OMX Decoder Test Application Guide
OpenCORE 2.04, rev. 2
Jul 25, 2009

References

1*OpenMAX Integration Layer Application Programming Interface Specification.* Version 1.1.2, http://www.khronos.org/openmax/

2OpenMAX Call Sequences. OpenCORE 2.1, rev. 1. http://android.git.kernel.org/?p=platform/external/opencore.git;a=summary

3OpenMAX Core Integration Guide OpenCore 2.1, rev 1. http://android.git.kernel.org/?p=platform/external/opencore.git;a=summary

Table of Contents

List of Figures

1. Introduction

This document describes the Test Application Framework for testing the OpenMAX compliant audio and video decoder components. Currently it covers the following OpenMAX compliant audio/video decoders.

Video Decoders:

- AVC / H.264
- MPEG4 / H.263
- WMV

Audio Decoders:

- AAC
- WMA
- MP3
- AMR NB & WB

But it can be easily extended to add the support for others formats as well.

The test framework can run a single test or a series of tests on any component that can be specified as a command line argument. The design is kept modular and scalable, so that new test cases can be added without much effort and rework.

2. Building and running tests

The OMX decoder test application is built along with other OpenCore code. The integration of 3rd party OMX cores and OMX components into the OMX decoder test application follows the same procedure as integration of 3rd party OMX cores and OMX Components into OpenCore framework. This procedure is described in [3].

3. Command Line Arguments

Following are the command line arguments required to run the test app:

InputFileName {options}

{options} are:

-o OutFileName -r RefFileName -c CodecType -n ComponentName

-i SecondInputFileName -t x y -m/M -f AmrInputFileType -b AmrBandMode

where each of the above means:

-o OutFileName	Output file to be generated
-r RefFileName	Reference file for output verification
-c/-n Type/Name	EITHER the Codec Type (following -c) OR the complete OpenMAX Component Name (following -n) should be specified. Codec Type could be avc, aac, mpeg4, h263, wmv, wma, amr or mp3.
-t x y	A range of test cases to run from ‘x’ to ‘y’ To run one test use same index for x and y.
-m/-M	To specify it’s a mono file for AAC and MP3 audio components
-i SecondInputFileName	Second input bit-stream (of different W & H or configuration from the InputFileName) for testing the dynamic port reconfiguration test case (number 2)
-f AmrInputFileType	Input file format type required for AMR audio decoder component. This argument will only be required to run the “Without Marker Bit” test case (test case number 14). AmrInputFileType could be rtp, fsf, if2 or ets

-b AmrBandMode	Band Mode for AMR input stream, only required for running "Without Marker Bit" test case (test case number 14). AmrBandMode could be either nb or wb.

Most of the command line arguments are optional, but some are mandatory.

Mandatory arguments out of the above list are:

-InputFileName

-OutFileName

-CodecType / ComponentName

-SecondInputFileName (only for test case number 2) and

-AmrInputFileType and AmrBandMode (for test case number 14)

There is an option with the user to specify either the codec type or the complete component's name, as required to instantiate the component.

If specified, the reference file name (though optional) will help in matching the bit-exactness of the output file generated. Though it is not a mandatory requirement & the code is still going to execute without it, the SUCCESS/FAIL of the test case will be clearly determined if this file is present.

If the reference file name is not specified, the test will check the normal return values of different API's and other errors occurring in-between and will generate the end result accordingly.

4. Range of test cases

To specify the range of test cases to run, the command line argument is

-t x y

where x is the start index and y is the end index of the test case.

Note: to run one test case, use same index for x and y (e.g. to run only test case 3, use -t 3 3)

If -t argument is not provided, the OMX test application will run all the available test cases from first to last.

The current test cases can be divided into following three categories.

1) Test cases in the range of 0 – 10 are the basic test cases that can be run on all the components and verify the basic configurations & settings of the component, some of the state transitions and also test the dynamic port reconfiguration.

2) Test cases in the range of 11 – 20 are the general decoding tests that can also be run on all the components. These test cases verify the decoding of input bit-streams under several conditions.

3)Test cases in the range of 21 – 30 are specifically designed to test AVC decoding.

5. Description of Test Cases

The test cases written to test different features of the OpenMAX component and the corresponding decoder are described below.

5.1. Basic Tests

These are the basic tests mostly related to negotiating initial configuration parameters with the component and also verifying some of state transitions and dynamic port reconfiguration under various conditions.

