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:

```bash
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:

```bash
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:

```bash
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.

![imx8mm-lpddr4-evk for i.MX 8M Mini EVK board](image1)

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

![imx8mm-lpddr4-evk for i.MX 8M Nano LPDDR4 EVK board](image2)

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

![HiFiBerry DAC+ ADC Pro](image3)

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

<table>
<thead>
<tr>
<th>EVK</th>
<th>HiFiBerry</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>5 V</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>I2C SDA</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>I2C SCK</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>GND</td>
</tr>
<tr>
<td>35</td>
<td>40</td>
<td>I2S TX</td>
</tr>
<tr>
<td>36</td>
<td>12</td>
<td>I2S clock</td>
</tr>
<tr>
<td>37</td>
<td>35</td>
<td>I2S word select for RX and TX</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>I2S RX</td>
</tr>
</tbody>
</table>
The following figure shows a complete setup, with a handmade transposer to respect above pinout.

![Image of i.MX 8M EVK with Hifibery audio card](image)

**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).
• 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.
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:

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

For i.MX 8M Nano:

```bash
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:

```bash
modprobe jailhouse
Enabling jailhouse root cell

Initializing Jailhouse hypervisor v0.12 (78-g6a4d89fc-dirty) on CPU 0
Code location: 0x00000fff00200800
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
[ 15.551242] 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 0xff0000
[ 15.584367] pci 0001:00:01.0: reg 0x10: [mem 0x00000000-0x00000fff]
[ 15.591273] pci 0001:00:02.0: [110a:4106] type 00 class 0xff0000
[ 15.597564] pci 0001:00:02.0: reg 0x10: [mem 0x00000000-0x00000fff]
[ 15.604570] pci 0001:00:03.0: [110a:4106] type 00 class 0xff0000
[ 15.610780] pci 0001:00:03.0: reg 0x10: [mem 0x00000000-0x00000fff]
[ 15.619636] pci 0001:00:00.0: BAR 0: assigned [mem 0xbb900000-0xbb900000]
[ 15.626519] pci 0001:00:01.0: BAR 0: assigned [mem 0xbb901000-0xbb901000]
```
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
```

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

