cadence®

Simulate dual core system software on "Linux + XTSC HiFi with XAF"

User's Guide

Version 1.0
February 2023

For HiFi DSPs

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

Version: 1.0
Last Updated: February 2023

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

Contents

Figures

Tables

No table of figures entries found.

Document Change History

Version	Changes
0.1	Initial draft version
0.2	Added generic command and command examples
1.0	Update version to 1.0

1. Introduction to dual core system software running on “Linux + XTSC HiFi with XAF”

1.1. Document Overview

This software package is designed to accelerate the integration of audio applications using Xtensa Audio Framework (XAF) into a dual core heterogeneous system, where one core is ARM/X86 running with Linux operating system. The audio processing task is offloaded to the other core which is a HiFi core running with XOS operating system and using XRP IPC to communicate with the core running Linux.

Host Processor: X86 running with Linux OS and XRP as IPC.

DSP: XTSC HiFi4 running with XOS, XAF (Xtensa Audio Framework) and XRP as IPC.

Audio application running on Host does file I/O and/or buffer I/O, then constructs XRPM messages and sends to HiFi DSP using Xtensa Remote Processing (XRP) to process data, then DSP sends back the processed data and/or acknowledgement to Host.

1.2. Terminology

The following terms are used within this guide.

AP: Application Processor

DSP: Digital Signal Processor

IPC: Inter Processor Communication

OS: Operating System

XAF: Xtensa Audio Framework

XRP: Xtensa Remote Processing

XRPM: Xtensa Remote Processing Message

XTSC: Xtensa System C

XOS: Xtensa Operating System

2. Dual core system software building blocks

The following figure shows the various building blocks for dual core system software stack on “Linux + XTSC HiFi with XAF”.

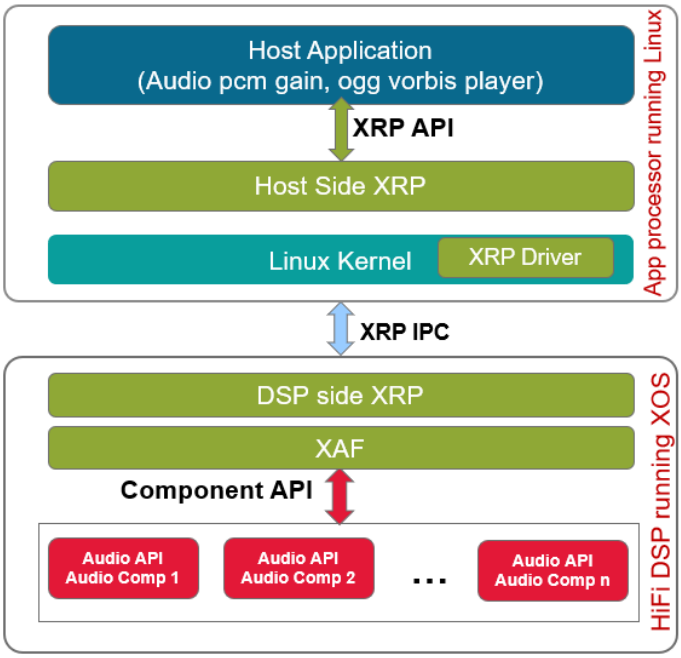

Figure 1- System Architecture

2.1. About Host Application

Host application comes with two threads each one specified with specific task.

shell_thread: Responsible for listening user input and parseing user input for valid string. User can type "help" to see all the options supported. It checks for a valid command and passes the command to **host_dsp_thread** for further processing.

host_dsp_thread: Responsible for sending commands/data synchronously from Host to DSP and receiving acknowledgement and processed data from DSP.

2.2. Xtensa Remote Processing (XRP)

Xtensa Remote Processing (XRP) is a communication interface for Linux-based systems containing Xtensa processors. It allows Linux user space tasks to send messages to the firmware running on Xtensa processors. Its implementation supports master-slave communication model with a single host node sending messages and one or more DSP nodes executing actions requested in received messages and returning the results. Message structure details are not defined by the XRP, but it is assumed that the message is a small structure that describes requested action accompanied by a vector of buffers with data or storage for that action.

Refer here for XRP user guide for more details.

2.3. DSP software running on XTSC HiFi

XRP running on HiFi DSP polls for commands and data from Host side. Once it gets XRP message, it passes the XRP messages to XAF framework which in turn calls format specific CODEC to process it.