5.1.1. GET_ROLES_TEST

This test case verifies how many roles are supported by the component and whether it supports the correct role or not. Given the codec type of the component as an input argument, the test uses the appropriate OpenMAX API's to acquire the information. Based on the above information, it then instantiates the component.

The test is considered passed if the component is created properly.

The test case number is 0.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -t 0 0

5.1.2. BUFFER_NEGOTIATION_TEST

Using the OMX_GetParameter and OMX_SetParameter API's, this test does the following tasks:

- Check the correct number of ports on the component
- Negotiate the input/output buffer counts & sizes
- Change them on the respective ports
- Verify that the component accepts the changed configuration and reports it back.
- Finally load the component, allocate the buffers and do the state transition from Loaded->Idle

The test is considered passed if all the API's work as expected and the component transitions to idle state correctly.

The test case number is 1.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c avc -t 1 1

There are no optional arguments required to run this test case.

5.1.3. DYNAMIC_PORT_RECONFIG

This test simulates the dynamic port reconfiguration sequence on the output port of the component, as specified in the OpenMAX specification.

The mandatory requirement for this test case is to specify two input bit-streams of different specifications as input arguments like different width and height for video bitstreams. The idea behind running this test case with two different bit-streams is that component may not send the Port Settings Change callback if the default settings of the component match with that of the input bit-stream, so specifying second bit-stream will make sure that at least one callback is received by the client and dynamic port reconfiguration sequence happens.

This test will also ensure that if no Port Settings Change callback arrives at the client, then the port settings should not have changed at the component's output port at the time of receiving the first filled output buffer.

The test is considered passed if the callback (port settings change) arrives at the client for at least one input bit-stream and for the same stream, other sequences of reconfiguration like port disable, port enable etc also occur properly.

The test case number is 2.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -i SecondInputFileName -t 2 2

5.1.4. PORT_RECONFIG_TRANSITION_TEST

This is a special test case that is the combination of dynamic port reconfiguration and state transitions.

This test case should only be run for the bit-streams that send the OMX_EventPortSettingsChanged callback.

While doing dynamic port reconfiguration, the client follows the following command flow:

- Port disable command and free the output buffers

- State transition command from executing -> idle

- State transition command back from Idle->executing

- Resume the dynamic port reconfiguration by sending the enable port command and allocating back the buffers

The test is considered passed if all the commands get executed properly with their respective callbacks reached back to client and test app doesn't encounter any dead-lock/crash condition in between.

The test case number is 3.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -t 3 3

5.1.5. PORT_RECONFIG_TRANSITION_TEST_2

Like the test case above, this should also be run only for the bit-streams that send the OMX_EventPortSettingsChanged callback.

There is a slight variation here in this test case for the command flow from the above test case.

While doing dynamic port reconfiguration, client follows the following command flow:

-Port disable command and free the output buffers

-State transition command from executing -> idle

-State transition command from Idle->Loaded.

-Stop the component now.

The test is considered passed if all the commands get executed properly with their respective callbacks reached back to client and test application doesn't encounter any dead-lock/crash condition in between.

The test case number is 4.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -t 4 4

5.1.6. PORT_RECONFIG_TRANSITION_TEST_3

This test case is also a slight variation of the above test case. The mandatory requirement again here is the reception of OMX_EventPortSettingsChanged callback to the client for the bit-streams that have to be run with this test case.

While doing dynamic port reconfiguration, client follows the following command flow:

Port disable command and free the output buffers

State transition command from executing -> idle

Enable port command and allocate back the buffers

Resume the reconfiguration by sending the state transition command back from Idle->executing

The expected outcome is same as that of above test case.

The test case number is 5.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -t 5 5

5.2. General Tests

These tests can also be run on any of the component. They are mostly related to the input bit-stream decoding in various conditions.

5.2.1. NORMAL_SEQ_TEST

This is the most basic of all general tests that follows the normal decoding flow of the OpenMAX API's like:

- Initializing the component
- Doing the dynamic reconfiguration if there is a port settings change callback
- Sending/receiving the input/output buffers till end of stream
- Sending the EOS stream marker flag and
- Finally stopping the component.

The API used to allocate the buffers is OMX_AllocateBuffer and the client packs one complete frame into an input buffer and sends it to the component with the marker flag (OMX_BUFFERFLAG_ENDOFFRAME) set.

The expected outcome of this test would be to match the output file generated against the reference file for bit exactness. It is therefore recommended that the reference file is specified by the user, though it is not mandatory.

