
Hifi4 DSP User's Guide

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
1.0	11-Feb-2018	Weiguang Kong	Initial Draft

Table of Contents

1	Introduction	4
2	System Architecture	5
3	File Organization.....	6
3.1	Hifi4 driver	6
3.2	Hifi4 framework	6
3.3	Hifi4 wrapper and unit test	6
3.4	Interface header files	6
4	Hifi4 toolchain setup	7
4.1	Xtensa development toolchain	7
4.2	Linaro compiler toolchain	8
5	Usage of Hifi4 binary files	9
5.1	Get Hifi4 binary files.....	9
5.2	Binary files in linux rootfs	9
5.3	Unit test and playing	10
6	Memory allocation for Hifi4.....	11
7	Make loadable library for Hifi4.....	12
7.1	Finding the Custom LSPs.....	12
7.2	Modifying and Compiling the Source Code.....	12
7.3	Linking the Library Code.....	13

1 Introduction

This document is the overall introduction of hifi4 dsp, include system architecture, file organization, hifi4 related toolchain and so on. From this document, users can have an overall understanding of hifi4 related code. Currently, the hifi4 dsp is used to decode and encode audio streams in imx8qxp platform. It supports the following codecs. In addition, for current hifi4 framework and hifi4 driver, they can support up to 5 clients to run together.

Decoder:

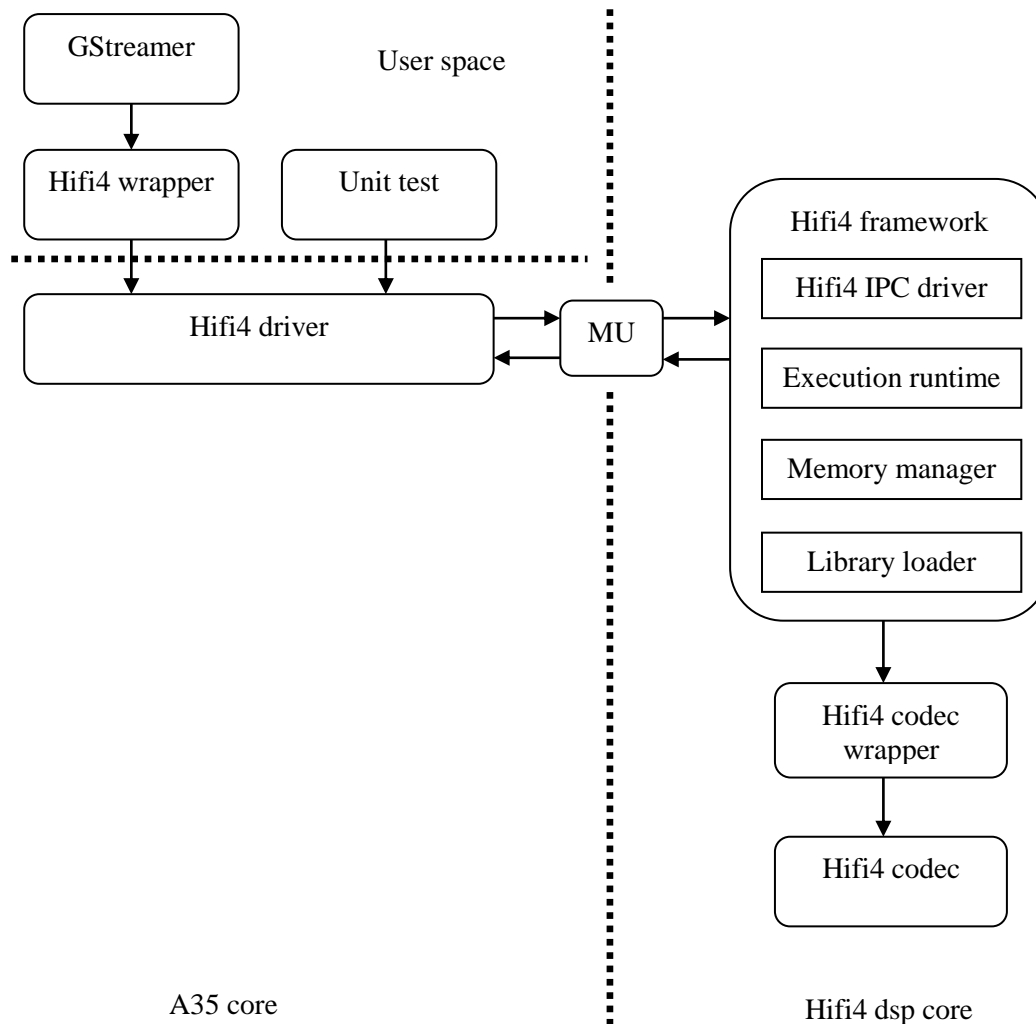
- AAC-LC
- AAC plus(HE-AAC/HE-AACv2)
- BSAC
- DAB+
- MP2
- MP3
- DRM
- SBC

Encoder:

- SBC

2 System Architecture

The following figure gives the overall system architecture of hifi4 related code.



