OMX Encoder 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 encoder components. Currently, the test application covers the following OpenMAX complaint audio/video encoders.

Video Encoders:

AVC / H.264

MPEG4 / H.263

Audio Encoders:

AMR NB

AAC

This test application can be easily extended further to add the support for others components 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 encoder test application is built along with other OpenCore code. The integration of 3rd party OMX cores and OMX components into the OMX encoder 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

3.1. Mandatory command line arguments

Following are the command line arguments required to run the OMX encoder test application:

ConfigFileName -c CodecType

where each of the above means:

ConfigFIle Name	Configuration text file containing the input stream name, output stream name and all the other encoding parameters required to encode the input stream. A sample configuration file for each encoder component is given in the .../codecs_v2/omx/omx_testapp_enc/data folder.
-c CodecType	Codec Type (following -c) should be specified. CodecType could be avc, mpeg4, amr or aac corresponding to the appropriate encoders.

These two arguments - specifying codec configuration file and codec type - are mandatory

3.2. Optional command line arguments

-t x y	A range of test cases to run from ‘x’ to ‘y’

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.

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. To run a single test case, x and y will be identical, equal to the test case number.

The test cases can be divided into following two categories.

1) Test case number 0 and 1 are the basic test cases that can be run on all the components and verify the basic configuration & setting of the component and some of the state transitions.

2) Test case numbers 2 to 9 are the general encoding tests that can also be run on all the components. These test cases verify the encoding of input bit-streams under several conditions.

See the section below to find out the individual test case number/index and their description.

4. Description of Test Cases

4.1. Basic Test Cases

These are the basic tests mostly related to negotiating initial configuration parameters with the component and also verifying some of state transitions.

4.1.1. 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 and a valid component handle is returned.

The test case number is 0.

Example command line to run this test is

test_omxenc_client ConfigFileName -c avc -t 0 0

4.1.2. PARAM_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.

Negotiate the different formats supported on the input and output port for audio/video domain like audio encoding, compression format, video color format etc.

Set some of these parameters back to the component and verify that they are set correctly.

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_omxenc_client ConfigFileName -c avc -t 1 1

4.2. General Test Cases

4.2.1. NORMAL_SEQ_TEST

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

-Initializing the component

-Negotiating and settings the encode parameters.

-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 test 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 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/failure condition in between. 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.

An output encoded bitstream, as specified in the configuration file, is generated by this test, which can be matched against the reference file if available.

The test case number is 2.

Example command line to run this test is

test_omxenc_client ConfigFileName -c mpeg4 -t 2 2

4.2.2. USE_BUFFER_TEST

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

Like the above test case, the flow of this test case is like this:

-Initializing the component

-Negotiating and settings the encode parameters

-Sending/receiving the input/output buffers till end of stream

-Sending the EOS stream marker flag and

-Finally stopping the component.

The test 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 the test is to verify that all the commands/api's get executed properly with their respective callbacks reached back to client and test app doesn't encounter any dead-lock/crash/failure condition in between. The test will give the end result as FAIL/SUCCESS.

An output encoded bitstream, as specified in the configuration file, is also generated by this test, which can be matched against the reference file if available.

The test case number is 3.

Example command line to run this test is

test_omxenc_client ConfigFileName -c mpeg4 -t 3 3

4.2.3. BUFFER_BUSY_TEST

In this test case, the test client instantiates the OMX component, negotiates the encode 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 normally after the supply of input/output buffers has been started again.

The expected outcome of the test is to verify that component behaves as expected with no failure/crashes in between for any api and generates the correct output bitstream.

The test case number is 4.

Example command line to run this test is

test_omxenc_client ConfigFileName -c avc -t 4 4

4.2.4. 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 to collect all the partial fragments before sending it to the corresponding encoder.

The expected outcome of the test is to verify that component is able to correctly assemble and send one full frame to the encoder and without a failure reported by the component/encoder. Also it is expected that all the commands / APIs are executed properly with their respective callbacks reached back to client.

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. An encoded output bit-stream will also be generated in this test case.

The test case number is 5.

Example command line to run this test is

test_omxenc_client ConfigFileName -c amr -t 5 5

NOTE: The PV framework will typically NOT split the encoder input into partial buffers and will send one full input video frame at a time. In case of audio (PCM) data – input buffers can in general contain any number of PCM samples. This test can be used optionally to test the ability of the OMX component to assemble partial frames.

4.2.5. 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 all the frames are encoded properly. The test will check the normal return values of different API's and callbacks and will give the end result as FAIL/SUCCESS based on those values. An output encoded bit-stream will also be generated that can be used to verify the encoding.

The test case number is 6.

Example command line to run this test is

test_omxenc_client ConfigFileName -c avc -t 6 6

NOTE: The PV framework will typically NOT split the encoder input into partial buffers and will send one full input video frame at a time. In case of audio (PCM) data – input buffers can in general contain any number of PCM samples. This test can be used optionally to test the ability of the OMX component to assemble partial frames.

4.2.6. 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 in the code. The value in the code can be modified if necessary to test various scenarios.

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

The test will check the normal return values of different API's and callbacks and will give the end result as FAIL/SUCCESS based on those values.

The test case number is 7.

Example command line to run this test is

test_omxenc_client ConfigFileName -c avc -t 7 7

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

4.2.7. ENDOFSTREAM_MISSING_TEST

This test case is also similar to the NORMAL_SEQ_TEST, 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 problems (e.g. deadlocks). The test will give the end result as FAIL/SUCCESS.

The test case number is 8.