If the reference file name is not specified, the test will check the normal return values of different API's and will give the end result as FAIL/SUCCESS based on those values.

The test case number is 11.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -r RefFileName -t 11 11

-r RefFileName though an optional argument here is strongly recommended.

All other arguments mentioned above in the example use case are mandatory.

Other mandatory arguments include

-m -> only required in case of mono bit-streams for MP3 and AAC components.

5.2.2. NORMAL_SEQ_TEST_USEBUFF

This test case is similar to the above NormalSeqTest test except one difference. The API used to allocate buffers in this test case is OMX_UseBuffer().

The flow of this test case is as follows:

- Initializing the component
- Doing the dynamic reconfiguration if there is a port settings change callback
- Sending/receiving the input/output buffers till end of stream
- Sending the EOS stream marker flag and
- Finally stopping the component.

Client packs one complete frame into an input buffer and sends it to the component with the marker flag (OMX_BUFFERFLAG_ENDOFFRAME) set.

As stated in the above test case, the expected outcome of this test would be to match the output file generated against the reference file for bit exactness. So it is recommended that the reference file should be specified by the user, though it is not mandatory.

If ref file if not specified, the test will check the normal return values of different API's and will give the end result as FAIL/SUCCESS based on those values.

The test case number is 12.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -r RefFileName -t 12 12

-r RefFileName though an optional argument here is strongly recommended.

All other arguments mentioned above in the example use case are mandatory.

Other mandatory arguments include

-m -> only required in case of mono bit-streams for MP3 and AAC components.

5.2.3. ENDOFSTREAM_MISSING_TEST

This test case is also similar to the NormalSeqTest, except the process of sending OMX_BUFFERFLAG_EOS flag is missing at the end of stream.

After sending all the input buffers to the component till the end of stream, the client here sends the state transition command (executing->idle) instead of sending the input buffer with EOS flag marked.

The expected outcome is that component should still be able to be destroyed correctly without any deadlocks.

The output here cannot be matched for bit exactness as the component may stop processing the buffers as soon as state transition command is received, so specifying reference file is not required in this test case.

The test case number is 13.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -t 13 13

All the arguments specified above in the example use case are mandatory.

Other mandatory arguments include

-m -> only required in case of mono bit-streams for MP3 and AAC components.

5.2.4. PARTIAL_FRAMES_TEST

In this test case, the client splits input frames into N partial fragments and sends one fragment at a time to the component in each input buffer. The input buffer containing the last piece of the input frame (i.e. the last fragment) will be marked with the OMX_BUFFERFLAG_ENDOFFRAME flag. The OMX component is expected ensure proper decoding in case of partial frames (e.g. to collect and assemble all the partial fragments before sending the frame to the corresponding decoder if the decoder expects to receive full input frames).

The expected outcome of the test is to verify the bit-exactness of the output file generated, so specifying the reference file is recommended.

The test case number is 15.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -r RefFileName -t 15 15

-r RefFileName though an optional argument here is strongly recommended.

All other arguments mentioned above in the example use case are mandatory.

Other mandatory arguments include

-m -> only required in case of mono bit-streams for MP3 and AAC components.

5.2.5. EXTRA_PARTIAL_FRAMES_TEST

This test is similar to the above partial frame test with a single variation.

An input frame is split into N partial fragments where N is kept greater than the number of input buffers allocated on the component's input port.

The client sends one fragment at a time to the component in an input buffer. The last input buffer containing the last fragment of the frame is marked with the OMX_BUFFERFLAG_ENDOFFRAME flag.

The idea here is to make sure that the component can assemble a partial frame without a deadlock (i.e. that the component does not hold – but rather returns input buffers so that client can send remaining pieces of the frame to complete sending one frame – without a deadlock).

The test is considered passed if there is no deadlock created and everything decodes properly. It also verifies the bit-exactness of the output file generated against the reference file specified; so specifying the reference file is recommended.

The test case number is 16.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -r RefFileName -t 16 16

-r RefFileName though an optional argument here is strongly recommended.

All other arguments mentioned above in the example use case are mandatory.

Other mandatory arguments include

-m -> only required in case of mono bit-streams for MP3 and AAC components.

5.2.6. INPUT_OUTPUT_BUFFER_BUSY_TEST

In this test case, the test client instantiates the OMX component, negotiates the parameters and starts the processing by placing the component in executing state. Then, the client stops sending the input & output buffers to the component at a specified frequency for some time before resuming sending input & output buffers again.

