HiFi AF Hosted User Guide

User Guide

For HiFi DSPs and Fusion F1 DSP

Cadence Design Systems, Inc.
2655 Seely Ave.
San Jose, CA 95134
www.cadence.com

Version 0.1
April 2023

Contents

Figures

Tables

Document Change History

Version	Changes
0.1	Initial alpha release

1. Introduction

1.1 Hosted XAF Overview

Hosted XAF implementation separates the Host or application side processing from the DSP side processing. There exists a host core that runs Linux and provides commands to the DSP for creating and running processing pipelines. The IPC layer facilitates communication between two separate subsystems and operating systems. The App Interface Layer of XAF runs only on the host CPU whereas the DSP Interface Layer is enabled on DSP.

This document highlights the information related to the Hosted implementation. More details on XAF is available in the XAF Programmer's guide HiFi-AF-Hostless-ProgrammersGuide.pdf.

1.2 Hosted XAF Architecture

The following diagram shows the software stack of Hosted XAF. While the software stack looks quite similar to Hostless XAF, there are few distinctions between the two XAF implementations.

1. In Hostless XAF, the IPC layer is RTOS based whereas in Hosted XAF the IPC layer is a shared memory Linux kernel driver that facilitates communication between the Host application and DSP.
2. RTOS is replaced by native OS APIs in Host application.

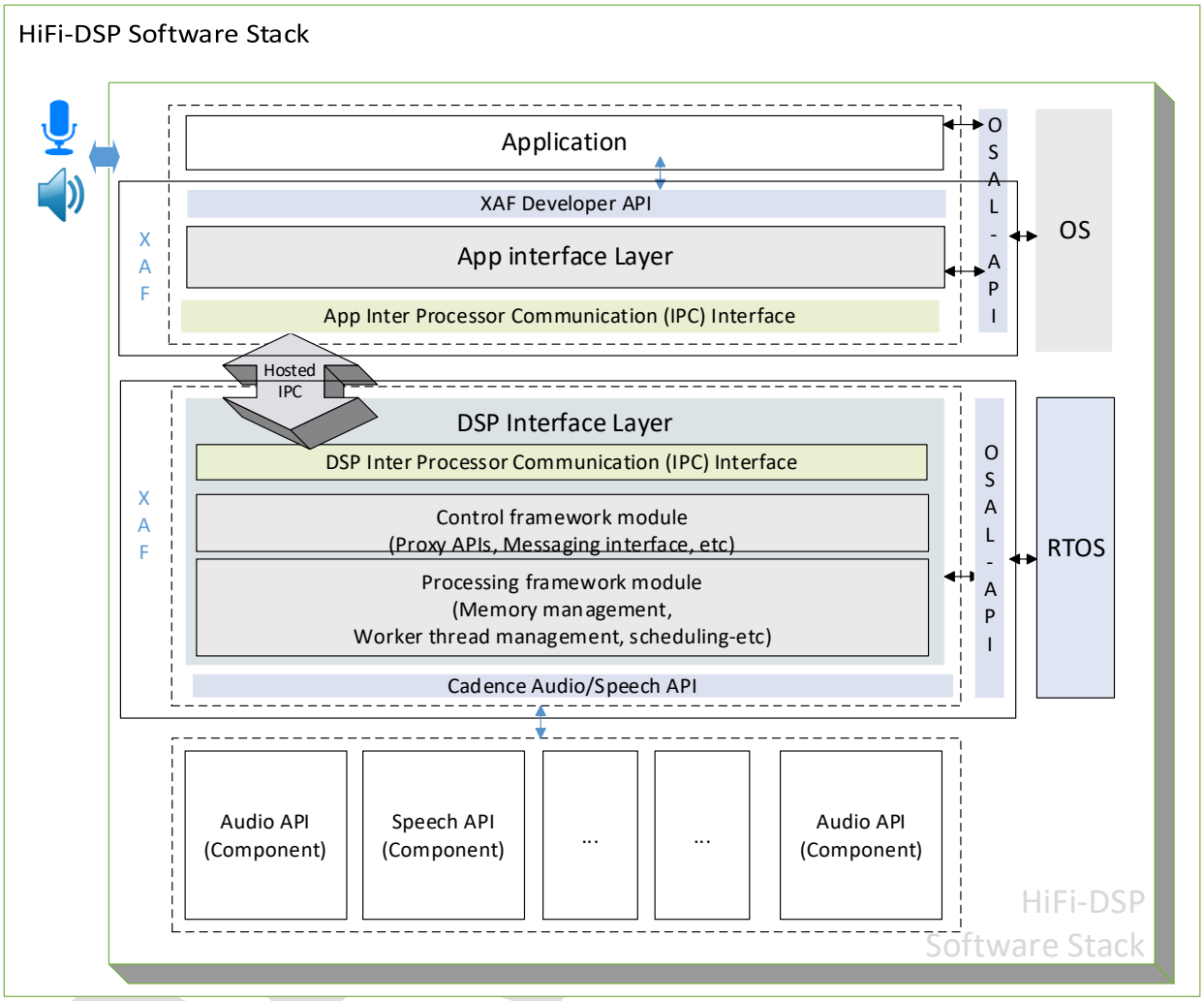

Figure 1-1 Application Software Stack Diagram (Hosted XAF)

The following diagram represents command and response flow between the Host and DSP. Application pipeline is created by the host-CPU. Any command (created by Developer API call from application) from the App Interface Layer is received by DSP Interface Layer through inter processor communication channel (hosted-IPC) on master DSP (DSP0). The command and response parsing from App Interface Layer happens on the DSP core with access to Proxy MSGQ (DSP0).

