Embedded OpenMAX Driver Robustness

Sarfaraz Shaikh[1]

[1]*V.V.P.I.E.T, Solapur University, Solapur, India*

Abstract – *OpenMAX stands for Open Media Acceleration. OpenMAX is an cross platform based standard for the development of multimedia application. It is used across embedded industry for multimedia application development across multiple platform. OpenMAX is mainly used in video, audio, image processing industry. In this paper we will be discussing different methods on making the OpenMAX APIs robust so that the fail rate of the multimedia drivers can be reduced and the development of device driver can be accelerated. We will also discuss different layers of OpenMAX and APIs of IL layer in details.*

Key Words: OpenMAX, OMX, Video Decoder, Video Encoder, API, Audio Systems, Image Processing, Video Processing.

1. INTRODUCTION

OpenMAX is free and royalty free cross platform set of API in C language. It is maintained by Khronos group. We will be considering the review paper "OpenMAX in Embedded Systems". This paper describes the basics of OpenMAX, its different layers. OpenMAX consists of three major layers Application layer, Integration Layer, and Development Layer. The tool that we will be developing for the testing of OpenMAX API will be based on CPP open source test framework.

1.1 Gtest

Google test also known as Gtest is a unit testing library developed on the lines of cpp, it is based on the xUnit Architecture. The license is based on BSD 3 class and can be used ofr free. It can be compiled for POSIX and Windows based platforms. This library makes the debugging process very easy and makes the development of testing very fast.

Most of the test framework is classified as unit, integration, functional. Gtest is divided into three category of tests, small tests, medium tests, and large scale tests.

1.1.1 Small Tests (Unit Tests)

Small tests are mostly single function and module. They are mostly automated for the testing. The main target of these tests is to test the functional issues, error conditions, data corruption if any. Small test generally require a test environment to be set up. Using small tests a typical test can be targeted rather than testing the whole system.

1.1.2 Medium Tests (Integration Tests)

Medium tests involve more functionality of the software to be tested. They are mostly automated. Most of the features of the system under test interact between each other, this kind of functionality can be tested under medium tests. The main aim is to test the interaction of two blocks of system with each other as expected.

1.1.3 Large Tests (Acceptance Tests)

Large tests mostly target real world scenarios with real user requirements and data inputs. Large tests mainly target three or more features. Large tests are result driven tests, the main aim is to satisfy the user needs. The product should operate in the real word data after this testing.

2 OpenMAX Software Landscape

The main aim of the framework is to test the Driver level API of OpenMAX IL layer. That means our aim is to develop an framework which ill make a direct call to the IL layer API and test their return errors and missing params.

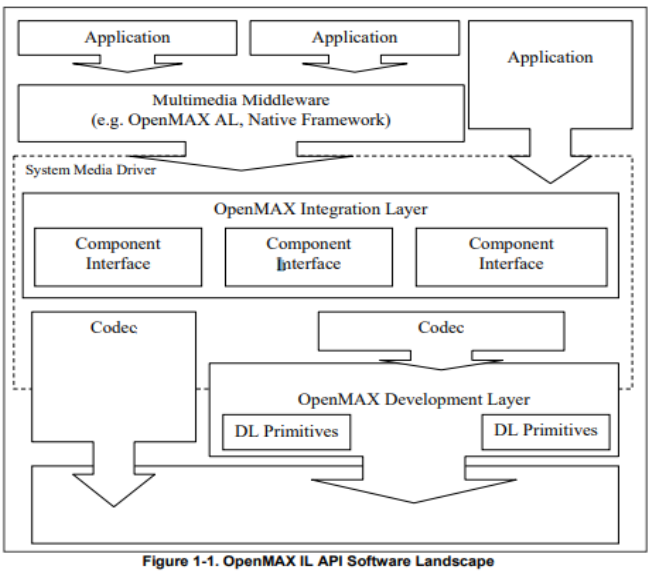


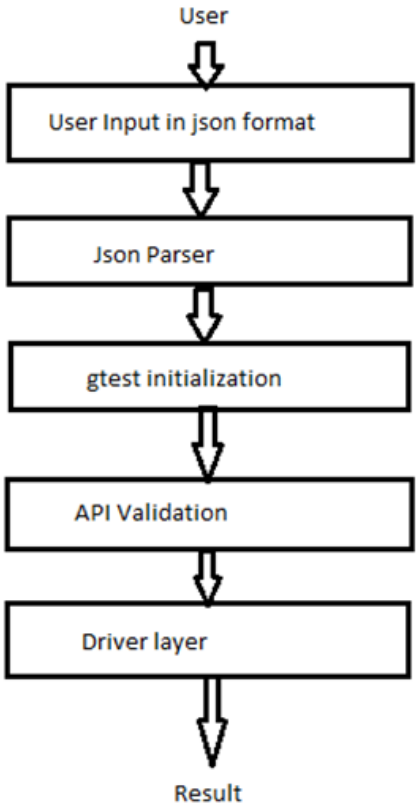
Figure 1-1. OpenMAX IL API Software Landscape

The OpenMAX has three layers, DL being the lowest layer and AL being the highest application layer. DL directly interacts with the Hardware while IL is the part of the Driver Software. Most of the application makes use of AL layer to interact with the Driver. But there is provision of bypassing the AL layer and user can directly interact with the IL layer. Our test framework will be based on this approach of

bypassing the AL layer and directly interacting with the IL layer API.

