
[00026.129049] bootstrap2 >     :OUT_CTRL 0x00000000
[00026.129106] bootstrap2 > imx_sai_init:1389: mclk_tx_config ret code: 0
[00026.129125] bootstrap2 > imx_sai_init:1404: mclk_rx_config ret code: 0
[00026.129191] bootstrap2 > imx_sai_init:1460: SAI1 mclk is output : false
[00026.129219] bootstrap2 > imx_sai_set_mclk:985: SAI1: Using direct access to root clock mux
as fallback
[00026.129224] bootstrap2 > imx_sai_set_mclk:985: SAI1: Using direct access to root clock mux
as fallback
[00026.129240] bootstrap2 > imx_sai_init:1478: SAI1 RX configured for master mode...
[00026.129257] bootstrap2 > imx_sai_init:1482: SAI1 TX configured for master mode...
[00026.129274] bootstrap2 > of_gpio_request_by_name:234: Error while parsing handle and args for
gpio,mclk-is-extern node
[00026.129314] bootstrap2 > SAI1: TX max circ buf size: 1048576
[00026.129330] bootstrap2 > SAI1: RX max circ buf size: 524288
[00026.129402] bootstrap2 > imx_sai_init:1540: SAI1 rx synchronization mode set to 0
[00026.129406] bootstrap2 > imx_sai_init:1542: SAI1 tx synchronization mode set to 0
[00026.129432] bootstrap2 > imx_sai_init:1550: SAI1 RX FCOMB configuration set to 0.
[00026.129438] bootstrap2 > imx_sai_init:1552: SAI1 TX FCOMB configuration set to 0.
[00026.129462] bootstrap2 > imx_sai_init:1567: SAI1 driver running with DMA RX mode enabled (mode 2)
[00026.129466] bootstrap2 > imx_sai_init:1574: SAI1 driver running with DMA TX mode enabled (mode 2)
[00026.129617] bootstrap2 > imx_sai_init:1389: mclk_tx_config ret code: 0
[00026.129636] bootstrap2 > imx_sai_init:1404: mclk_rx_config ret code: 0
[00026.129710] bootstrap2 > imx_sai_init:1460: SAI3 mclk is output : true
[00026.129742] bootstrap2 > imx_sai_set_mclk:985: SAI3: Using direct access to root clock mux
as fallback
[00026.129749] bootstrap2 > imx_sai_set_mclk:985: SAI3: Using direct access to root clock mux
as fallback
```