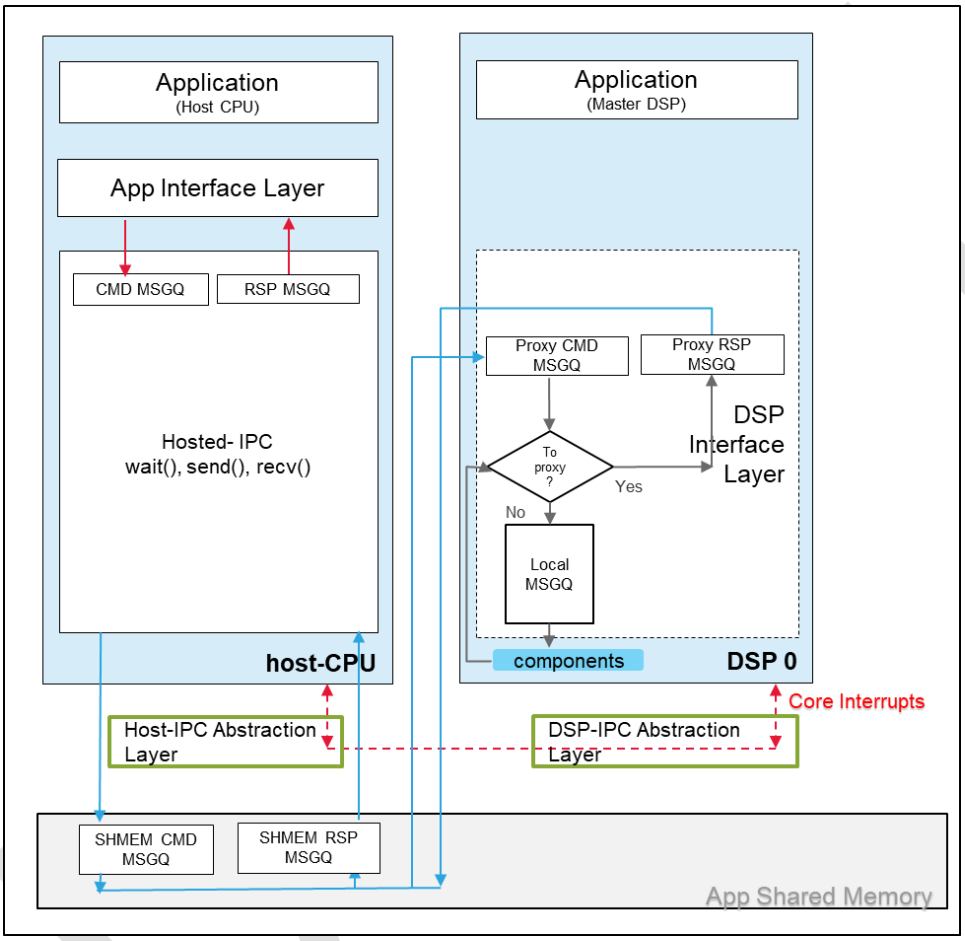

Figure 1-2 Hosted-XAF Software Architecture

1.3 Hosted IPC Overview

The Hosted IPC is implemented using a Linux kernel driver that provides set of APIs (system calls) for communication with the Host application and writes to the shared memory that is shared between DSP and Host. The Host application issues commands by “writing” to the file I/O interface provided by the driver and receives responses by “reading” from the file descriptor.

The kernel driver is a separate piece of code from host and DSP libraries and needs to be configured with the shared memory location.

The IPC kernel driver runs a polling thread that monitors incoming interrupts from DSP for receiving responses. It also provides “wait” functionality for the host application to wait until a response is ready.

The kernel driver also facilitates access of the shared memory to the host application via memory mapping.

1.4 Hosted XAF Platform

The following diagram represents the Linux + XTSC platform that is used as a reference platform for the Hosted XAF implementation.