2.1 Test Framework Block Diagram

Below block diagram explains our approach for the test framework development.



2.2 Test Framework Flow

The first block in our framework is Json input block. End user needs to give input to the framework like which API to test and different parameters to that API. There can be multiple return values that needs to be validated from the API, all these input information can be provided to framework using json format. Jason format is easy to understand and also easy to parse at machine level.

Next block in the design is json parser. The input provided by the end user will be parsed. Like the API name will be stored in its proper structure. The return value will be stored. The input parameters will be stores and validated by the framework.

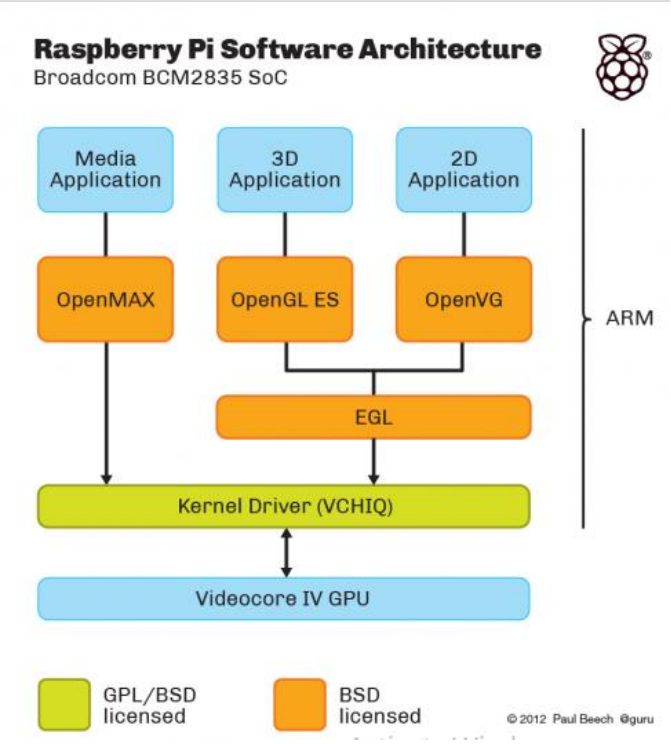
The next block is the main gtest block. This block is coded into cpp for the object oriented nature and ease of use. This is the open source framework available from Google. This block will contain individual file for each API supported by the OMX standard. We will skip the validation for the API not supported by our hardware GPU.

API Validation block will be responsible for calling the driver level API and verifying the return values and checking if any unhandled parameter is passed to driver OMX layer. This layer should be capable of handling exceptions raised from driver side.

The Driver layer is the part of hardware GPU which we will be validating in our framework. We will not make any changes to the driver code. We will only verify the API and check for the robustness of API.

3. Raspberry PI OMX Architecture

Below is the block diagram for GPU on Raspberry PI board. Raspberry PI is a mini computer available at cheaper price and mostly used for multimedia applications.



The above block diagram makes clear that OpenMAX layer is Raspberry PI can directly interact with Kernel Driver for video core. The video core is capable of image, audio, video processing. So our robustness framework can be used for any OpenMAX API supported by Raspberry PI. To enable the video core on Raspberry PI we require a camera connection on the board and initialization of the same.

4. Results

The basic API is OMX_Init() this API must return OMX_ErrorNone if the return value is other than this then the video core is failed to initialize. If the unexpected value is returned then the issue can be raised with the maintainers of OMX_Init(). The time taken for testing this API is around 5.43ms on Raspberry PI board. Time on other platforms may vary.

Same is the case with OMX_DeInit(), this API uninitialized the OMX component. If the return value is incorrect then the Driver has failed to properly uninitialized the Driver part.

Time taken for testing this API is around 6.5ms on Raspberry PI. Time constrains are not available for other Opensource Drivers.

For OMX_GetHandle() this important API to start using any component within OMX framework, Before doing any decoding or encoding using OMX driver the user must ensure that the handle is obtained properly from hardware.

This API's return value can be checked for failure. Time taken for testing this API is around 6.83ms.

Multiple Driver issues filed using the robustness framework in different GPU OMX Drivers.

5. CONCLUSION

OPENMAX finds multiple application for the development of Multimedia application in Embedded systems. OpenMAX is widely integrated in mobile systems and TV based systems. Integration of OpenMAX helps developers of different multimedia domain to form cross platform applications which can be ported on different architecture and operating systems.

6. ACKNOWLEDGEMENT

We highly acknowledge the Khronos for properly maintaining the documentations and updating the documents for OpenMAX.

RaspberryPI open forums were helpful in getting started with the initial setup of the board and camera.

REFERENCES

[1] J. Barba, D. de la Fuente, F. Rincon, Member, IEEE, J.C. Lopez, member, IEEE "OpenMAX" Hardware Native Support for Efficient Multimedia Embedded System".

[2] Pablo Penil, Pablo Sanchez University of Cantabria " UML/MARTE Methodology for Automatic System Code Generation of OpenMAX Multimedia Applications".

[3] OpenMAX IL Specifications from Khronos Group.

[4] OpenMAX DL Specification from Khronos Group.

[5] OpenMAX AL Specification from Khronos Group.