The hifi4 related code mainly includes hifi4 framework, hifi4 driver, hifi4 wrapper, unit test, hifi4 codec wrapper and hifi4 codec. The hifi4 framework is the firmware code which is running on hifi4 dsp core. The hifi4 driver is used to load hifi4 firmware into memory and transfer messages between user space and hifi4 framework. The MU(message unit) is used to transfer messages between A35 and hifi4 dsp core. The hifi4 wrapper and unit test are application code in user space, which use `ioctl()` interface to transfer messages between hifi4 driver and user space. In addition, the hifi4 wrapper is used to provide unified interfaces for the GStreamer. The hifi4 codec provides actual decoding and encoding functions. The hifi4 codec wrapper is a wrapping code for the hifi4 codec and provides unified interfaces for the hifi4 framework.

3 File Organization

The hifi4 framework, hifi4 wrapper and unit test's code are placed in imx-audio-framework package. The hifi4 driver's code belongs to linux kernel. The hifi4 codec wrapper and hifi4 codec are license restricted code, license authorization is required if you want to get them.

3.1 Hifi4 driver

The hifi4 driver's code can be found in linux kernel, it includes 3 files and the detailed information is as following.

```
linux-kernel/sound/soc/fsl/fsl_hifi4.c
linux-kernel/sound/soc/fsl/fsl_hifi4.h
linux-kernel/include/uapi/linux/mxc_hifi4.h
```

3.2 Hifi4 framework

The hifi4 framework's code is placed in the following path.
imx-audio-framework/hifi4_framework

3.3 Hifi4 wrapper and unit test

The hifi4 wrapper and unit test are placed in the following path.
imx-audio-framework/hifi4_wrapper
imx-audio-framework/unit_test

3.4 Interface header files

The hifi4 related code mainly include 4 interface header files.

```
imx-audio-framework/include/mxc_hifi4.h
imx-audio-framework/include/hifi_codec_interface.h
imx-audio-framework/hifi4_wrapper/include/uni_audio/fsl_unia.h
imx-audio-framework/hifi4_wrapper/include/uni_audio/fsl_types.h
```

The mxc_hifi4.h file is same as the header file in linux kernel, this file includes the interfaces and command definitions, which will be used by hifi4 wrapper and unit test. Similarly, the hifi_codec_interface.h file wraps the hifi4 codec's header files. It includes unified interfaces and command definitions, which can be used by hifi4 framework. The fsl_unia.h and fsl_types.h header files include interfaces and command definitions, which can be used by GStreamer.

4 Hifi4 toolchain setup

Before you compile hifi4 related code, you should set up hifi4 related toolchains firstly. The hifi4 framework, hifi4 codec wrapper and hifi4 codec use Xtensa development toolchain. The hifi4 wrapper and unit test use the Linaro compiler toolchain for yocto platform.

4.1 Xtensa development toolchain

Xtensa development toolchain consist of two components that are installed separately on your linux system, including:

- Configuration-independent Xtensa Tool
- Configuration-specific core files and Xtensa Tool

For Configuration-independent Xtensa Tool, it is released by Cadence. For current code, the version of this tool is XtensaTools_RF_2016_4_linux.tgz. You can use two ways to get this package, one is downloading it from Xtensa Xplorer, the other is getting it from other people who have installed the Xtensa Xplorer.

For Configuration-specific core files and Xtensa Tool, it is released by NXP. For current code, the version of this tool is hifi4_nxp_v3_3_1_2_dev_linux.tgz. In addition, you should also get the configurable memory map linker files from NXP, these files will be placed in memmap/mainsim folder.

Once you have gotten these two components, you can set up this toolchain as the following steps.

- Enter imx-audio-framework folder and execute the following command
mkdir -p ./imx-audio-toolchain/Xtensa_Tool/tools
mkdir -p ./imx-audio-toolchain/Xtensa_Tool/builds
- Set up Configuration-independent Xtensa Tool
cd imx-audio-toolchain/Xtensa_Tool
tar zxvf XtensaTools_RF_2016_4_linux.tgz -C ./tools
- Set up Configuration-specific core files and Xtensa Tool
cd imx-audio-toolchain/Xtensa_Tool
tar zxvf hifi4_nxp_v3_3_1_2_dev_linux.tgz -C ./builds
- Copy configurable memory map files to the following path
cd imx-audio-toolchain/Xtensa_Tool
cp -r memmap/mainsim ./builds/ RF-2016.4-linux/hifi4_nxp_v3_3_1_2_dev/xtensa-elf/lib
- Install Xtensa development toolchain
cd imx-audio-toolchain/Xtensa_Tool
./ builds/RF-2016.4-linux/hifi4_nxp_v3_3_1_2_dev/install --xtensa-tools ./tools/RF-2016.4-linux/XtensaTools --registry ./tools/RF-2016.4-linux/XtensaTools/config
- Set the PATH environment variable
export PATH= ./imx-audio-toolchain/Xtensa_Tool/tools/RF-2016.4-linux/XtensaTools/bin:\$PATH
- Set the LM_LICENSE_FILE environment variable