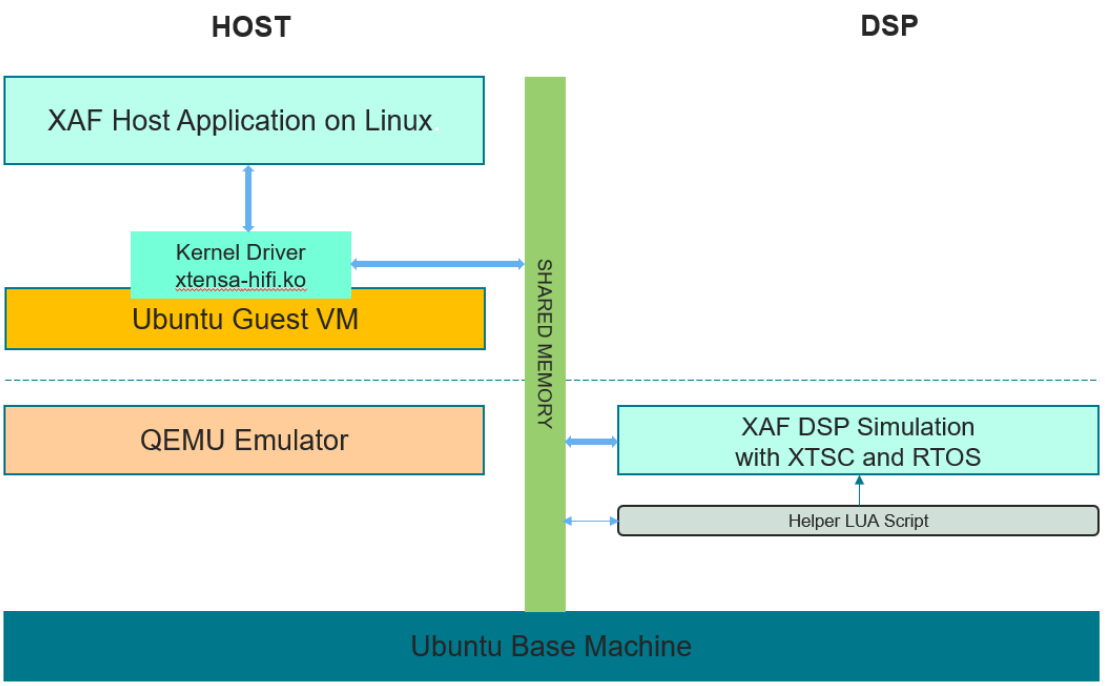

Figure 1-3 Hosted XAF Co-Simulation

XAF Host-AP code runs in a virtual machine on the QEMU emulator setup as shown above. An enhanced version QEMU emulator is used which facilitates shared memory access to the guest VM as a physical memory device.

DSP executes under XTSC environment, similar to that of Hostless XAF.

Shared-memory address is a unique global address common to Host application processor and HiFi DSP.

2. How-To Guide for Hosted XAF Package

2.1 Hosted XAF Package Directory Structure

Testbench specific source files (/test/src/)

xaf-amr-wb-dec-test.c

xaf-capturer-mp3-enc-test.c

xaf-capturer-pcm-gain-test.c

xaf-capturer-tflite-microspeech-test.c

xaf-dec-mix-test.c

xaf-dec-test.c

xaf-full-duplex-opus-test.c

xaf-gain-renderer-test.c

xaf-mimo-mix-test.c

xaf-mp3-dec-rend-test.c

xaf-pcm-gain-test.c

xaf-person-detect-microspeech-test.c

xaf-playback-usecase-test.c

xaf-renderer-ref-port-test.c

xaf-tflite-person-detect-test.c

Common testbench source files (/test/src/)

xaf-clk-test.c – Clock functions used for MCPS measurements.

xaf-dsp-test.c –application function with API calls to create DSP thread.

xaf-mem-test.c – Memory allocation functions.

xaf-utils-test.c – Other shared utility functions.

`xaf-fio-test.c` – File read and write support.

Test directories (in `/test/`)

`include/audio` – API header files for different audio components.

`plugins/` – Wrappers for the different audio components.

`test_inp/` – Input data for the test execution.

`test_out/` – Output data from test execution will be written here.

`test_ref/`– Reference data against which the generated output can be compared.

DSP binary build directory (/`test/build/`)

`makefile_testbench_dsp` – makefile to build DSP binary

`cosim-launcher2.py` – python script to help launch QEMU VM and execute DSP binary + Lua script under xtsc-run environment

Host-AP binary build directory (/`test/build_host/`)

`common.mk` – common make template, included in other makefiles

`makefile` – Top makefile to build application binary

`makefile_host` – makefile to build XAF library (.a) required for host-AP part

`makefile_testbench` – testbench makefile with rules, options to compile application test files, link component library or .o files, host-AP XAF library and create host-AP application binary.

`symbols_af_hosted.txt` – XAF library symbols available for application at host-AP

`xaf_xtsc_run.sh` – helper script to run/execute application binary

Hosted XAF DSP Library (`/algo/`)

`hifi-dpf/` – DSP Interface Layer source and include files.

Hosted XAF Host Library (`/algo/`)

`host-apf/` – App Interface Layer source and include files. Includes XAF Developer APIs implementation.