2.4. Runtime interaction between Host and DSP

At runtime, when AP and DSP are ready, the interaction is mainly about the data and control signals. The interaction can be synchronous or asynchronous. This software package currently supports synchronous IPC.

Cadence/Tensilica XRP IPC software is designed to handle the various data and control signal interactions between AP and DSP.

This software package uses XRP in standalone-mode implementation and can run on a Linux-based system that communicates with Xtensa DSP code running in a simulator on the same Linux-based system. It consists of the following two parts: the standalone Host implementation of the XRP interface and the DSP implementation of the XRP interface.

- The standalone host XRP implementation is a library code that runs in a process that directly communicates through the shared memory with the Xtensa DSP side XRP code.
- The DSP implementation of the XRP interface is the same for both hosted and standalone host. The only difference is in the way the DSP firmware gets the shared memory address.

Figure - 1 and *Figure - 2* shows the high-level interaction between AP and DSP from software perspective. *Figure - 2* indicates a simple scenario, where AP sends command/data to DSP for processing. DSP reads data from IPC, processes it, and then sends back to AP using XRP IPC.

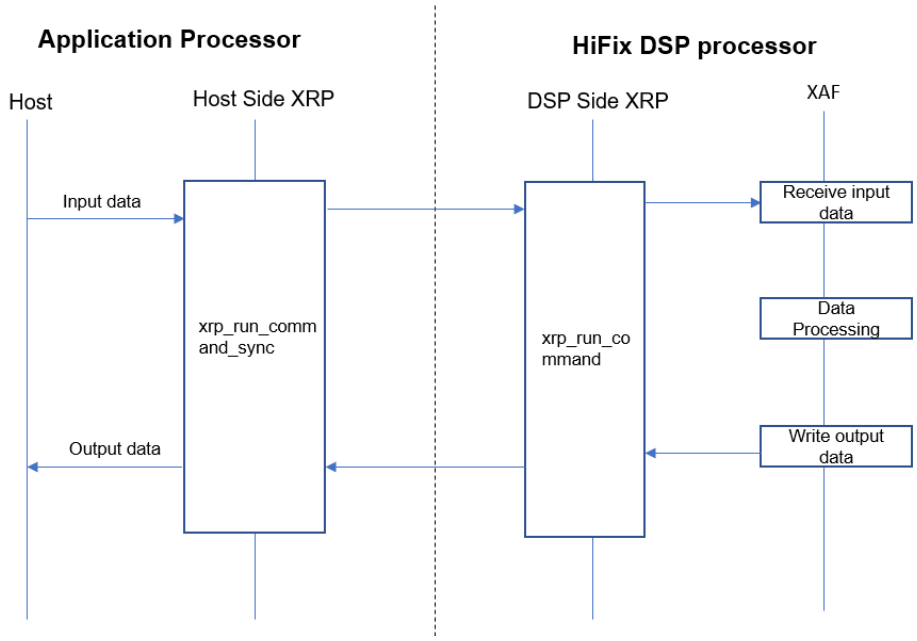

Figure 2- Runtime Interaction

3. Run and Debug from Command Line Environment / LINUX

Launch a terminal and change directory to the xa_linux_xaf: In order to use the command line Xtensa tools, environment variables need to be setup that are used by the build scripts.

```
# Add Xtensa tools binaries to PATH and Setup Xtensa env vars needed for compile and linking
export PATH=$PATH:~/xtensa/XtDevTools/install/tools/RI-2021.6- linux/XtensaTools/bin
export XTENSA_SYSTEM=~/xtensa/XtDevTools/install/tools/RI-2021.6-linux/XtensaTools/config
export XTENSA_CORE= AE_HiFi4_LE5
export PATH=~/xtensa/XtDevTools/install/tools/RI-2021.6- linux/XtensaTools/Tools/bin:$PATH

# only set this below, if /usr/bin/gcc version is gretaer than ~/XtensaTools/Tools/bin/gcc
export PATH=/usr/bin:$PATH

# Install libfdt-dev on your local machine only if /usr/bin/gcc is used.
user@linux~ xa_linux_xaf$ sudo apt-get install libfdt-dev

#Build and run pcm gain audio application on the Dual core system (Linux + XTSC HiFi)
user@linux~ xa_linux_xaf$ cd build
user@linux~ xa_linux_xaf/build$ make clean

# Copies over the XRP host source files from XtensaTools
user@linux~ xa_linux_xaf/build$ make xrp_setup

#Generates the XTSC model and the MP LSPs
user@linux~ xa_linux_xaf/build$ make build

# Generates the XRP libraries for the host
user@linux~ xa_linux_xaf/build$ make stmp-xrp

#Build XAF library with pcm gain/ogg-vorbis components.
user@linux~ xa_linux_xaf/build$ cd ../dsp/xaf-hostless/build/
user@linux~ xa_linux_xaf/dsp/xaf-hostless/build$ make clean all
user@linux~ xa_linux_xaf/dsp/xaf-hostless/build $ cd ../../../build/

#Builds the binaries for the host and the single core DSP and runs them
user@linux~ xa_linux_xaf/build$ make run
```