The component is expected to restart processing the data normally after the supply of input/output buffers has been started again.

The expected outcome of the test is again to verify the bit-exactness of the output file generated, so specifying the reference file is recommended.

The test case number is 17.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c h263 -r RefFileName -t 17 17

-r RefFileName though an optional argument here is strongly recommended.

All other arguments mentioned above in the example use case are mandatory.

Other mandatory arguments include

-m -> only required in case of mono bit-streams for MP3 and AAC components.

5.2.7. PAUSE_RESUME_TEST

In this test case, after processing N input buffers, the test client follows the following sequence of commands:

- Sends a state transition command to the component to go into executing ->pause state.
- Wait for the response from the component.
- Queue some more input and output buffers when in pause state.
- Resume processing of buffers by sending the pause->executing state transition command.

The value of N is specified as a #define value TEST_NUM_BUFFERS_TO_PROCESS.

The component is expected to restart decoding normally after the state has been changed back to executing without any loss of buffers in between.

The expected outcome of the test is again to verify the bit-exactness of the output file generated, so specifying the reference file is recommended.

The test case number is 18.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c aac -r RefFileName -t 18 18

-r RefFileName though an optional argument here is strongly recommended.

All other arguments mentioned above in the example use case are mandatory.

Other mandatory arguments include

-m -> only required in case of mono bit-streams for MP3 and AAC components.

5.2.8. FLUSH_PORT_TEST

In this test case, after processing N input buffers, the test client follows the following sequence of commands:

- Sends a flush command to the component on both its ports.
- Wait for the response from the component.
- Verify whether all the input and output buffers are returned from the component.
- Resume processing of buffers by sending back the input and output buffers.

The value of N is specified as a #define value TEST_NUM_BUFFERS_TO_PROCESS.

The component is expected to restart decoding normally after the flush command has been completed and the supply of buffers is resumed.

The expected outcome of the test is to verify that all the commands get executed properly with their respective callbacks reached back to client and test app doesn't encounter any dead-lock/crash condition in between. Specifying a reference file is not required in this test case, as the client cannot check for the bit-exactness of the output stream generated.

The test case number is 19.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c wma -t 19 19

All the arguments mentioned above in the example use case are mandatory.

Other mandatory arguments include

-m -> only required in case of mono bit-streams for MP3 and AAC components.

5.3. AVC Specific Tests

5.3.1. MISSING_NAL_TEST

In this test case, the test client skips some of the NAL units in an input bit-stream at a random frequency.

The expected outcome of this test is to ensure that code runs normally till the end of stream without crashing/abrupt exit and the client also follows the normal API flow.

One more additional verification added here is to count the number of output frames generated and to check that they should be greater than or equal to the total frames in that bit-stream minus the missing frames.

The test case number is 21.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c avc -t 21 21

All the arguments specified above in the example use case are mandatory.

There are no optional arguments required to run this test case.

NOTE: If the OMX component and the underlying decoded cannot handle missing NALs and cannot pass this test - then the corresponding “iOMXComponentCanHandleIncompleteFrames” capability flag described in [2] should be set to “OMX_FALSE”

5.3.2. CORRUPT_NAL_TEST

In this test case, the client corrupts the input bit stream by adding a random error of 1 bit to 8 bit in two consecutive bytes of the NALs occurring at a regular intervals.

Like the above MISSING_NAL_TEST test, the expected outcome of this test is also to ensure that code runs normally till the end of stream without crashing/abrupt exit and the client also follows the normal API flow. It also counts the number of output frames generated and

checks/verifies that they should be greater than or equal to the total frames in that bit-stream minus the corrupted frames.

The test case number is 22.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c avc -t 22 22

All the arguments specified above in the example use case are mandatory.

There are no optional arguments required to run this test case.

NOTE: If the OMX component and the underlying decoded cannot handle corrupt NALs and cannot pass this test - then the corresponding “iOMXComponentCanHandleIncompleteFrames” capability flag described in [2] should be set to “OMX_FALSE”

5.3.3. INCOMPLETE_NAL_TEST

In this test case, the client splits NALs into multiple input buffers. For a NAL split over multiple input buffers, this test case drops one buffer and sends the incomplete NAL to the component. The dropped piece of NAL could be somewhere in the middle of NAL or it could be the last piece of NAL containing the OMX_BUFFERFLAG_ENDOFFRAME flag.