Xtensa development tools use FLEXlm for license management. FLEXlm licensing is required for tools such as the Xtensa Explorer, TIE Compiler, Xtensa C and C++ compiler. So if you plan to use a floating license, you need to install the FLEXlm license manager and set the LM_LICENSE_FILE environment variable. If any problem, you can find useful information from Cadence's document.

Xtensa® Development Tools Installation Guide User's Guide.doc

After the above steps, the Xtensa development toolchain is set up successfully. In addition, the Xtensa Tools and additional tools are provided as 32-bit(x86) binaries. They are supported on 32-bit(x86) systems, and also on recent 64-bit(x86-64) systems that have appropriate 32-bit compatibility packages installed. If you use a 64-bit system, for examples ubuntu 16.04(64-bit), you should install 32-bit compatibility packages firstly. You can use the following commands.

```
sudo apt-get install lib32ncurses5 lib32z1
sudo dpkg --add-architecture i386
sudo apt-get install libc6:i386 libstdc++6:i386
```

4.2 Linaro compiler toolchain

Currently, gcc-linaro-4.9-2015.02-3-x86_64_aarch64-linux-gnu toolchain is used to compile the hifi4 wrapper and unit test's code for yocto platform. This toolchain should be placed in /usr folder of your linux server if you want to build code successfully, you can get more information from the Makefile file of hifi4 wrapper and unit test.

5 Usage of Hifi4 binary files

5.1 Get Hifi4 binary files

You can get hifi4 binary files from NXP directly or compile the source code to produce them by yourself.

After hifi4 related toolchains have been installed on your server successfully, you can compile the hifi4 related code. You can just execute make command in imx-audio-framework folder to compile hifi4 framework, hifi4 wrapper and unit test. If you want to compile them separately, you can refer the REAME file in imx-audio-framework folder. After the compiling process, you can find the binary files in imx-audio-framework/release folder.

For hifi4 framework:

`imx-audio-framework/release/hifi4.bin`

For hifi4 wrapper:

`imx-audio-framework/release/wrapper/lib_dsp_wrap_arm_elinux.so`

For unit test:

`imx-audio-framework/release/exe/dsp_test`

For hifi4 codec wrapper and hifi4 codec's binary files, you need to get the authorization.

5.2 Binary files in linux rootfs

If you want to use these binary files to run, you should place them into linux rootfs. The location of each binary is as following. For hifi4 framework, its location is determined by hifi4 driver, so you should keep it in the following place. For hifi4 wrapper, its location is determined by GStreamer, you should keep it in the following place. For unit test, you can change its location by yourself.

For hifi4 framework:

`/lib/firmware/imx/hifi/hifi4.bin`

For hifi4 wrapper:

`/usr/lib/imx-mm/audio-codec/wrap/lib_dsp_wrap_arm_elinux.so`

For hifi4 codec wrapper and hifi4 codec, you can keep them in the following place of linux rootfs.

`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_codec_wrap.so`
`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_mp3_dec.so`
`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_aac_dec.so`
`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_bsac_dec.so`
`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_dabplus_dec.so`
`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_drm_dec.so`
`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_mp2_dec.so`
`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_sbc_dec.so`
`/usr/lib/imx-mm/audio-codec/hifi/lib_dsp_sbc_enc.so`

5.3 Unit test and playing

After you have placed these binary files into correct location of rootfs. If you want to decode or encode audio streams directly, you can use the unit test binary file. For example, if you want to decode one .mp3 file, you can use the following command.

```
./dsp_test -f1 -d16 -itest.mp3 -otest.pcm
```

You can get more help information about dsp_test by executing the following command.

```
./dsp_test
```

If you want to play one music by using GStreamer and hifi4 wrapper, you can use gplay and execute the following command.

```
gplay-1.0 test.mp3
```

6 Memory allocation for Hifi4

The hifi4 firmware is loaded into memory by hifi4 driver, the loading address is decided by memory map linker files of Xtensa development toolchain. You can change the loading address based on the following memory map list of imx8qxp.