NOTE: The commands provided are for BASH shell on a Linux system. For other shells like C-Shell, appropriate modifications need to be done to setup above mentioned environment variables.

Once this process has finished, you should see a series of files getting compiled, followed by some output logs from a test execution, which should conclude with PASSED - Output matches with reference.
If you see this, it means that a pcm gain audio application has been built and run buffer I/O on Host Linux and processed gain on HiFi DSP.

The following log shows gain application ran on Host and DSP.

```
user@linux~ xa_linux_xaf/build$ make run
Cadence Xtensa Audio Framework
Library Name    : Audio Framework (Hostless)
Library Version : 2.3
```

API Version : 1.3

[DSP Gain] Audio Device Ready
[DSP Gain] Gainer component started
[DSP_ProcessThread] start
[DSP_ProcessThread] input over
[DSP_ProcessThread] Execution complete - exiting
[DSP_ProcessThread] exiting thread
[DSP Gain] Audio device closed

PASSED - Output matches with reference

Note: This software package is prepared only to run for **AE_HiFi4_LE5** core configuration.
To work for other cores, modify subsys.xld and subsys.yml files located at
user@linux~ xa_linux_xaf/dsp/xaf/build$

4. Usage of XAF on DSP (HiFi 5, HiFi 4, HiFi 3, etc.)

Xtensa Audio Framework (XAF) is a framework designed to accelerate the development of audio processing applications for the HiFi family of DSP cores. Application developers may choose components from the rich portfolio of audio and speech libraries already developed by Cadence and its ecosystem partners. In addition, customers may also package their proprietary algorithms and components and integrate them into the framework. Towards this goal, a simplified “Developer API” is defined, which enables application developers to rapidly create an end application and focus more on using the available components. XAF is designed to work on both the instruction set simulator as well as actual hardware.

Refer here for XAF user guide for more details.

5. Running application on Host side

Application on Host side can run in two modes. These two modes are Single run mode and Interactive mode.

5.1. Single run mode

In Single run mode application starts running with command line arguments pre provided by the user in run_sim.sh script file.

5.1.1. Single run mode with gain application

Refer Section-3 on how to use “make run” command. By default, “make run” runs in Single run mode with gain application.

```
[                  build]$ make run
rm -rf run_sim.sh; cat run_prologue.sh > run_sim.sh; echo "./host_application_test gain   & pids[0]=\$!" >> run_sim.sh; echo
 "sleep 1" >> run_sim.sh
chmod 755 run_sim.sh
./run_sim.sh
+ trap on_exit EXIT
+ trap on_exit INT
+ pids[0]=305477
+ sleep 1
+ ./host_application_test gain

========================================================================

Linux + XTSC HiFix with XAF version     : 0.1
Host process id                         : 305477

========================================================================
+ xtsc-run --set_xtsc_parm=turbo=true --set_xtsc_parm=turbo_max_relaxed_cycles=10000000 --define=DSP0_BINARY=xa_dsp_firmware
 --define=SHAREDRAM_NAME=SharedRAM_L.305477 --define=SYSTEMRAM_DELAY=1 --define=SYSTEMROM_DELAY=1 --include=sysbuild_dir/xts
c-run/SubSystem.inc

        SystemC 2.3.1-Accellera --- Aug 27 2019 15:08:06
          Copyright (c) 1996-2014 by all Contributors,
                      ALL RIGHTS RESERVED

  Cadence Xtensa Audio Framework
  Library Name    : Audio Framework (Hostless)
  Library Version : 2.3
  API Version     : 1.3

[DSP Gain] Audio Device Ready
[DSP Gain] Gainer component started
[DSP_ProcessThread] start
[DSP_ProcessThread] input over
[DSP_ProcessThread] Execution complete - exiting
[DSP_ProcessThread] exiting
[DSP Gain] Audio device closed
```

Figure 3- Single run mode with gain Application

5.1.2. Single run mode with OGG-Vorbis file playback

In order to use Single run mode with OGG-Vorbis file playback, replace 'gain' with 'file ../test/test_inp/hihat.ogg' in run_sim.sh as shown below.