`src/xf-fio.c` – IPC functions for Host-AP – Kernel-IPC interface (replacement for src/xf-msgq1.c of hostless-XAF)

`include/sys/fio` – IPC functions header/helper files for Host-AP – Kernel-IPC interface (replacement for include/sys/xos-msgq of hostless-XAF)

Hosted XAF Library common (`/algo/`)

`xa_af_hostless/` – XAF common internal header files.

XAF include directories (`/include/`)

`audio/` – XAF processing class specific header files. Also includes API, error, memory, type definition standard header files.

`sysdeps/freertos` – FreeRTOS OSAL API definition header files. (Used by DSP)

`sysdeps/xos` – XOS OSAL API definition header files. (Used by DSP)

`sysdeps/linux` – Linux OSAL API definition header files. (Used by host-AP)

`xaf-api.h` – XAF Developer APIs header file.

`xf-debug.h` – XAF debug trace support header file.

XAF-library build directory (`/build/`)

`build/common.mk` – common make template, included in other makefiles

`build/makefile` – Top make file

`build/makefile_lib` – files, options to build library (.a)

`build/getFreeRTOS.shbuild/getTFLM.sh` – shell script to facilitate download and build tflm library

`build/readme_tflm.txt` – general guidelines for tflm testcases

`build/symbols_af_hosted.txt` – library functions available for the application

Hosted IPC Kernel Driver (`/xf_kernel_driver/`)

`src/xf-proxy.c` – IPC Kernel Driver primary source file.

`include/sys/xt-shmem/xf-config.h` – IPC Kernel Driver configuration parameters.

`include/sys/xt-shmem/xf-shmem.h` – IPC Kernel Driver helper macros and constants

`include/xf.h` – top include file for xf-proxy.c

`include/xf-proxy.h` – for xf-proxy.c

`include/xf-opcode.h` – include file that contains the opcode and id bit-mask macros

Subsystem (`/xtsc/`)

`makefile` – builds subsystem (with 1 HiFi DSP) to be used to run DSP-binary

`TestBenchMaster1c.vec` – Lua script for polling Host-AP interrupt on a SHMEM address and map it to HiFi-DSP interrupt.

`xaf_xtsc_sys_1c.xtsys`, `xaf_xtsc_sys_1c.yml` – XTSC subsystem specification files

2.2 Build and Execute using tgz Package

In Hosted XAF, separate binaries are built for Host-AP and DSP. The host binary is built with GCC tools whereas the DSP binary is built with Cadence Xtensa tools. The following sections describe steps to build and run the binaries.

Note that for the simulation to run, both the DSP and Host-AP should be up and running before XAF applications can be created, which requires a co-simulation setup to launch both DSP and Host-AP simulation. The DSP binary execution is done with XTSC-simulation and the host binary is executed in Linux.

2.2.1 Co-simulation Overview

Co-simulation requires a base Linux machine to execute XTSC and DSP side XAF code and a guest virtual Linux machine running under QEMU to run the Host-AP code. The co-simulation process consists of the following steps:

a. Building the subsystem (single DSP)

b. Building the binary for HiFi-DSP

c. Launching the DSP binary under XTSC

d. Start the QEMU emulation to boot up a guest VM with Linux

e. Copy XAF Host-AP code and IPC Linux Kernel code (xf_kernel_driver) into the guest VM

f. Compile and insert the Linux Kernel Driver

g. Compile the host binary applications

h. Execute host applications

Steps (a) to (e) are to be done on the base machine and steps (f) to (h) are to be done on the guest virtual machine.

The steps on the base machine have been simplified using scripts (`xtsc/makefile`, `test/build/cosim-launcher2.py`) that perform these tasks and start co-simulation. However, the following pre-requisites need to be fulfilled for a successful execution.

Pre-requisites for co-simulation

- A base Ubuntu machine (Ubuntu 20.04 or later) to run XTSC and QEMU emulator. It needs to have at least 1GB disk space to store the guest OS image which is typically few hundred megabytes when downloaded and grows upon usage.

- Python3 should be installed on the base Ubuntu machine.

- Xtensa tools setup (version RI-2022.9 or later) on the base Ubuntu machine for building the subsystem, building the DSP binary and XTSC launch.

- QEMU binary (patched version, to enable shared memory access by guest VM). This can be obtained from the following GitHub link:

  https://github.com/foss-xtensa/xtfld_binaries/blob/v1.0/image/qemu-system-x86_64

- Ubuntu cloud images (QEMU compatible, ready to boot VM images) from https://cloud-images.ubuntu.com/<version>/current/. Ubuntu releases from version 16.04 (Xenial) to 22.04 (Jammy) have been tested to be working.