Cortex-A35/Cortex-M4	Hifi4	Content
	0x80000000 ~ 0x806FFFFFFF	Reserved (Cannot be used)
0x59700000 ~ 0x5971FFFF	0x80700000 ~ 0x8071FFFF	Hifi4 OCRAM-System RAM
0x59720000 ~ 0x5973FFFF	0x80720000 ~ 0x8073FFFF	Hifi4 OCRAM-System ROM
	0x80740000 ~ 0x80FFFFFFF	Reserved (Cannot be used)
0x80700000 ~ 0x8073FFFF		Linux kernel (not visible from Hifi4)
0x81000000 ~ 0x9FFFFFFF	0x81000000 ~ 0x9FFFFFFF	SDRAM

Currently, linux kernel has reserved one memory for hifi4 dsp in SDRAM separately. The range of this reserved memory is 0x8e000000 ~ 0x8fffffff (32M bytes). You can set this reserved memory by changing the fsl-imx8qxp.dtsi file in linux-kernel/arch/arm64/boot/dts/freescale folder.

```
reserved-memory {
    .....

    hifi4_reserved: hifi4@0x8e000000 {
        no-map;
        reg = <0 0x8e000000 0 0x1fffffff>;
    };

    .....
}
```

For current reserved memory, hifi4 driver has split it into two parts. One part is used to keep the hifi4 firmware, the other part is considered as scratch memory for hifi4 framework. The detailed information about these two parts is as following.

0x8e000000 ~ 0x8eFFFFFFF	Hifi4 firmware(16M bytes)
0x8f000000 ~ 0x8FFFFFFF	Scratch memory(16M bytes)

Notice that if you make some changes for memory map linker files of Xtensa development toolchain, please do related changes for the hifi4 driver.

7 Make loadable library for Hifi4

The hifi4 loadable library is available as two different types: a fixed-location overlay and a position-independent library. For a fixed-location overlay, you can load code into a predetermined location in memory. For a position-independent library, you can load code at an address determined at run time. You can link the loadable library using a special LSP named `piload` or `pisplitload` (see the Xtensa Linker Support Packages (LSPs) Reference Manual). For our binary files that are used by hifi4 framework, it belongs to the position-independent library, so this chapter briefly documents the flow to generate the position-independent library. You can get more detailed information from the Chapter 4 of Xtensa System Software Reference Manual.

A position-independent library can be loaded and run at any address that supports both code and data, like normal system RAM. Alternatively, you can use the `pisplitload` LSP to load code and data into separate memory blocks located in local RAMs. Library location need to be decided until run time.

Before making loadable library, Xtensa development toolchain should be installed firstly, then you can follow the steps below.

7.1 Finding the Custom LSPs

Loadable libraries need to be linked with a custom linker support package. For position-independent libraries, you don't need to generate or edit an LSP. Instead, you will link your position independent library by using the standard `pisplitload` LSP provided as part of your configuration.

7.2 Modifying and Compiling the Source Code

The API only allows the main program to directly access a single symbol in the library, the `_start` symbol. The library can't directly access any symbols in the main program. Any other symbol's address must be passed to or from the library as an argument to the `_start` function. The following code is an example.

```
#include <stdio.h>

/* declare a printf function pointer */
int (*printf_ptr)(const char *format, ...);

/* replace all calls to printf with calls through the pointer */
#define printf printf_ptr

/* This is the function provided by the library */
char * interface_func(unsigned int input)
{
    printf("executing function interface_func\n");
}
```

```

    return "this is string returned from interface_func";
}

void * _start(int (*printf_func)(const char *format, ...))
{
    printf_ptr = printf_func;

    /* The main application wants to call the function interface_func, but can't directly
       reference it. Therefore, this function returns a pointer to it, and the main application
       will be able to call it via this pointer. */

    return interface_func;
}

```

The main application calls `_start`, passing a pointer to `printf`, and taking a pointer to `interface_func()` in return. If the library and the main program need to communicate the value of more than one symbol, then `_start` can take can return arrays of pointers, rather than just single pointers.

After you have finished your source code, you can use `xt-xcc` of Xtensa development toolchain to compile the code. For position-independent libraries, because it can be loaded at any address, make sure that the code in the library is position-independent by using the `-fpic` flag along with your normal compile options, as shown below:

```
xt-xcc -fpic -O3 -o library.o -c library.c
```

7.3 Linking the Library Code

In this step, you will link the library code into a loadable library using the appropriate LSP. For position-independent library, you can use the following command.

```
xt-xcc -mlsp=pisplitload -Wl,--shared-pagesize=128 -Wl,-pie -lgcc -lc -o library.so library.o
```

After this command, you can get a position-independent library with code and data loadable separately. If you want to get a contiguous position-independent library, you can use the following command.

```
xt-xcc -mlsp=piload -Wl,--shared-pagesize=128 -Wl,-pie -lgcc -lc -o library.so library.o
```

After the linking stage, you can get a loadable library which can be loaded by hifi4 framework. In addition, for current hifi4 framework, it can only support loading code and data section separately.