```
[ 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]
```
[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:
[00026.111138] idle 0 >         arch:     ARM64
[00026.111139] idle 0 >         platform: IMX8
[00026.111140] idle 0 >         project:  IMX8MM_DTS
[00026.111141] idle 0 >         target:   IMX8MM_DTS
[00026.111141] idle 0 >         buildid:  K5B9K_LOCAL
[00026.111141] idle 0 >         LK version:
[00026.111142] idle 0 >         imx8 rev: D2E6942F
[00026.111143] idle 0 >         lk   rev: D2E6942F
[00026.111146] idle 0 > INIT: cpu 0, calling hook 0xffff00000004e680 (vm) at level 0x50000, flags 0x1
[00026.111164] idle 0 > INIT: cpu 0, calling hook 0xffff00000004e628 (vm_preheap) at level 0x3ffff, flags 0x1
[00026.111176] idle 0 > initializing heap
[00026.111174] idle 0 > calling constructors
[00026.111177] idle 0 > INIT: cpu 0, calling hook 0xffff00000004e680 (vm) at level 0x50000, flags 0x1
[00026.111175] idle 0 > INIT: cpu 0, calling hook 0xffff00000004e680 (vm_preheap) at level 0x3ffff, flags 0x1
[00026.111174] idle 0 > INIT: cpu 0, calling hook 0xffff0000000462 (vm_preheap) at level 0x3fff,
flags 0x1
[00026.111192] idle 0 > IVSHMEM: ABI revision: 2
[00026.111192] idle 0 > IVSHMEM: VPCI region mapping: 1048576@bb800000
[00026.111192] idle 0 > IVSHMEM: IRQ base: 108
[00026.111192] idle 0 > Found i10a:4106 at 00:00.0
[00026.111192] idle 0 > IVSHMEM: bar0 is at 0xfffffffff00000000/ff000000
[00026.111192] idle 0 > IVSHMEM: bar1 is at 0xfffffffff00100000/ff001000
[00026.111203] idle 0 > IVSHMEM: ID is 1
[00026.111203] idle 0 > IVSHMEM: max. peers is 1
[00026.111203] idle 0 > IVSHMEM: state table: 4096@0xffffffffbbaf0000
[00026.111203] idle 0 > IVSHMEM: R/W section: 2093056@0xffffffffbbaf1000
[00026.111203] idle 0 > IVSHMEM: [NOT USED] input section: 4096@0xffffffffbbaf0000
[00026.111203] idle 0 > IVSHMEM: [NOT USED] output section: 4096@0xffffffffbbaf1000
[00026.1112103] idle 0 > INIT: cpu 0, calling hook 0xffff00000004c2f0 (ivshm_pipe) at level 0x50000, flags 0x1
[00026.112257] idle 0 > ivshm_pipe_init:932: IRQ: 0x6c
[00026.112257] idle 0 > ivshm_pipe_init:934: IO Mapping: 0xfffffffff0000000
[00026.112257] idle 0 > ivshm_pipe_init:936: SHM Mapping: [0xffffffffbaf1000-0xffffffffbceffff]
[00026.112257] 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
[00026.112300] idle 0 > initializing ports
[00026.112301] idle 0 > INIT: cpu 0, calling hook 0xffff00000004c2f0 (ivshm_pipe) at level 0x50002, flags 0x1
[00026.112258] idle 0 > ivshm_pipe_init:932: IRQ: 0x6c
[00026.112529] idle 0 > ivshm_pipe_init:934: IO Mapping: 0xfffffffff0000000
[00026.112621] idle 0 > ivshm_pipe_init:936: SHM Mapping: [0xffffffffbaf1000-0xffffffffbceffff]
[00026.112623] 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 0xffff00000004c2f0 (ivshm_pipe) at level 0x50002, 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 0x8ffe, flags 0x1
[00026.112475] unknown > INIT: cpu 1, calling hook 0xffff0000000263c8
idle 1 > entering scheduler on cpu 1

console: [0xffff0000006fe3f0-0xffff0000007003ef]

bootstrap2 > INIT: cpu 0, calling hook 0xffff000000041720 (appargs_core) at level

bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(1)...

bootstrap2 > imx_gpio_init: entry

bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(8)...

bootstrap2 > PDM base address: 0xffffffff30080000

bootstrap2 > SAI1: Using direct access to root clock mux as fallback

bootstrap2 > SAI1: Using direct access to root clock mux as fallback

bootstrap2 > SAI1: Using direct access to root clock mux as fallback

bootstrap2 > of_gpio_request_by_name:234: Error while parsing handle and args for
gpio,mclk-is-extern node

bootstrap2 > SAI1: Using direct access to root clock mux as fallback

bootstrap2 > SAI1: Using direct access to root clock mux as fallback

bootstrap2 > SAI1: Using direct access to root clock mux as fallback
bootstrap2 > imx_sai_init:1478: SAI3 RX configured for master mode...
bootstrap2 > imx_sai_init:1482: SAI3 TX configured for master mode...
bootstrap2 > of_gpio_request_by_name:234: Error while parsing handle and args for gpio,mclk-is-extern node
bootstrap2 > SAI3: TX max circ buf size: 1048576
bootstrap2 > SAI3: RX max circ buf size: 524288
bootstrap2 > imx_sai_init:1540: SAI3 rx synchronization mode set to 0
bootstrap2 > imx_sai_init:1542: SAI3 tx synchronization mode set to 0
bootstrap2 > imx_sai_init:1550: SAI3 RX FCOMB configuration set to 0.
bootstrap2 > imx_sai_init:1552: SAI3 TX FCOMB configuration set to 0.
bootstrap2 > imx_sai_init:1567: SAI3 driver running with DMA RX mode disabled (mode 2)
bootstrap2 > imx_sai_init:1574: SAI3 driver running with DMA TX mode disabled (mode 2)
bootstrap2 > of_gpio_request_by_name:234: Error while parsing handle and args for gpio,mclk-is-extern node
bootstrap2 > SAI5: TX max circ buf size: 1048576
bootstrap2 > SAI5: RX max circ buf size: 524288
bootstrap2 > imx_sai_init:1540: SAI5 rx synchronization mode set to 1
bootstrap2 > imx_sai_init:1542: SAI5 tx synchronization mode set to 0
bootstrap2 > imx_sai_init:1550: SAI5 RX FCOMB configuration set to 0.
bootstrap2 > imx_sai_init:1552: SAI5 TX FCOMB configuration set to 0.
bootstrap2 > imx_sai_init:1567: SAI5 driver running with DMA RX mode disabled (mode 2)
bootstrap2 > imx_sai_init:1574: SAI5 driver running with DMA TX mode disabled (mode 2)
bootstrap2 > of_gpio_request_by_name:234: Error while parsing handle and args for gpio,mclk-is-extern node
bootstrap2 > imx_spdif_init:1091: SPDIF driver running with DMA mode enabled.
bootstrap2 > imx_spdif_init:1111: Using 4 period buffers of length 2048 bytes
bootstrap2 > RX circ buf (size / address) : 1048576bytes / 0x0xffff000000712d80
bootstrap2 > SPDIF base address: 0xffffffff30090000
bootstrap2 > SCR: 0x170437
bootstrap2 > SRCD: 0x0
bootstrap2 > SRPC: 0x98
bootstrap2 > SIE: 0x0
bootstrap2 > SIS: 0x2
bootstrap2 > SRFM: 0x96d1
bootstrap2 > STC: 0x20900
bootstrap2 > imx_uart_init:537: UART: started IRQ driven TX
bootstrap2 > imx_uart_init:548: Using TX buffered mode
bootstrap2 > calling apps_init()
bootstrap2 > INIT: cpu 0, calling hook 0xffff00000041720 (appargs_target) at level 0xa0000, flags 0x1
bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(100)...
bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(100)...
bootstrap2 > INIT: cpu 0, calling hook 0xffff00000041720 (appargs_hal) at level 0xaffff, flags 0x1
bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(100)...
bootstrap2 > of_device_init_all:892: Initializing devices, initlvl(1000)...
bootstrap2 > starting app shell
bootstrap2 > Initialization devices, initlvl(4000)...
bootstrap2 > shell > entering main console loop
bootstrap2 > INIT: cpu 0, calling hook 0xffff00000041720 (appargs_app) at level 0xaffff, flags 0x1
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.

### 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 `dtmf_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 than 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, \texttt{PROJECT} is \texttt{imx8mm-dts} for i.MX 8M Mini and \texttt{imx8mn-dts} for i.MX 8M Nano.

If the toolchain has not been installed to the /opt default location, environment variable \texttt{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 \texttt{build-<PROJECT>}. Artefacts to be used on target are as follows:

- \texttt{lk.bin}: Little Kernel binary
- \texttt{imx8m(m,n)-evk.dtb}: device tree for EVK without HiFiBerry sound card
- \texttt{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 \texttt{littlekernel-imx8}, in directory \texttt{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 >;
```