  Notes:

  1. 32-bit Ubuntu image is available only in 18.04 (Bionic) and earlier releases. Please select an image with name *i386* for 32-bit and *amd64* for 64-bit variants. For example, the following are the URLs of 32-bit and 64-bit Ubuntu images:

     a. https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-i386-disk1.img (16.04, 32-bit)

     b. https://cloud-images.ubuntu.com/jammy/current/jammy-server-cloudimg-amd64.img (22.04, 64-bit)

  2. After downloading the image, it is recommended to verify the checksum of the downloaded file with the one provided on the web page https://cloud-images.ubuntu.com/<version>/current/MD5SUMS to ensure download integrity.

  3. The image download path should have read/write permissions.

- pc-bios and user-data.img from the following path:

  https://github.com/foss-xtensa/xtfld_binaries/tree/v1.0/image

  Note, the user-data.img file contains pre-configured login credentials for user “ubuntu”. The password is set to 12345. You may create your own user-data.img with the instructions given in section 4.3.

- Shared memory address needs to be user configured. Eg: `0xE0000000`

- SSH forward port (A unique TCP port number is required for each VM) Eg: `10022`

2.2.2 XTSC subsystem setup for HiFi-DSP

The reference setup can be found in HiFi-AF-Hostless-ProgrammersGuide.doc in section *#4.7 Building Multicore Subsystem*, with number of cores configured as 1.

Below are the additional changes required.

1. xaf_xtsc_sys_1c.yml:

The subsystem should have a DeviceMemories segment named 'Shared_RAM_L' of 512 KB or more in size

```
DeviceMemories:
  - {Name: Shared_RAM_L,
     GlobalAddr: 0xE0000000,
     Size: 256Mb,
     HostShared: true,
     HostName: Shared_RAM_L
     }
```

Note: This step is automated in xtsc/makefile

2.2.3 LUA Script setup

The Lua script "TestbenchMaster.vec" is available in sysbuilder/xtsc-run/ directory (after make sysbuild is done). Note, the Lua script is required only for simulation.

Shared memory address for interrupt from Host-AP (kernel-IPC) to DSP should be correctly provided in the LUA script.

Providing the steps for the change as example below.

Two parts in the changes required in the LUA script are

1. The HiFi-DSP interrupt used:

   In include/xaf-api.h, the XF_EXTERNAL_INTERRUPT_NUMBER is 7 which is one of the interrupts available in the subsystem along with interrupt number 6.

2. Shared memory address where the Host-AP indicates interrupt.

   INTR_SHMEM: The offset should be 0x184. The base address of shared memory (SHMEM) in this example is 0xE0000000

   INTR_SHMEM_NAME: The SHMEM segment name should be the one created by the subsystem (**Shared_RAM_L**)

```
#define INTR_SHMEM 0xE0000184

#define INTR_SHMEM_NAME "Shared_RAM_L"
```

3. The HiFi DSP address where LUA should write and map the host-AP interrupt to that of DSP interrupt

   DSP_MMIO: The MMIO address of HiFi DSP which is 0x80010000 in this case (available in sysbuilder/include/xtensa/system/xmp_subsystem.h as `XMP_core0_BINTR7_ADDR`)

   DSP_INTERRUPT_MASK: LSB arrangement of 32-bit external interrupt mask. The interrupt number used is 7 which is '0x02' in this case. (Available in sysbuilder/include/xtensa/system/xmp_subsystem.h as `XMP_core0_BINTR7_MASK`)

   DSP_INTERRUPT_TYPE: Interrupt #7 is of type "Edge" in this example.

```
#define DSP_MMIO {0x80010000}

#define DSP_INTERRUPT_MASK {"0x02 0x00 0x00 0x00"}

#define DSP_INTERRUPT_TYPE {"Edge"}
```

Note: Address update part of this step is automated in `xtsc/makefile`

2.2.4 Steps to start co-simulation (cosim)

1. Extract the TGZ package and change directory to `xa_af_hosted` directory on the base Ubuntu machine.

2. Set up environment variables to have Xtensa Tools in `$PATH, $XTENSA_SYSTEM` and `$XTENSA_CORE`.

3. Build XTSC subsystem

   ```
   $cd build; make sysbuild [SHMEM_ADDR=0xE0000000]
   ```

   or

   ```
   $make -C build sysbuild [SHMEM_ADDR=0xE0000000]
   ```

   NOTE: `SHMEM_ADDR` is optional, default address (eg: `0xE0000000`) will be used otherwise.

4. Setup XTENSA_SYSTEM and XTENSA_CORE environment variables for the subsystem created in step#3

   ```
   $setenv XTENSA_SYSTEM <base-dir>/xtsc/mbuild/package/config

   $setenv XTENSA_CORE core0
   ```

5. Build XAF library required by the DSP binary

   ```
   $make all install
   ```

6. Copy the relevant component libraries (.a files) and headers (.h files) into respective `test/plugins` and `test/include/audio` directories.

