cadence®

# How-To Guide for Multicore XAF tgz Package

## Programmers Guide

For HiFi DSPs and Fusion F1 DSP




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

Version 0.5
July 2021

# Contents

# Figures

**No table of figures entries found.**

# Tables

**No table of figures entries found.**

# Document Change History

| Version | Changes |
|---|---|
| 0.5 | Initial alpha release |

# 1. How-To Guide for Multicore XAF tgz Package

## 1.1 Multicore XAF Package Directory Structure

### Testbench specific source files (/test/src/)

xaf-pcm-gain-test.c

xaf-dec-test.c

xaf-dec-mix-test.c

xaf-full-duplex-test.c

xaf- amr-wb-dec-test.c

xaf-src-test.c

xaf-aac-dec-test.c

xaf-mp3-dec-rend-test.c

xaf-gain-renderer-test.c

xaf-capturer-pcm-gain-test.c

xaf-capturer-mp3-enc-test.c

xaf-vorbis-dec-test.c

xaf-mimo-mix-test.c

xaf-playback-usecase-test.c

xaf-renderer-ref-port-test.c

### Common testbench source files (/test/src/)

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

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

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

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

### Other 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.

## XAF library directories (`/algo/`)

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

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

`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.

`sysdeps/xos` – XOS OSAL API definition header files.

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

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

`sysdeps/mc_ipc/xf-mc-ipc.h` – IPC locks and interrupt definition header `file.`

## XAF shared memory directories (`/xf_shared/`)

`src/xf-shared.c` – IPC shared memory buffer definition.

`include/xf-shared.h` – IPC shared memory buffer macro definition and references.

## XAF system file directories (`/xtsc/`)

`xaf_xtsc_sys_2c.xtsys`, `xaf_xtsc_sys_2c.yml` – System specification files for 2 core system (NCORES=2)

`xaf_xtsc_sys_3c.xtsys`, `xaf_xtsc_sys_3c.yml` – System specification files for 3 core system (NCORES=3)

`xaf_xtsc_sys_4c.xtsys`, `xaf_xtsc_sys_4c.yml` – System specification files for 4 core system (NCORES=4)

# 1.2 Building Multicore Subsystem

## 1.2.1 Core Configuration Requirements

This section describes the core-configuration requirements for using cores to build multicore subsystem for multicore-XAF. The system files are available under $(ROOTDIR)/xtsc folder as described in the package directory structure.

1. Core configuration requirements

   Reference base cores: HiFi4 or HiFi5 (current release extensively tested for HiFi4)

   Additional configuration requirements:

   - PIFWriteResponse = 1 (XCHAL_HAVE_PIF_WR_RESP = 1)
   - IsaUseSynchronization = 1 (XCHAL_HAVE_S32C1I = 1)
   - PIFInbound = 1
   - XCHAL_NUM_INTERRUPTS > 0 (at least one edge triggered interrupt)

2. The system definition is provided in yml and xtsys file pairs in $(ROOTDIR)/xtsc folder

   xtsc/xaf_xtsc_sys_2[3,4]c.yml

   xtsc/xaf_xtsc_sys_2[3,4]c.xtsys

   Note, the above files are for example usage and user should carefully review and update these for their multicore subsystem.

3. Note, yml and xtsys files mention core config as: AE_HiFi4_LE5_XC_MOD_XTSC. User must update it with their ‘core config name’ in both files.

4. User must update yml file for PIF width, local memory access widths, cache access widths, memory configurations etc.to exactly match their core configuration.

5. For memory size and partition updates, user must update respective details in xtsys file.

## 1.2.2 Updating the shared memory

The shared memory buffer and the buffer size are defined in `xf_shared/src/xf-shared.c` and `xf_shared/include/xf-shared.h` respectively. User should update the size of the shared buffer as required. The size should be within the allocated partition as specified in xtsys file.

## 1.2.3 Steps to Build Multicore Subsystem

To build the subsystem follow these steps:

1. Go to the `xtsc` directory
2. Set the environment variables XTENSA_SYSTEM, XTENSA_TOOLS and XTENSA_CORE for the target tools.
3. For example, if the tools used is RI2019.2, then:

   XTENSA_SYSETM =../xtensa/XtDevTools/install/tools/RI-2019.2-linux/XtensaTools/config

   XTENSA_TOOLS = ../xtensa/XtDevTools/install/tools/RI-2019.2-linux/XtensaTools

   XTENSA_CORE = AE_HiFi4_LE5_XC_MOD_XTSC
4. At the prompt, enter the command

   ```
   xt-make NCORES=[2(default),3,4]
   ```

5. This will generate the subsystem as sysbuild and mbuild directories. The subsystem will be used to build both library and the test binaries as described in next section.