Example: Launch a terminal and change directory to the xa_linux_xaf and then open run_sim.sh file from **build** folder.

user@linux:~/xa_linux_xaf$vim ./build/run_sim.sh

```
./host_application_test gain  & pids[0]=$!
```

Modify the above line from **run_sim.sh** file with the below line.

```
./host_application_test file ../test/test_inp/hihat.ogg  & pids[0]=$!
```

```
[                    build]$ ./run_sim.sh
+ trap on_exit EXIT
+ trap on_exit INT
+ pids[0]=365496
+ sleep 1
+ ./host_application_test file ../test/test_inp/hihat.ogg

================================================================================

Linux + XTSC HiFix with XAF version     : 0.1
Host process id                         : 365496

================================================================================
File format: VORBIS
>>+ xtsc-run --set_xtsc_parm=turbo=true --set_xtsc_parm=turbo_max_relaxed_cycles=10000000 --define=DSP0_BINARY=xa_dsp_firmwa
re --define=SHAREDRAM_NAME=SharedRAM_L.365496 --define=SYSTEMRAM_DELAY=1 --define=SYSTEMROM_DELAY=1 --include=sysbuild_dir/x
tsc-run/SubSystem.inc

        SystemC 2.3.1-Accellera --- Aug 27 2019 15:08:06
          Copyright (c) 1996-2014 by all Contributors,
                      ALL RIGHTS RESERVED

  Cadence Xtensa Audio Framework
  Library Name    : Audio Framework (Hostless)
  Library Version : 2.3
  API Version     : 1.3

[DSP Codec] Audio Device Ready
[DSP Codec] Decoder created
[DSP Codec] Decoder component started
  [DSP Codec] Setting decode playback format:
  Decoder    : vorbis_dec
  Sample rate: 44100
  Bit Width  : 16
  Channels   : 2
[DSP Codec] Renderer component created
[DSP_BufferThread] start
```

Figure 4- Single run mode with OGG-Vorbis file playback

5.2. Interactive mode

Host Application and DSP binary are invoked in two different terminals. User can provide the inputs to run various audio applications interactively.

5.2.1. Steps to run Interactive mode

1. Open the terminal and set the Xtensa tools as described below and change directory to xa_linux_xaf/build

   # Add Xtensa tools binaries to PATH and Setup Xtensa env vars needed for compile and linking
   export PATH=$PATH:~/xtensa/XtDevTools/install/tools/RI-2021.6- linux/XtensaTools/bin
   export XTENSA_SYSTEM=~/xtensa/XtDevTools/install/tools/RI-2021.6-linux/XtensaTools/config
   export XTENSA_CORE= AE_HiFi4_LE5
   export PATH=~/xtensa/XtDevTools/install/tools/RI-2021.6- linux/XtensaTools/Tools/bin:$PATH

   #Change directory to build
   user@linux~ xa_linux_xaf$ cd build

2. Complete Section-3 steps and ensure that host_application_test and xa_dsp_firmware binaries are generated in the **build** folder (user@linux:~/xa_linux_xaf/build$)

3. Run host_application_test from the terminal as shown in the below *Figure - 5*

#Run host_applicaiton_test
user@linux~ xa_linux_xaf/build$./host_application_test

```
[                  build]$ ./host_application_test

================================================================================

Linux + XTSC HiFix with XAF version     : 0.1
Host process id                         : 145725

================================================================================
User commands:

help      : List of all commands,
exit      : Exit both host and dsp programs,
version   : Query DSP for XAF/Vorbis Library version,
gain      : Run PCM gain on DSP with I/O buffers from host,
file      : Perform audio file decode and playback on DSP,
            USAGE  :file [list|stop|<audio_file>],
                    file  list <path>    :  List audio files from the specified directory
                                         :  By default if no path is provided, it list files from ../test/test_inp/ directory
                    file  <audio_file>   :  Decode the audio_file and write into file output
                    file  stop           :  Stop file decoding
>>
```

Figure 5- Running Host Application

4. Open another terminal and complete the step-1 as mentioned above (to set xtensa tool and change directory to build folder) and run DSP binary using xtsc-run. Get host process id of host application run (shown in the above *figure - 5*) and pass this id to the command line argument of DSP binary as shown in the below *figure - 6*.