7. Build HiFi-DSP binary

   ```
   $cd ../test/build; make -f makefile_testbench_dsp ROOTDIR=../..
   ```

   To build TFLM testcases, use TFLM_SUPPORT=1 TFLM_BASE=<path to tensorflow directory> (Ex: `make -f makefile_testbench_dsp ROOTDIR=../.. TFLM_SUPPORT=1 TFLM_BASE=/home/cadence/tensorflow`)

   (Note: Any addition/deletion of plugin components is required to be done in this `makefile_testbench_dsp`. By default, all the plugins in the package i.e `test/plugins/cadence` are enabled)

8. Edit the `cosim-launcher2.py` before launch and update the paths to the pre-requisites mentioned in section 2.2.1.

   Issue the following command:

```
$python3 ./cosim-launcher2.py
```

The script will prompt to confirm the shared memory address and will replace the address at all required locations. Then it will perform steps from (c) to (d) (of co-sim steps mentioned in section 2.2.1) and in few minutes the QEMU guest VM will boot up.

Note: One can assume the VM has booted up after seeing "`Audio Master DSP[0] Ready`" and few "`DSP waiting on interrupt`" messages logged to the console by the DSP binary up and running under XTSC simulation. The DSP's *stdout* logs are available in the file `LOG_FILE` (the default name is `c0_run_log.txt`)

Details of the co-simulation commands (QEMU and XTSC command) are mentioned in Appendix: Co-simulation.

9. Next, SSH into the QEMU guest VM using the following command, assuming 10022 is the SSH forward port used during co-simulation launch.

```
$ ssh -p 10022 ubuntu@localhost
```

`password`: 12345

Notes:

1. SSH access may take longer if the VM has not booted up. There might be few errors like "`kex exchange identification`" which can be ignored and it is recommended to wait up to a minute before trying again to get the SSH password prompt.

2. After booting a new VM image for the first time, the "build-essential" package needs to be installed to install the pre-requisite tools for building XAF Host-AP code and the kernel driver. Use the following commands to install the same. Internet access is assumed on the guest VM for these commands to work.

   a. `sudo apt update`

   b. `sudo apt install build-essential`

10. Copy the Host-AP code with SCP (from the base Ubuntu machine)

```
$ scp -r -P 10022 <source_path> ubuntu@localhost:<dst_path>
```

```
Example: scp -r -P 10022 xa_af_hosted ubuntu@locahost:/home/ubuntu/
```

Note: You may build the host binaries on the base Ubuntu machine and transfer only the binaries into the guest VM and run the tests using these copied binaries.

11. On the guest Ubuntu VM SSH terminal, cd into the copied directory and build the kernel driver

```
$ cd <base dir>/xf_kernel_driver

$ make clean && make
```

This will build the kernel driver. Insert it with the following command

```
$ sudo insmod xtensa-hifi.ko
```

Note: To verify that the kernel module is inserted, use the command `lsmod | grep xtensa_hifi`

12. Build test-application binaries on host-AP

```
$ cd <base dir>/test/build_host

$ make clean && make all ROOTDIR=../..
```

This will build PCM-Gain testbench application (`xa_af_hostless_test`).

(for TFLM tests: `$ make all ROOTDIR=../.. TFLM_SUPPORT=1;`)

Notes:

1. If the build error '`/usr/bin/ld: -r and -pie may not be used together`' is encountered, enable the following flags in the file `common.mk` (enabled by default).

```
CFLAGS += -no-pie

LDFLAGS+= -no-pie
```

2. If a build error related to `"get_pc_thunk.bx"` is encountered, then enable the following flag in the file common.mk (disabled by default).

```
CFLAGS += -fno-pic
```

13. Run the application with the following command

```
$ sudo ./xa_af_hostless_test -infile:../test/test_inp/sine.pcm -
outfile:../test/test_out/pcmgain_out1.pcm

Or

$./xaf_xtsc_run.sh      xt-run      ./xa_af_hostless_test      -
infile:../test/test_inp/sine.pcm              -outfile:../test/
test_out/pcmgain_out1.pcm
```

(Note: if using `xaf_xtsc_run.sh` script, it is required to copy `xtensa-hifi.ko` to `~/kernel-driver` directory or change the path of `xtensa-hifi.ko` in `xaf_xtsc_run.sh` to point it appropriately)

14. To build other testbenches, ensure the required codec libraries and header files are present in `test/plugin` and `test/include/audio` folder and use the following command.

    ```
    $ make clean && make all-dec
    ```

    Note: For capturer-based tests, the `capturer_in.pcm` should be available in the same path as that of the DSP binary; Similarly for renderer-based tests, renderer output `renderer_out.pcm` is available in the same path as the DSP binary.

2.3 Building FreeRTOS for XAF

This section describes how to build the required version of FreeRTOS library to be used with XAF. Note that the FreeRTOS compilation is only supported under Linux environment.

1. Copy `/build/getFreeRTOS.sh` from XAF Package to the directory of choice outside XAF Package under Linux environment. This directory is referred to as `<BASE_DIR>` in the following steps.
2. Set up environment variables to have Xtensa Tools in `$PATH` and `$XTENSA_CORE` defined to your HiFi core.
3. Execute `getFreeRTOS.sh`.