The expected outcome of this test is to ensure that the underlying AVC decoder continues to decode till the end of bit-stream without crashing/abrupt exit and the client also follows the normal API flow.

This test also counts the number of output frames generated and verifies that they should be greater than or equal to the total frames in that bit-stream minus the incomplete frames.

The test case number is 23.

Example command line to run this test is

test_omx_client InputFileName -o OutFileName -c avc -t 23 23

All the arguments specified above in the example use case are mandatory.

There are no optional arguments required to run this test case.

NOTE: If the OMX component and the underlying decoded cannot handle incomplete NALs and cannot pass this test - then the corresponding “iOMXComponentCanHandleIncompleteFrames” capability flag described in [2] should be set to “OMX_FALSE”

6. Input and Output Bit-stream format

Most general test cases (except PARTIAL_FRAME_... test cases) specified above are written in such a way that they require the input stream to be parsed into frame boundaries.

The client then packs these full frame or whole number of multiple frames into an input buffer before sending it to the component. All these input buffers are also marked with the OMX_BUFFERFLAG_ENDOFFRAME flag at the end to indicate the completion of the frame.

Due to the above requirement, test framework can only accept the input stream to be in a specific format so that the parsing of frames is easy and require minimal complexity.

The sections below describe some of the special input format requirements and output format generation for different decoder components. These custom formats are required to run all the test cases.

To eliminate the need for file parsing – the input data for most input formats should be presented to the OMX decode test application in a format where 4 byte frame lengths are interleaved with frame data.

The OMX test application provides several sample files in the custom format in the data folder. To create more test files – one should observe the custom formatting rules described below.

6.1. WMA/WMV Component Input Bitstream Format

WMA/V specific test cases require the bit-stream to be in a format where every frame is preceded by a four byte header in little endian format. The 4-byte header represents the length of the frame (in bytes) that follows. The configuration data for WMV and WMA should also be preceded by such a 4-byte header.

This enables the test client to determine the size of each frame from the four bytes header. Thus, the entire full frame of data can be read from the test file and packed into an input buffer and sent to the component.

Input extension for wma and wmv files is typically “.rca” or “.rcv”.

6.2. AAC Component Input Bitstream Format

Similar to the wma/v test format explained above, the input bit-stream format for AAC should also include a four byte header in little endian format preceding every AAC frame. This 4 byte header should also precede the AAC configuration data. The header will enable the OMX Decoder Test Application to determine the size of the frame to follow.

This custom format will be required for all the test cases.

To obtain the raw aac bitstream with headers, there is a win32 utility “dump_aac.exe” that converts an mp4 file containing aac content or .aac file into the required format.

The command line for using this exe is:
dump_aac.exe -test 803 803 -source inputfilename.mp4

Several AAC files in this custom format readable by the omx decoder test application are provided in folder:

.../codecs_v2/omx/omx/omx_testapp/data

6.3. AMR Component Input Bitstream Format

The AMR format is slightly different from the other formats. It should have the following structure:

The first four bytes of the custom file indicate the AMR type based on the table below

Value	Format
0	OMX_AUDIO_AMRFrameFormatConformance
2	OMX_AUDIO_AMRFrameFormatFSF when the mode is NB (Narrow Band)
4	OMX_AUDIO_AMRFrameFormatIF2
6	OMX_AUDIO_AMRFrameFormatRTPPayload when the mode is NB (Narrow Band)
7	OMX_AUDIO_AMRFrameFormatRTPPayload when the mode is WB (Wide Band)
8	OMX_AUDIO_AMRFrameFormatFSF when the mode is WB (Wide Band)

After the first 4 bytes of the file, each data buffer is preceded by the following header structure:

Four bytes -> size of frame

Four bytes -> timestamp of the frame

followed by the buffer data equal to the size of frame determined above.

Here - each buffer data may contain multiple full frames as well. All the header values should be in little endian format.

Several AMR files in this custom format readable by the omx decoder test application are provided in folder:

.../codecs_v2/omx/omx/omx_testapp/data

6.4. AVC Input Bitstream Format

AVC input files (.264) must to be in the raw, i.e. so-called “ByteStream” format which precedes each NAL with the NAL start code.

6.5. MP3 Input Bitstream Format

OMX Decoder Test Aplication reads MP3 files directly (there is no specialized format required).

6.6. MPEG4/H263 Input Bitstream Format

OMX Decoder Test Application requires mpeg4/h263 bitstreams to be in the RAW .m4v format.