#Execute the DSP binary
user@linux~ xa_linux_xaf/build$ xtsc-run --set_xtsc_parm=turbo=true --set_xtsc_parm=turbo_max_relaxed_cycles=10000000 --define=DSP0_BINARY=xa_dsp_firmware --define=SHAREDRAM_NAME=SharedRAM_L.145725 --define=SYSTEMRAM_DELAY=1 --define=SYSTEMROM_DELAY=1 --include=sysbuild_dir/xtsc-run/SubSystem.inc

```
[                  build]$ xtsc-run --set_xtsc_parm=turbo=true --set_xtsc_parm=turbo_max_relaxed_cycles=10000000 --
define=DSP0_BINARY=xa_dsp_firmware --define=SHAREDRAM_NAME=SharedRAM_L.145725 --define=SYSTEMRAM_DELAY=1 --define=SYS
TEMROM_DELAY=1 --include=sysbuild_dir/xtsc-run/SubSystem.inc

        SystemC 2.3.1-Accellera --- Aug 27 2019 15:08:06
          Copyright (c) 1996-2014 by all Contributors,
                      ALL RIGHTS RESERVED

  Cadence Xtensa Audio Framework
  Library Name    : Audio Framework (Hostless)
  Library Version : 2.3
  API Version     : 1.3
```

Figure 6- Running DSP binary

5.2.2. Interactive mode – user commands

Once application is started, observe terminal output with shell prompt. User commands are listed in the shell. See the below *figure - 7*.

```
>>version
Linux + XTSC HiFix with XAF version : 0.1
Component versions from DSP:
Audio Framework version 2.3
Audio Framework API version 1.3
VORBIS Decoder Lib version 1.12

>>gain

PASSED - Output matches with reference

>>file list ./
Available audio files:
File 1:: chirp_1ch_16b_192kbps.mp3
File 2:: hihat.ogg

>>file hihat.ogg
File format: VORBIS
>>
DSP file playback complete

>>file chirp_1ch_16b_192kbps.mp3
File format: MP3
>>file stop
STOP Command success!!!!

DSP file playback complete

>>exit
HOST :: EXIT COMMAND ACK FROM DSP:
```

Figure 7- User commands in Interactive mode

5.2.3. Steps to run Interactive mode with generic ‘pipe’ command structure

Provide the command as described below on the host-terminal command prompt

Example command structure on the host:

>>`pipe` "`create`,<rate,ch,pcm_width,cid,comp-type,ninbuf,noutbuf,infile-path,outfile-path>;`create` <rate,ch,pcm_width,cid,comp-type,ninbuf,noutbuf,infile-path,outfile-path>;`connect`,cid,port,cid,port,nbuf>;`connect`,cid,port,cid,port,nbuf>"

All values except for *infile-path, outfile-path* are positive integers

`pipe, create, connect` are command keywords, semicolon (;) is used as terminator of each command for that keyword.

cid is component ID 0 to N-1 in integer for N components

comp-type is enum identifying the component type xrpm_audio_component_t

ninbuf is number of input buffers (default 2) of input comp

noutbuf is number of output buffers (default 1) of output comp

The following example commands work for 1 component with input and output files.

(Test vectors need to be present in the path provided in the command structure)

PCM_GAIN: Create component PCM-GAIN with rate:44100, channels:1, pcm_width:16 bits, cid:0, comp-type:3 (DSP_COMPONENT_PCM_GAIN in `xrpm_audio_component_t` enum), number of input buffers:2, number of output buffers:0, infile-path:../test/test_inp/sine.pcm, outfile-path: ../test/test_out/out_xaf.pcm, semicolon(;)

>>`pipe` "`create`,44100,1,16,0,3,2,1,../test/test_inp/sine.pcm,../test/test_out/out_xaf.pcm;"

OPUS_ENC: Create component OPUS Encoder with rate:44100, channels:1, pcm_width:16 bits, cid:0, comp-type:4 (DSP_COMPONENT_OPUS_ENC in `xrpm_audio_component_t` enum), number of input buffers:2, number of output buffers:0, infile-path:../test/test_inp/sine.pcm, outfile-path: ../test/test_out/out_xaf.pcm, semicolon(;)

>>`pipe` "`create`,44100,1,16,0,4,2,1,../test/test_inp/sine.pcm,../test/test_out/out_xaf.bit;"