## 1.3 Build and Execute using tgz Package

### 1.3.1 Making the Executable

Before building the executable, ensure that multicore subsystem is ready and the environment variable `$XTENSA_CORE` is set correctly. The make commands mentioned below will build XAF Library and testbenches with XOS.

To build XAF Library and testbenches with FreeRTOS as RTOS:

1. Follow steps mentioned in Section 1.4 to build FreeRTOS library.
2. Use the make commands mentioned below with the options specified in square brackets `[]`. Note, `FREERTOS_BASE` directory should be `<BASE_DIR>/FreeRTOS` from step 1 above.

#### XAF Library:

If source code distribution is available, the library must be built before building the testbench application. To build the XAF library, follow these steps:

1. Go to `build/`.
2. At the prompt, enter to build the libraries

```
$ xt-make clean all install [XA_RTOS=freertos FREERTOS_BASE=<dir>]
NCORES=[2(default),3,4]
```

This command will build the XAF library and copy it to the `/lib/` folder.

## Testbench 1 Only:

To build the pcm-gain testbench application, follow these steps:

1. Go to /test/build.
2. At the prompt, enter:

```
$ xt-make -f makefile_testbench_sample clean all [XA_RTOS=freertos
          FREERTOS_BASE=<dir>] NCORES=[2(default), 3, 4]
```

This will build the example test application `xa_af_hostless_test_core0` to `xa_af_hostless_test_core1` for NCORES=2, additionally `xa_af_hostless_test_core2`, `xa_af_hostless_test_core3` for NCORES=4.

**Note** Both library and test binaries are created for each core. The makefiles iterate from 0 to NCORES-1 and assign XF_CORE_ID with the core number for each core. For a 4-core build, XF_CORE_ID=0 for master core, XF_CORE_ID=1 for 2nd core and so on.

## All Testbenches:

To build the other testbenches, the Cadence MP3 decoder, MP3 encoder, AMR-WB decoder, Sample rate convertor, AAC decoder, Ogg-Vorbis libraries and the respective API header files are required.

Copy these libraries to the following directories.

```
/test/plugins/cadence/mp3_dec/lib/xa_mp3_dec.a
/test/plugins/cadence/mp3_enc/lib/ xa_mp3_enc.a
/test/plugins/cadence/amr_wb/lib/xa_amr_wb_codec.a
/test/plugins/cadence/src-pp/lib/xa_src_pp.a
/test/plugins/cadence/aac_dec/lib/xa_aac_dec.a
/test/plugins/cadence/vorbis_dec/lib/xa_vorbis_dec.a
```

Copy these API header files to the following directory.

```
/test/include/audio/xa_mp3_dec_api.h
/test/include/audio/xa_mp3_enc_api.h
/test/include/audio/xa_amr_wb_codec_api.h
/test/include/audio/xa_src_pp_api.h
/test/include/audio/xa_aac_dec_api.h
/test/include/audio/xa_vorbis_dec_api.h
```

3. Go to /test/build.
4. At the prompt, enter:

```
$ xt-make –f makefile_testbench_sample clean all-dec
[XA_RTOS=freertos FREERTOS_BASE=<dir>] NCORES=[2(default), 3, 4]
```

This will build all the testbench applications.

### Special Build Settings

To build in the debug mode, add "`DEBUG=1`" to both XAF library and testbench compilation command lines described above.

To build with trace prints, add "`XF_TRACE=<TRACE_LEVEL>`" to both XAF library and testbench compilation command lines described above. For all trace prints, set TRACE_LEVEL as 1. For trace prints related to command, response transactions, set TRACE_LEVEL as 2.

## 1.3.2 Usage

The sample application executables can be run as described below using the cycle-accurate mode of the SystemC Simulator (XTSC). The input files for the applications are stored in the `test/test_inp` folder. The generated output files are available in the `test/test_out` folder. These can be compared against the reference output files in the `test/test_ref` folder. Refer to individual testbench help to get more details on command line options to run different test cases. Note that there is no difference in run commands for XAF with XOS or FreeRTOS.