   ```
   $ ./getFreeRTOS.sh
   ```

   This downloads and builds FreeRTOS library in `<BASE_DIR/FreeRTOS>`. The FreeRTOS library will be created in `<BASE_DIR>/FreeRTOS/demos/cadence/sim/build/<your_hifi_core>` directory.

2.4 Building TFLM for XAF

This section describes how to build the required version of TFLM (Tensorflow Lite Micro) library to be used with XAF. Note that this TFLM compilation method is only supported under Linux environment.

1. Copy `/build/getTFLM.sh` from XAF Package to the directory of choice outside XAF Package under Linux environment. This directory is referred to as `<BASE_DIR>` in the following steps.
2. Set up environment variables to have Xtensa Tools in `$PATH` and `$XTENSA_CORE` defined to your HiFi core.
3. Execute `getTFLM.sh <target>` as below. This downloads and builds the tensorflow TFLM libraries in the directory `<BASE_DIR>/tensorflow`

   ```
   $ ./getTFLM.sh hifi3/hifi3z/hifi4/hifi5/fusion_f1
   ```

   Following libraries will be created in directory:

`libtensorflow-microlite.a` – TFLM Library

`libmicro_speech_frontend.a` – Fronend lib for Microspeech Application

Path for HiFi 5 core:
`<BASE_DIR>/tensorflow/tensorflow/lite/micro/tools/make/gen/xtensa_hifi5_default/lib/`

Path for other cores:

`<BASE_DIR>/tensorflow/tensorflow/lite/micro/tools/make/gen/xtensa_fusion_f1_default/lib/`

4. One can copy `<tensorflow>` directory from Linux to Windows for building XAF Library and testbenches. In that case, the destination directory on Windows becomes the new `<BASE_DIR>`.

2.5 Debug Options

DSP: The XAF trace logging mechanism can be enabled with the compile switch `XF_TRACE=<1|2>` . Since we are using launch script to compile and run, the trace level can be defined in the `XF_TRACE` variable in `launch_cosim.sh`.

Notes: If trace was not enabled before launching the DSP and co-simulation, then the guest VM needs to be shut down and re-launched with trace logs enabled. This is true for any compile time changes in DSP. Trace logs logged to *stdout* are redirected to a file `c0_run_log.txt` in test/build path. If this is not the case, redirecting the co-sim output to a file can be used to collect the trace logs.

Kernel: Kernel logs are available at `/var/log/kern.log` or `/var/log/syslog`. Kernel logs provide important insight into the IPC transactions.

Notes: On any fatal error at the DSP, if the host cannot issue XF_STOP command to DSP, it is best to reboot the DSP to avoid stale state of DSP state machine affect the next tests. Similarly, on any fatal error or user aborting a running test on Host-AP, it is recommended to remove (`sudo rmmod xtensa_hifi`) and re-insert the kernel-IPC module (`sudo insmod xtensa-hifi.ko`) to avoid stale data that may affect the next execution.

Host: XAF trace logs can be enabled with the compile switch `XF_TRACE=<1|2>`

Notes: To shut down or reboot the guest Ubuntu VM, use the command "`sudo init 0`" or "`sudo sysctl poweroff`". This will gracefully power-off the VM and the co-simulation script will exit (in few seconds). Abruptly terminating the co-simulation may corrupt the VM image.

More details on XAF trace levels are mentioned in XAF Programmer's guide HiFi-AF-Hostless-ProgrammersGuide.pdf.

2.6 Shared Memory overview

With offsets and size as constants and the start address of shared memory as `0xE0000000` the memory segmentation is as shown in the following table:

Table 2-1 Shared Memory overview

Memory Segment	Size in bytes	Start Offset	Start address
reserved for kernel-IPC interrupt	`0x100`	`0x0`	`0xE0000000`
Interrupt to DSP	`4`	`0x184`	`0xE0000184`
DSP's Response ready SHMEM location	`4`	`0x18C`	`0xE000018C`
kernel-IPC SHMEM queue structures	`0x3000`	`0x1000`	`0xE0001000`
SHMEM buffers for IPC	`0x40000`	`0x4000`	`0xE0004000`

3. Limitations and Known Issues

1. Both 32-bit and 64-bit Linux versions are supported for Host-AP and IPC Kernel. This release is tested with Ubuntu Linux versions from 16.04 till 22.04.

2. Only single HiFi-DSP subsystem is supported (NCORES=1)

3. Following APIs are not supported:
   a. `xaf_comp_set_config_ext`
   b. `xaf_comp_get_config_ext`
   `NOTE:` Hence `xa_af_full_duplex_opus_test` (*xaf-full-duplex-opus-test.c)* works with decoder in raw mode only

4. DSP execution with FreeRTOS may not work for applications that have Capturer and Renderer class of components due to timer interrupts not getting generated from FreeRTOS library.