OPUS_DEC: Create component OPUS Decoder with rate:44100, channels:1, pcm_width:16 bits, cid:0, comp-type:5 (DSP_COMPONENT_OPUS_DEC in `xrpm_audio_component_t` enum), number of input buffers:2, number of output buffers:0, infile-path:../test/test_inp/sine.pcm, outfile-path: ../test/test_out/out_xaf.pcm, semicolon(;)

>>`pipe` "`create`,44100,1,16,0,5,2,1,../test/test_inp/testvector01.bit,../test/test_out/out_xaf.pcm;"

The following example commands work for chain of 3 components with input and output files.

PCM_GAIN, PCM_GAIN, PCM_GAIN: Create component PCM-GAIN with rate:44100, channels:1, pcm_width:16 bits, cid:0, comp-type:3 (DSP_COMPONENT_PCM_GAIN in

xrpm_audio_component_t enum), number of input buffers:2, number of output buffers:0, infile-path:../test/test_inp/sine.pcm, outfile-path: (leave blank), semicolon(;)

Create component PCM-GAIN with rate:44100, channels:1, pcm_width:16 bits, cid:1, comp-type:3 (DSP_COMPONENT_PCM_GAIN in xrpm_audio_component_t enum), number of input buffers:2, number of output buffers:0, infile-path: (leave blank), outfile-path: (leave blank), semicolon(;)

Create component PCM-GAIN with rate:44100, channels:1, pcm_width:16 bits, cid:2, comp-type:3 (DSP_COMPONENT_PCM_GAIN in xrpm_audio_component_t enum), number of input buffers:2, number of output buffers:0, infile-path:(leave blank), outfile-path: ../test/test_out/out_xaf.pcm, semicolon (;)

Connect cid:0 port:1 to cid:1 port:0, number of connect buffers 4

Connect cid:1 port:1 to cid:2 port:0, number of connect buffers 2

>>pipe
"create,44100,1,16,0,3,2,0,../test/test_inp/sine.pcm,;create,44100,1,16,1,3,0,0,,;create,44100,1,16,2,3,0,1,,../test/test_out/out_xaf.pcm;connect,0,1,1,0,4;connect,1,1,2,0,2;"

PCM_GAIN, OPUS_ENC, OPUS_DEC

>>pipe
"create,44100,1,16,0,3,2,0,../test/test_inp/sine.pcm,;create,44100,1,16,1,4,0,0,,;create,44100,1,16,2,5,0,1,,../test/test_out/out_xaf.pcm;connect,0,1,1,0,4;connect,1,1,2,0,2;"

Notes:

N1. Supports one input and one output stream.

N2. The message parameter offsets are provided in common/include/xrpm_msg.h (_param_index, _param_connect_index). User can modify the offsets as required (by taking into consideration param[] size in the xrpm_message structure).

N3. input-file and output directory should be valid.

For ">>file <filename>" type of commands, as the output paths are not provided, output is written to the file test/test_out/out_xaf.pcm.

N4. XRPM_Command_CompCreate command is sent from host for each component and can take multiple iterations depending on the number of components in the pipe. XRPM_Command_CompConnect command and connect info for all components are sent once and that single command should contain 5-tuples for all the connects in the pipe.

N5. The component libraries should be copied to dsp/xaf-hostless/test/plugins/cadence/<comp>/lib/.

N6. Limited testing is done. It is recommended to provide command from a fresh execution of build/host_application_test binary, in case a 2nd command issued on the host-shell doesn't work i.e exits/times-out.

6. References

[1] *Xtensa XOS Reference Manual* – For Version RI-2021.6 of the Xtensa tool chain, this is provided as part of the Xtensa tool chain, <TOOLS_INSTALL_PATH>/XtDevTools/downloads/RI-2021.6/docs/xos_rm.pdf.

[2] *HiFi Audio Codec Application Programming Interface (API) Definition*, Ver 1.0. This document is provided as part of this package.

[3] *HiFi Speech Codec Application Programming Interface (API) Definition*, Ver 1.0. This document is provided as part of this package.

[4] *Cadence MP3 Decoder* – Library version 3.18 for Tensilica HiFi DSPs.

[5] *Cadence Ogg-Vorbis Decoder* – Library version 1.12 for Tensilica HiFi DSPs.

[6] *Cadence Opus Codec* – Library version 1.8 for Tensilica HiFi DSPs.