NXP Semiconductors
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.

![Architecture of “hifiberry” application](image)

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:

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

5.3.4.3 Parsing application arguments

The prototype of the application’s main function is:

```c
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:

```c
/* 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:

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

To open SAI RX class:

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

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

```c
/* 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...

```c
/* 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:

```c
/* 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:

```c
/* get the PCM512x dev by bus id */
cpym512x_dev = class_dac_get_device_by_id(51);
```

The DAC class is then opened:

```c
/* 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…

```c
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:

```c
/* get the PCM186x dev by bus id */
cpym186x_dev = class_adc_get_device_by_id(18);
```

The ADC class is then opened:

```c
/* pcm186x setup */
ret = class_adc_open(pcm186x_dev);
```
Once the ADC class is open, its characteristics are set by sending structure \texttt{adc\_audio\_hw\_params\_t} to function \texttt{class\_adc\_set\_format()}. This structure defines parameters like the sample rate, the number of channels, sample format...

```c
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

<table>
<thead>
<tr>
<th>Revision number</th>
<th>Date</th>
<th>Substantive changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LF5.10.35_2.0.0</td>
<td>06/2021</td>
<td>Initial release of Linux Harpoon.</td>
</tr>
<tr>
<td>LF5.10.52_2.1.0</td>
<td>09/2021</td>
<td>Moved Harpoon artifacts to the /usr/share/harpoon directory, and added the Section &quot;Starting Linux kernel&quot;.</td>
</tr>
<tr>
<td>LF5.10.72_2.2.0</td>
<td>12/2021</td>
<td>Minor updates for the LF5.10.72_2.2.0 release.</td>
</tr>
</tbody>
</table>
Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

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.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. 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 decision regarding its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLY, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Hybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, "vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.

© NXP B.V. 2021. All rights reserved.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 17 December 2021
Document identifier: IMX/HPUG