```
[00026.129767] bootstrap2 > imx_sai_init:1478: SAI3 RX configured for master mode...
[00026.129786] bootstrap2 > imx_sai_init:1482: SAI3 TX configured for master mode...
[00026.129805] bootstrap2 > of_gpio_request_by_name:234: Error while parsing handle and args for
gpio,mclk-is-extern node
[00026.129852] bootstrap2 > SAI3: TX max circ buf size: 1048576
[00026.129869] bootstrap2 > SAI3: RX max circ buf size: 524288
[00026.129955] bootstrap2 > imx_sai_init:1540: SAI3 rx synchronization mode set to 0
[00026.129960] bootstrap2 > imx_sai_init:1542: SAI3 tx synchronization mode set to 0
[00026.129992] bootstrap2 > imx_sai_init:1550: SAI3 RX FCOMB configuration set to 0.
[00026.129998] bootstrap2 > imx_sai_init:1552: SAI3 TX FCOMB configuration set to 0.
[00026.130019] bootstrap2 > imx_sai_init:1567: SAI3 driver running with DMA RX mode disabled (mode 2)
[00026.130025] bootstrap2 > imx_sai_init:1574: SAI3 driver running with DMA TX mode disabled (mode 2)
[00026.130099] bootstrap2 > imx_sai_init:1389: mclk_tx_config ret code: 0
[00026.130122] bootstrap2 > imx_sai_init:1404: mclk_rx_config ret code: 0
[00026.130212] bootstrap2 > imx_sai_init:1460: SAI5 mclk is output : true
[00026.130242] bootstrap2 > imx_sai_set_mclk:985: SAI5: Using direct access to root clock mux
as fallback
[00026.130249] bootstrap2 > imx_sai_set_mclk:985: SAI5: Using direct access to root clock mux
as fallback
[00026.130271] bootstrap2 > imx_sai_init:1478: SAI5 RX configured for slave mode...
[00026.130294] bootstrap2 > imx_sai_init:1482: SAI5 TX configured for slave mode...
[00026.130316] bootstrap2 > of_gpio_request_by_name:234: Error while parsing handle and args for
gpio,mclk-is-extern node
[00026.130371] bootstrap2 > SAI5: TX max circ buf size: 1048576
[00026.130392] bootstrap2 > SAI5: RX max circ buf size: 524288
[00026.130491] bootstrap2 > imx_sai_init:1540: SAI5 rx synchronization mode set to 1
[00026.130496] bootstrap2 > imx_sai_init:1542: SAI5 tx synchronization mode set to 0
[00026.130535] bootstrap2 > imx_sai_init:1550: SAI5 RX FCOMB configuration set to 0.
[00026.130540] bootstrap2 > imx_sai_init:1552: SAI5 TX FCOMB configuration set to 0.
[00026.130561] bootstrap2 > imx_sai_init:1567: SAI5 driver running with DMA RX mode disabled (mode 2)
[00026.130567] bootstrap2 > imx_sai_init:1574: SAI5 driver running with DMA TX mode disabled (mode 2)
[00026.130622] bootstrap2 > imx_spdif_init:1091: SPDIF driver running with DMA mode enabled.
[00026.130648] bootstrap2 > imx_spdif_init:1111: Using 4 period buffers of length 2048 bytes
[00026.130663] bootstrap2 > imx_spdif_init:1120: SPDIF1 driver running with DMA RX mode enabled
(mode 2)
[00026.130696] bootstrap2 > imx_spdif_init:1132: Allocating memory once during initialization
[00026.130732] bootstrap2 > rx circ buf (size / address) : 1048576bytes / 0x0xffff000000712d80
[00026.130750] bootstrap2 > SPDIF base address: 0xffffffff30090000
[00026.130753] bootstrap2 >     SCR: 0x170437
[00026.130756] bootstrap2 >     SRCD: 0x0
[00026.130760] bootstrap2 >     SRPC: 0x98
[00026.130761] bootstrap2 >     SIE: 0x0
[00026.130766] bootstrap2 >     SIS: 0x2
[00026.130770] bootstrap2 >     SRFM: 0x96d1
[00026.130771] bootstrap2 >     STC: 0x20900
[00026.130782] bootstrap2 > imx_uart_init:537: UART: started IRQ driven TX
[00026.130786] bootstrap2 > imx_uart_init:548: Using TX buffered mode
[00026.130792] bootstrap2 > calling apps_init()
[00026.130797] bootstrap2 > INIT: cpu 0, calling hook 0xffff000000041720 (appargs_target) at level
0xa0000, flags 0x1
[00026.130801] bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(100)...
[00026.131402] bootstrap2 > INIT: cpu 0, calling hook 0xffff000000041720 (appargs_hal) at level
0xaffff, flags 0x1
[00026.131405] bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(1000)...
[00026.133209] bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(4000)...
[00026.133218] bootstrap2 > starting app shell
[00026.133253] shell > entering main console loop
[00026.133262] bootstrap2 > INIT: cpu 0, calling hook 0xffff000000041720 (appargs_app) at level
```

```
0xb0000, flags 0x1
[00026.133268] bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(10000)...
```

NOTE

If Little Kernel starts on an EVK on which the HiFiBerry card has not been connected, the I2C driver asserts when trying to probe DAC/ADC elements of the audio card. To run Little Kernel on a HiFiBerry-less EVK, run the following command instead:

```
/usr/share/harpoon/scripts/jstart_lk.sh
```

This runs Little Kernel with a device tree that does not contain the definition of the HiFiBerry card.

After booting, Little Kernel displays a prompt and waits for applications to be started. A list of available commands is shown using the following command:

```
] help
```

4.4 Playing sound with the HiFiBerry

Simple tones can be generated and played by the HiFiBerry card. Connect a pair of earphones to the audio card's RCA connectors and use the **HiFiBerry** application from the Little Kernel's prompt.

Different tones can be played at different sampling frequencies.

To generate a basic sinewave:

```
] hifiberry play sine <sample_frequency>
```

Where *sample_frequency* is one of 44100, 48000, 96000, and 192000. This parameter is optional, and the default value is 44100.

The higher the sampling frequency, the higher pitched the sinewave will sound.

To generate a basic saw or square wave, following the same syntax as the sinewave:

```
] hifiberry play saw <sample_frequency> ] hifiberry play square <sample_frequency>
```

To generate a DTMF sequence:

```
hifiberry dtmf <sample_frequency> <dtmf_string_left> <dtmf_string_right>
```

All parameters are optional; *sample_frequency* defaults to 44100, *dtmf_string_left* to 1123ABCD0123456789*#, and *dtmf_string_right* to #*9876543210DCBA3211.

The *dmtf_string* parameters are strings for left and right audio channels that correspond to DMTF key tone sequences. When present, both left and right strings must have the same length.

4.5 Recording sound with the HiFiBerry

For the sake of simplicity, the record functionality in the **HiFiBerry** Little Kernel application is coupled to a playback functionality.

Connect an audio source to the HiFiBerry's input jack and a pair of headphones to the RCA audio connectors.

Run the following command:

```
hifiberry record <sample_frequency>
```

Here again, the *sample_frequency* can be one of 44100, 48000, 96000, and 192000. It is optional and defaults to 44100.

When running this command, a thread reads audio data from the ADC and another feeds the DAC with these data.

5 Technical Details

5.1 Hardware partitioning

Jailhouse hypervisor is used to run Little Kernel RTOS in parallel with Linux.

Jailhouse is a simple hypervisor that assigns hardware resources to a guest OS instead of virtualising them. For instance, a CPU core statically belongs to an OS while the other is not shared.

In Jailhouse terms, Little Kernel belongs to a cell. A configuration file describes which hardware resources are assigned to this cell. This configuration file contains descriptions as follows:

- CPU cores assigned to the cell
- Interrupt lines assigned to the cell
- Memory regions assigned to the cell
- Virtual PCI device used for communication between Linux OS and the RTOS

There is also a root cell configuration, which describes the hardware prior to the hardware partitioning.

The source files of the cell configurations are in the imx-jailhouse source code, at the following locations:

- `configs/arm64/imx8mm-lk.c` for the Little Kernel's cell configuration
- `configs/arm64/imx8mm.c` for the root cell configuration

The CPU cores allocated to Little Kernel forms a bitmap in the "*cpu*" structure. Here, CPU cores 0 and 1 are assigned to Little Kernel:

```
.cpus = { 0b0011, }
```

Memory regions assigned to Little Kernel are listed in the "*mem_regions*" structure. Memory regions can be reserved for Little Kernel or shared with Linux OS.

Memory regions can be DDR chunks for RTOS use as well as device memory mapped region such as SAI.

Interrupts are mapped to Little kernel with the "*irqchips*" structure.

Virtual PCI device is defined with the "*pci_devices*" structure. This virtual device is used by Jailhouse to implement an IVSHMEM v2 communication channel.

5.2 Architecture

The architecture of the Harpoon solution is as follows.

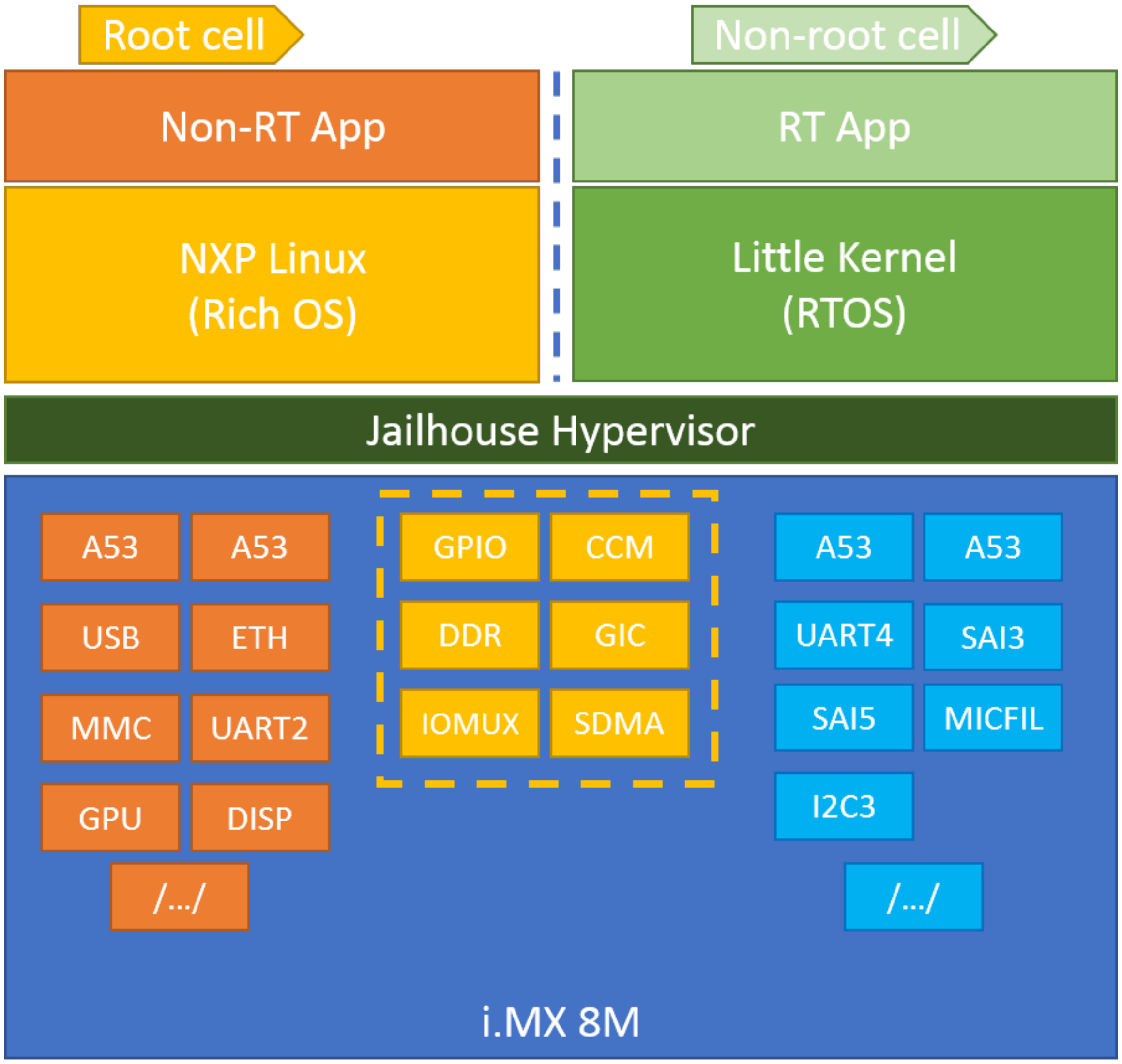

Figure 8. Harpoon solution architecture

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

5.3 Little Kernel

5.3.1 Manual build

During development on Little Kernel side, it may be easier to compile the RTOS manually rather that build a whole Yocto image. This section provides the steps needed to build Little Kernel manually.

Little Kernel must be compiled with a compatible toolchain.

Reference toolchain is the 7.3.1 toolchain from Linaro.

To download the toolchain and install it:

```
wget https://releases.linaro.org/components/toolchain/binaries/7.3-2018.05/aarch64-elf/gcc-
linaro-7.3.1-2018.05-x86_64_aarch64-elf.tar.xz
tar -C /opt/ -xvJf gcc-linaro-7.3.1-2018.05-x86_64_aarch64-elf.tar.xz
```

Get the source code for Little kernel:

```
mkdir lk
cd lk
git clone https://github.com/NXPmicro/littlekernel-lk.git lk
git clone https://github.com/NXPmicro/littlekernel-imx8.git imx8
```

Then build Little Kernel:

```
cd imx8
PROJECT=<project> make -j$(nproc)
```

Where, `PROJECT` is `imx8mm-dts` for i.MX 8M Mini and `imx8mn-dts` for i.MX 8M Nano.

If the toolchain has not been installed to the `/opt` default location, environment variable `ARCH_arm64_TOOLCHAIN_PREFIX` can be set like this:

```
export ARCH_arm64_TOOLCHAIN_PREFIX=/path/to/gcc-linaro-$TOOLCHAIN_VERSION-x86_64_aarch64-
elf/bin/aarch64-elf-
```

Builds artefacts are available in directory `build-<PROJECT>`.

Artefacts to be used on target are as follows:

- `lk.bin`: Little Kernel binary
- `imx8m{m,n}-evk.dtb`: device tree for EVK without HiFiBerry sound card
- `imx8mn-evk-hifiberry-dacplus.dtb`: device tree for EVK with HiFiBerry sound card

These files must be copied to the rootfs partition of the SD card, in directory `/home/root/`.

5.3.2 Device tree

Little Kernel uses a device tree similar to those used by Linux OS.

Two device trees per SoC have been written for this release. One defines the EVK alone, and the other one the EVK is connected to a HiFiBerry sound card.

Device tree sources are in repository littlekernel-imx8, in directory `target/<soc>/`:

```
target/imx8mm/dts/imx8mm-evk-hifiberry-dacplus.dts
target/imx8mm/dts/imx8mm-evk.dts
target/imx8mn/dts/imx8mn-evk-hifiberry-dacplus.dts
target/imx8mn/dts/imx8mn-evk.dts
```

The HiFiBerry card is defined by its ADC and DAC in the device tree node i2c3:

```
&i2c3 {
/.../
    pcm512x: pcm512x@4d {
        compatible = "dac_pcm512x";

        reg = < 0x4d >;
        reg-names = "core";
        bus-id-i2c = < 3 >;
        bus-id = < 51 >;
        gpio-led = < 4 >;
        gpio-osc44 = < 6 >;
        gpio-osc48 = < 3 >;

        status = "ok";
    };

    pcm186x: pcm186x@4a {
        compatible = "adc_pcm186x";

        reg = < 0x4a >;
        reg-names = "core";
        bus-id-i2c = < 3 >;
        bus-id = < 18 >;
        gpio-led = < 2 >;

        status = "ok";
    };
};
```

Parameter *bus-id* gives a unique ID that is used by the Little Kernel application to reference the ADC/DAC through the `class_dac_get_device_by_id() / class_adc_get_device_by_id()` functions.

The ADC and the DAC have GPIO controllers, which their driver uses for LED and oscillator controls. The GPIO numbers are stored in parameters `gpio-led`, `gpio-osc44`, and `gpio-osc48`.

SAI synchronous mode, used for the same word select signal for both RX and TX, is enabled in device tree node sai5. In the same node, we indicate that the HiFiBerry generates the I2S clock:

```
&sai5 { /…/ /* RX synchronised to TX */ rx,sync-mode = < 1 >; /* Hifiberry card is master */
rx,slave_mode; tx,slave_mode; };
```

5.3.3 Starting Little Kernel with Jailhouse

To start Little Kernel, Linux module jailhouse must be loaded, Little Kernel’s cell must be created, and the RTOS payload must be loaded.

This is done with the following commands:

```
modprobe jailhouse
jailhouse enable <rootcell>
jailhouse cell create <lkcell>
jailhouse cell load lk <dtb> -a 0x80000000 lk.bin -a 0x80010000
jailhouse cell start lk
```

Where:

- `rootcell` is one of `imx8mm.cell` and `imx8mn.cell`
- `lkcell` is one of `imx8mm-lk.cell` and `imx8mn-lk.cell`
- `dtb` is one of `imx8mm-evk.dtb`, `imx8mm-evk-hifiberry-dacplus.dtb`, `imx8mn-evk.dtb`, and `imx8mn-evk-hifiberry-dacplus.dtb`.

A shell script template doing this is available in repository littlekernel-imx8: `scripts/jstart.sh.template`.

5.3.4 Developing a Little Kernel application

5.3.4.1 Architecture of “hifiberry” application

Application “hifiberry”, which serves as an example for this section, has the following architecture.

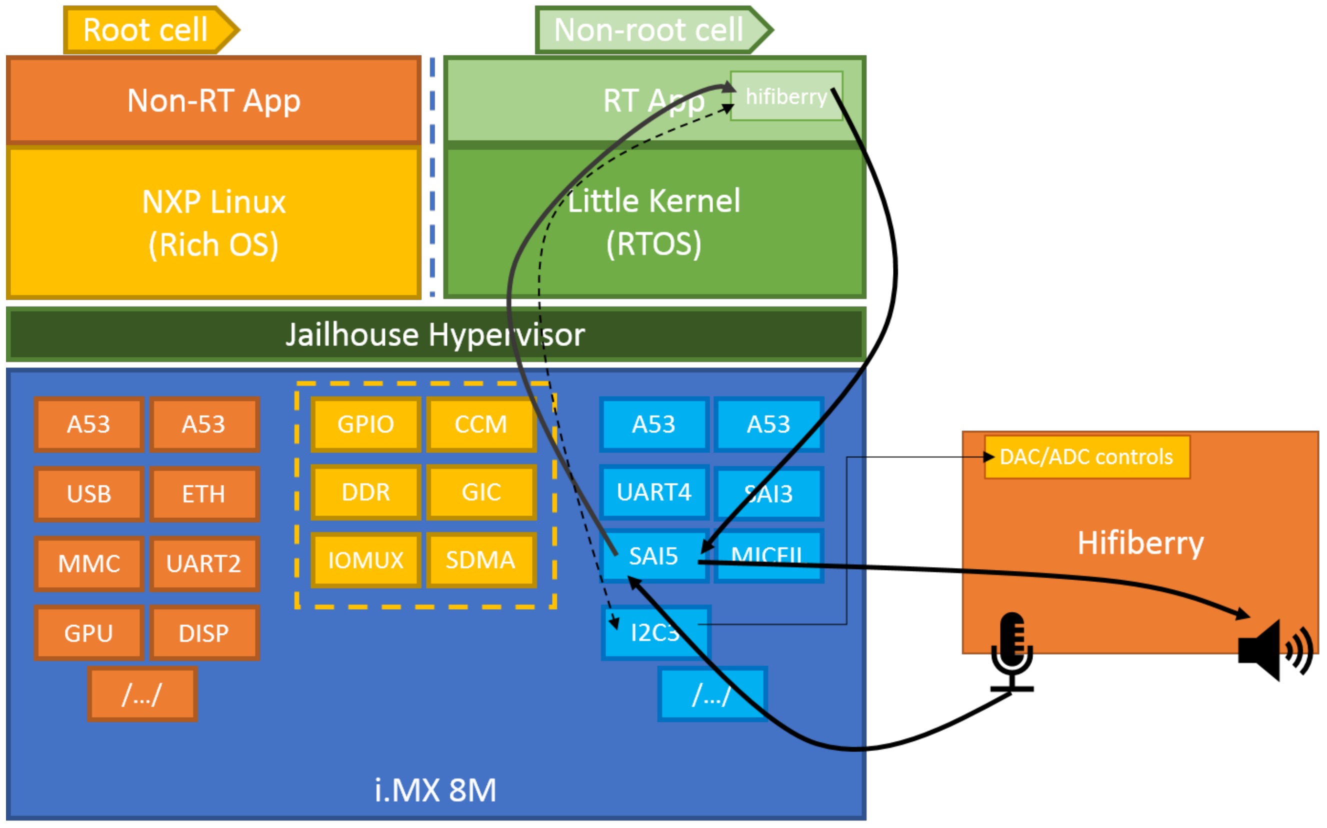

Figure 9. Architecture of “hifiberry” application

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

5.3.4.2 Source file creation

This section provides the information on how to develop an application for Little Kernel by using application “hifiberry” as an example.

First, the application directory must be created in directory `app/` of repository `littlekernel-imx8`.

This directory contains at least two files: the source code for the application, and a makefile `rules.mk`, which lists the source files:

```
app/hifiberry/
├── hifiberry.c
└── rules.mk
```

The source code starts with the following lines:

```
STATIC_COMMAND_START
STATIC_COMMAND("hifiberry", "Play sound on external Hifiberry card", &cmd_hifiberry)
STATIC_COMMAND_END(hifiberry);
```

This gives the name of the command to type at Little Kernel’s prompt to run the application (“hifiberry”), the help text associated with this application and entry point of the application (“cmd_hifiberry()”).

The source code ends with these lines:

```
APP_START(hifiberry)
    .flags = 0,
APP_END
```

5.3.4.3 Parsing application arguments

The prototype of the application's main function is:

```
static int cmd_hifiberry(int argc, const cmd_args *argv);
```

A variable number of arguments (value of `argc`) can be passed to the application.

Each argument can be interpreted as a string (`argv[i].str`), as an unsigned value (`argv[i].u`), as a pointer (`argv[i].p`), as a signed value (`argv[i].i`), or as a boolean (`argv[i].b`).

5.3.4.4 Opening and using SAI

To use SAI (for RX, TX, or both), a reference to the SAI IP must be obtained.

For instance, HiFiBerry uses SAI5, so application "hifiberry" gets the SAI reference with the following:

```
/* get SAI5 dev by bus id */
sai_dev = class_sai_get_device_by_id(APP_SAI5_BUSID);
```

This reference is used for further SAI operations.

To open SAI TX class:

```
ret = class_sai_open(sai_dev, false /* is_rx */);
```

To open SAI RX class:

```
ret = class_sai_open(sai_dev, true /* is_rx */);
```

A callback can be registered to a SAI class, which is especially useful for TX:

```
/* setup the callback for tx */
ret = class_sai_set_callback(sai_dev, dac_sai_cb, NULL, false);
```

Once the SAI class is open, its characteristics are set by sending structure `sai_format_t` to function `class_sai_setup()`. This structure defines parameters like the sample rate, the number of channels, sample width, the audio buffer size…

```
/* SAI TX setup */
sai_tx_fmt.sai_protocol = 2; /* I2S */
sai_tx_fmt.sampleRate_Hz = sample_rate;
sai_tx_fmt.bitWidth = REC_AUDIO_BITWIDTH;
sai_tx_fmt.num_channels = 2; /* stereo */
sai_tx_fmt.num_slots = 2;
sai_tx_fmt.period_size = REC_PERIOD_SIZE; /* frames per period */
sai_tx_fmt.polarity = SAI_BITCLOCK_POLARITY_ACTIVE_LOW;
sai_tx_fmt.master_slave = ksai_Slave;
sai_tx_fmt.bitclock_source = kSAI_BClkSourceMclkDiv;
ret = class_sai_setup(sai_dev, false, &sai_tx_fmt);
```

The class can then be started, and audio buffers can be read or written:

```
/* start SAI RX */
class_sai_start(sai_dev, true);

/* start SAI TX */
class_sai_start(sai_dev, false);

/* read an audio buffer from ADC */
class_sai_read(sai_dev, audio_buf, REC_AUDIO_PERIOD_BYTES);

/* send an audio buffer to DAC */
class_sai_write(sai_dev, audio_buf, REC_AUDIO_PERIOD_BYTES);
```

NOTE

The `class_sai_read()` function is blocking: It will exit when all requested samples have been read. On the other hand, `class_sai_write()` exits immediately: The callback initialized earlier is called with event `SAI_EVENT_PERIOD_ELAPSED` when the audio has effectively been through the SAI TX channel.

5.3.4.5 Opening and using a DAC

To use a DAC, a reference to it must be obtained.

For instance, the HiFiBerry's PCM5122 has been declared with bus-id 51 in the device tree. So the reference is obtained this way:

```
/* get the PCM512x dev by bus id */
pcm512x_dev = class_dac_get_device_by_id(51);
```

The DAC class is then opened:

```
/* pcm512x setup */
ret = class_dac_open(pcm512x_dev);
```

Once the DAC class is open, its characteristics are set by sending structure `dac_audio_hw_params_t` to function `class_dac_set_format()`. This structure defines parameters like the sample rate, the number of channels, sample format…

```
pcm512x_params.pcm_fmt = DAC_AUDIO_PCM_FMT_32;
pcm512x_params.fmt = DAC_AUDIO_FMT_I2S;
pcm512x_params.pkt = DAC_AUDIO_PKT_PCM;
pcm512x_params.num_ch = 2;
pcm512x_params.num_slots = 2;
pcm512x_params.rate = sample_rate;
ret = class_dac_set_format(pcm512x_dev, &pcm512x_params);
```

5.3.4.6 Opening and using an ADC

To use an ADC, a reference to it must be obtained.

For instance, the HiFiBerry's PCM1863 has been declared with bus-id 18 in the device tree. So the reference is obtained this way:

```
/* get the PCM186x dev by bus id */
pcm186x_dev = class_adc_get_device_by_id(18);
```

The ADC class is then opened:

```
/* pcm186x setup */
ret = class_adc_open(pcm186x_dev);
```

Once the ADC class is open, its characteristics are set by sending structure `adc_audio_hw_params_t` to function `class_adc_set_format()`. This structure defines parameters like the sample rate, the number of channels, sample format…

```
adc_params.pcm_fmt = DAC_AUDIO_PCM_FMT_32;
adc_params.fmt = DAC_AUDIO_FMT_I2S;
adc_params.pkt = DAC_AUDIO_PKT_PCM;
adc_params.num_ch = 2;
adc_params.rate = sample_rate;
ret = class_adc_set_format(pcm186x_dev, &adc_params);
```

6 Revision History

Table 2. Revision history

Revision number	Date	Substantive changes
LF5.10.35_2.0.0	06/2021	Initial release of Linux Harpoon.
LF5.10.52_2.1.0	09/2021	Moved Harpoon artifacts to the `/usr/share/harpoon` directory, and added the Section "Starting Linux kernel".
LF5.10.72_2.2.0	12/2021	Minor updates for the LF5.10.72_2.2.0 release.

Date of release: 17 December 2021

Document identifier: IMXHPUG