### Testbench 1 only:

To run only the pcm-gain test application, at the prompt (in `test/build)`, enter:

```
$ xt-make –f makefile_testbench_sample run
NCORES=[2(default),3,4]
```

### All Testbenches:

To run all the fifteen testbenches, at the prompt (in `test/build)`, enter:

```
$ xt-make –f makefile_testbench_sample run-dec
NCORES=[2(default),3,4]
```

### Individual testcase:

The script xaf_xtsc_run.sh accepts the identical command line to that of Hostless XAF and renders it to the multicore simulator xtsc-run.

```
./xaf_xtsc_run.sh NCORES [2(default),3,4] xt-run
xa_af_playback_usecase_test -
infile:../test_inp/hihat_1ch_16b_192kbps_cbr.mp3 -
infile:../test_inp/hihat_1ch_16b_192kbps_cbr.mp3 -
infile:../test_inp/hihat_1ch_16b_44.1kHz.adts -
infile:../test_inp/hihat_1ch_16b_44.1kHz.adts -outfile:out0.pcm -
outfile:out1.pcm -core-cfg:1,1 -core-cfg:2,2 -core-cfg:3,4
```

1. The binary names provided should be without the name _coreX (the binaries get generated with <test-name>_core0, _core1 -etc)

   ```
   Example: if the testcase binaries generated are
   xa_af_playback_usecase_test_core0,
   xa_af_playback_usecase_test_core1 then provide
   `./xaf_xtsc_run.sh xt-run xa_af_playback_usecase_test -
   infile:<> -infile:<> -outfile:<>
   ```

2. One can also use the command directly without the script xaf_xtsc_run.sh with appropriate comma separated arguments

   ```
   Example: xtsc-run --
   define=core0_BINARY=xa_af_playback_usecase_test_core0 --
   define=core0_BINARY_ARGS=-
   infile:../test_inp/hihat_1ch_16b_192kbps_cbr.mp3,-
   infile:../test_inp/hihat_1ch_16b_192kbps_cbr.mp3,-
   infile:../test_inp/hihat_1ch_16b_44.1kHz.adts,-
   outfile:out0.pcm,-outfile:out1.pcm,-D:0,1,4,0,5,1,-
   C:0,1,4,0,10,1,-core-cfg:1,1 --
   define=core1_BINARY=xa_af_playback_usecase_test_core1 --
   define=XTSC_LOG=0 --include=../../xtsc/sysbuilder/xtsc-
   run/multicore2c.inc
   ```

### 1.3.3 Component creation on a Worker-Core

By default, all the components are created on core-0 which is the master core with XF_CORE_ID=0.

To create a component on a different core -core-cfg:<core>, <component id or comp_id> command-line option should be used.

For example:

```
./xaf_xtsc_run.sh NCORES 4 xt-run xa_af_playback_usecase_test -
infile:../test_inp/hihat_1ch_16b_192kbps_cbr.mp3 -
infile:../test_inp/hihat_1ch_16b_192kbps_cbr.mp3 -
infile:../test_inp/hihat_1ch_16b_44.1kHz.adts -
infile:../test_inp/hihat_1ch_16b_44.1kHz.adts -outfile:out0.pcm -
outfile:out1.pcm -core-cfg:1,1 -core-cfg:2,2 -core-cfg:3,4
```

In the above example, there are 7 components with id ranging from 0 to 6.

-core-cfg:1,1 component 1 is created on worker core 1

-core-cfg:2,2 component 2 is created on worker core 2

-core-cfg:3,4 component 4 is created on worker core 3

Rest of the components 0,3,5 and 6 are created on core 0 (master DSP)

| | |
|---|---|
| **Note** | In System C Simulator (XTSC) mode, the renderer testbench output is stored to the output file `renderer_out.pcm` in the execution directory. Similarly, the input for capturer testbench is read from the input file `capturer_in.pcm` and is expected to be present in the execution directory. |

| | |
|---|---|
| **Note** | The xt-run command token is parsed by the shell script **xaf_xtsc_run.sh** in the test/build folder into comma separated tokens as required by the SystemC simulator **xtsc-run** for the execution of the testcase. |

## 1.4 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`. 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.
4. `$ ./getFreeRTOS.sh`
5. You can copy `<FreeRTOS>` directory from Linux to Windows for building XAF Library and testbenches. In that case, the destination directory on Windows is your new `<BASE_DIR>`.