5. Tested with cache disabled.

6. Multiple memory pools feature is not supported.

7. While building application binary in `build_host` directory, it may be required to enable `-no-pie` option added in `common.mk` for both CFLAGS, LDFLAGS depending on the linker error. Similarly, GCC tools version may need appropriate Makefile changes for the host-AP binary to compile successfully.

8. Package is available in TGZ format only.

4. Appendix: Co-simulation

4.1 XTSC Command parameters

A typical XTSC launch command is provided below.

```
xtsc-run              --xtensa-system=<XTENSA_SYSTEM_PATH>        --
set_xtsc_parm=turbo=false                                         --
define=core0_BINARY=<XAF_HOSTED_CODE_PATH>/build_xtsc/xa_af_hosted_
dsp_test_core0   --define=SHARED_RAM_L_NAME=SharedRAM_L.1155041   --
define=SYSTEM_RAM_L_NAME=SystemRAM_L.1155041                      --
define=SYSTEMRAM_DELAY=100      --define=SYSTEMROM_DELAY=100      --
define=SYSTEM_RAM_L_DELAY=100   --define=SHARED_RAM_L_DELAY=100   --
include='sysbuilder/xtsc-run/multicore1c.inc'
```

4.2 QEMU Command parameters

A typical QEMU launch command is provided below, assuming the required files are available at `<QEMU_IMG_DIR>`. Some of the important options are explained below.

```
<QEMU_IMG_DIR>/qemu-system-x86_64 -L <QEMU_IMG_DIR>/pc-bios -smp 2 -m
1024    -serial    file:serial.log    -display    none    --netdev
user,id=net0,hostfwd=tcp::10022-:22 -device e1000,netdev=net0 -drive
if=virtio,file=<QEMU_IMG_DIR>/xenial-server-cloudimg-i386-
disk1.img,cache=none    -drive    if=virtio,file=<QEMU_IMG_DIR>/user-
data.img,format=raw                                              -device
xtensa_xtsc,addr=0xE0000000,size=0x0E000000,comm_addr=0xE0000000
```

`-smp`: Number of cores to be allocated to the guest VM

`-m`: Memory(RAM) to be allocated to the guest VM

`-display none`: For headless booting, this should be removed if GUI access is required, may need for debugging purposes.

`hostfwd=tcp::10022-:22` SSH forward port 10022, to access the guest VM and to transfer files.

`addr=0xE0000000,size=0x0E000000,comm_addr=0xE0000000`: Shared memory address and size.

4.3 Working with the Ubuntu image

Create login-password

By default, password for the Ubuntu cloud image is not set. Thus, users can either use an existing "`user-data.img`" or create one using cloud-image-utils **on the base machine** with the following steps:

```
$ sudo apt-get install cloud-image-utils

$ cat > user-data <<EOF
#cloud-config
password: your_custom_password
chpasswd: { expire: False }
ssh_pwauth: True
EOF

$ cloud-localds user-data.img user-data
```

Adjust screen resolution (GUI mode)

To adjust the screen resolution **on guest VM**:

```
$ sudo vi /etc/default/grub

GRUB_CMDLINE_LINUX_DEFAULT="nomodeset"

GRUB_GFXPAYLOAD_LINUX=1024x768 (adjust as per requirement)

$ sudo update-grub
```

Resize disk space

By default, the Ubuntu cloud image VMs have 2GB space on the / partition. The disk space can be expanded with the following commands **on the base machine**, before booting up the guest VM.

```
$ sudo apt-get install qemu-utils

$ qemu-img resize focal-server-cloudimg-amd64.img +10G (adjust as per
requirement)
```

4.4 DSP to Host-AP interrupt

The DSP running under `XTSC` subsystem cannot directly interrupt the kernel running on host-AP. Hence a polling mechanism is used in the kernel-IPC on the host-AP which polls a specific SHMEM address (`SHMEM_ADDR + 0x100 + IRQIN_POL`) with a specific pattern (`0x00000077`) (in `xf_kernel_driver/src/xf-proxy.c`, `xf_kernel_driver/include/sys/xt-shmem/xf-shmem.h`).

When a response is ready, the HiFi-DSP writes to this address using the offset defined by the macro `IRQOUT_POL_ADDR` by calling `xf_ipi_assert` (in `algo/hifi-dpf/include/sys/xos-msgq/xf-dp_ipc.h`).

If the polling is not desired, the mentioned address on the HiFi-DSP can be mapped appropriately to generate interrupt to the host-AP.

5. References

[1] *XAF Hostless Programmer's Guide* - HiFi-AF-Hostless-ProgrammersGuide.pdf

[2] *Patched QEMU Emulator binary* – GitHub: qemu-system-x86_64

[3] *Multicore Hostless v3.5 GA Release* – Available on XPG and GitHub: xa_hifi_af_hostless_lib_3_5_api_3_2_src.xws