Example command line to run this test is

test_omxenc_client ConfigFileName -c amr -t 8 8

4.2.8. WITHOUT_MARKER_BIT_TEST

In this test case, instead of sending one full input frame with marker bit (i.e. OMX_BUFFERFLAG_ENDOFFRAME flag) set, the client sends input buffers containing a fixed chunk of data equal to the input buffer size to the component without marking the (OMX_BUFFERFLAG_ENDOFFRAME flag.

The expected outcome of the test is to verify that the component and encoder can handle this chunk of data and encode the bit-stream normally. The output bitstream generated can also be used to verify the test case.

The test case number is 9.

Example command line to run this test is

test_omxenc_client ConfigFileName -c amr -t 9 9

NOTE: The PV framework will typically NOT split the encoder input into partial buffers and will send one full input video frame at a time. In case of audio (PCM) data – input buffers can in general contain any number of PCM samples. This test can be used optionally to test the ability of the OMX component to handle arbitrary data in input buffers.

5. Input and Output Bit-stream format

In all of the general test cases described above (except “WITHOUT_MARKER_BIT_TEST” or tests that test partial frame assembly), client packs one full frame or whole number of multiple frames into an input buffer before sending it to the component and all these input buffers are also marked with the OMX_BUFFERFLAG_ENDOFFRAME flag at the end to indicate the completion of the frame.

In the “WITHOUT_MARKER_BIT_TEST”, the client sends a chunk of input data equal to the allocated input buffer length to the component. It is the responsibility of the component to send that input buffer to the encoder in an appropriate format, either by assembling/splitting the data or forwarding the same chunk.

A configuration file has to be given as an input argument to the test application. It is a text file that contains the information about input bitstream to encode, output bitstream to generate and all the encoding parameters that are to be passed to the component and the underlying encoder.

The sample configuration files and the input bit-streams as reference are given in the data folder of the test application. Users can write their own configuration files based on the sample ones to vary the parameters and to encode with different bit-streams.

The following sub-sections explain the same by taking an example of each of the encoder component and its configuration file.

5.1. AMR Encoder Component Configuration File

The configuration file that is required by the test application to run the amr encoder component contains the following information:

-Amr Input File Name

-Amr Output File Name

-Input bits per sample

-Input Sampling Frequency

-Number of channels

-Output band mode - 0 to 7

-Output format - if2 and fsf

The test application reads this configuration file line by line and stores these parameters in internal variables, so the order of these lines has to be strictly maintained.

These are the most common encoding parameters that are required by the encoder to encode a bitstream. Client sends these parameters via Get/Set parameter api to the component in the Loaded state so that component can initialize the underlying encoder. Afterwards a Loaded->Idle state transition is requested by the client.

5.2. AVC Encoder Component Configuration File

The configuration file that is required by the test application to run the avc encoder component contains the following information:

-Input file name

-Output file name

-Input width

-Input height

-Input source frame rate

-Input supported color format - RGB24, RGB12, YUV420

-Target frame width

-Target frame height

-Target bit rate

-Target frame rate

-Num of P Frames between two I frames

-Supported Rate Control Type - CONSTANT_Q, VBR or CBR

-Initial P-frame QP

-Search range around the MB origin

-Intra refresh type - Cyclic, Adaptive or Both

-Number of guaranteed I-MB in a frame, when intra refresh type is Cyclic or Both

-Profile - Baseline, Main, Extended, High, High10, High422, High444

-Level - 1, 1b, 11,12,13,2,21,22,3,31,32,4,41,42,5,51

-LoopFilterType - Enable, Disable or DisableSliceBoundary

-Constrained I prediction flag – 0 or 1

-Enumerated motion vector Accuracy - Pixel, HalfPel, QuarterPel, EighthPel

The test application reads this configuration file line by line and stores these parameters in internal variables, so the order of these lines has to be strictly maintained.

Like AMR, these are also the most common encoding parameters that are required by the avc encoder to encode a bitstream. Client sends these parameters via Get/Set parameter api to the component in the Loaded state so that component can initialize the underlying encoder. Afterwards a Loaded->Idle state transition is requested by the client.

5.3. MPEG4/H.263 Encoder Component Configuration File

The configuration file that is required by the test application to run the mpeg4/h263 encoder component contains the following information:

-Input file name

-Output file name

-Input width

-Input height

-Input source frame rate

-Input supported color format – RGB12, RGB 24, YUV420 etc

-Target frame width

-Target frame height

-Target bit rate

-Target frame rate

-Num of P Frames

-Content type - 0 for Mpeg4 and 1 for H.263

-Rate control type - CONSTANT_Q, VBR or CBR

-Initial I-frame QP

-Initial P-frame QP

-Search range around the MB origin

-mv8x8 flag – 0 or 1

-Intra refresh type - Cyclic, Adaptive or Both

-Number of guaranteed I-MB in a frame, when intra refresh type is Cyclic or Both

-Packet size in bytes

-Profile - Simple, SimpleScalable, Core, CoreScalable etc

-Level - 0,1,2,3 etc

-Resync Enable Flag – 0 or 1

-Data Partioning Flag – 0 or 1

-Short Header mode Flag – 0 or 1

-Resynch markers interval (in bits)

-Reversible variable length coding Enable Flag – 0 or 1

-The number of ticks per second in Mpeg4 bitstream

-GOB Header Interval in case of H.263 and Short header mode

The test application reads this configuration file line by line and stores these parameters in internal variables, so the order of these lines has to be strictly maintained.

Like the above two components, these are also the most common encoding parameters that are required by the mpeg4/h263 encoder to encode a bitstream. Client sends these parameters via Get/Set parameter api to the component in the Loaded state so that component can initialize the underlying encoder. Afterwards a Loaded->Idle state transition is requested by the client.