OpenMAX™

Adaptive Digital Technologies, Inc.

High level overview of OpenMAX

OpenMAX (OMX) is a set of programming interfaces that promotes standardized, cross-platform multimedia development. It offers API's at the development, integration, and application layers for a consistent development experience throughout the entire multimedia stack.

The Development Layer (DL) provides a set of general signal processing, audio, video and imaging functions that are commonly used in multimedia codecs, assisting codec vendors in delivering optimized products across different hardware architectures.

Example:

```
/* Here is an example of a signal processing function included in the OpenMAX
Development Layer */

/* This function computes the median over a region of an array */
OMXResult omxSP_FilterMedian_S32(const OMX_S32 *pSource, OMX_S32 *pDestination,
       OMX_INT length, OMX_INT maskSize);

/* Assume input data is passed in, and length = 256 */
int useOMXFilterMedian(OMX_S32 *input_data, OMX_S32 *medians) {

       result = omxSP_FilterMedian_S32(input_data, medians, 256, 0);
}

/* A Hardware Vendor could implement and optimize the omxSP_FilterMedian_S32
function ahead of codec development in order to speed up development and
optimization time */
```

The Integration Layer (IL) is a set of C-language media component interfaces. The components are usually audio, video, and imaging codecs, but any sort of data processing is supported. Components at this level can rely on optimized development layer functions while maintaining compatibility with other OpenMAX-supported hardware platforms.

This layer allows for a platform provider to support a wide range of multimedia codecs through a common interface. This allows codecs to be added, removed, or upgraded, all without effecting any other part of the system. In addition, codecs can be easily connected and centrally controlled to satisfy almost all use cases. More information, such as the specification and interface header files can be found at the Khronos Group OMX IL API Registry [http://www.khronos.org/registry/omxil/].

Example:

```
/*
 * This is an example of a function typical to a OpenMAX Integration
 * Layer client. It shows how codec abstraction in the form of
 * components can reduce development time and program complexity.
 * Please assume the 'read_input' and 'write_output' functions
 * represent actual program I/O.
 */


OMX_U32 process_data_with_component(OMX_HANDLETYPE currentOMXComponent) {

	OMX_ERRORTYPE omxErr;

	/* set up component parameters, input/output details, etc */
	/* ... */

	while (!outOfData) {

		/* Read in Input Data */
		read_input(inBuffer);

		/* Send Input Data Buffer to OpenMAX Component to empty */
		omxErr = OMX_EmptyThisBuffer(currentOMXComponent, inBuffer);

		if (omxErr != OMX_ErrorNone) {
			OMX_Error_Message("Empty this buffer error!");
			return omxErr;
		}

		/* Send Output Data Buffer to OpenMAX Component to fill */
		omxErr = OMX_FillThisBuffer(currentOMXComponent, outBuffer);

		if (omxErr != OMX_ErrorNone) {
			OMX_Error_Message("Fill this buffer error!");
			return omxErr;
		}

		/* Write Output Data */
		write_output(outBuffer);
	}
	return OMX_ErrorNone;
}
```

The Application Layer (AL) was designed to allow application developers a standardized, cross-platform interface to Integration Layer components for multimedia recording and playback. It is a C-language API that has been specifically designed to be useful on resource-constrained mobile and embedded devices while still including comprehensive functionality.

An example of this being used is provided by Google, who added support for OpenMAX AL in the Android 4.0 Native Development Kit. This will further increase the number of developers familiar with OpenMAX API's and interested in writing reusable multimedia code.

Example:

```
/*
 * This is an example of a function that creates an audio recorder
 * using the OpenMAX Application Layer API.  In this example the
 * audio recorder is set up to use a buffer queue data structure
 * to store the recorded audio.
 */



SLresult createAndStartAudioRecorder(void)
{
    SLresult result;

    /* Configure Audio Source */
    SLDataLocator_IODevice audio_input = {SL_DATALOCATOR_IODEVICE,
       SL_IODEVICE_AUDIOINPUT, SL_DEFAULTDEVICEID_AUDIOINPUT, NULL};
    SLDataSource audioSource = {&audio_input, NULL};

    /* Configure the audio output (sink) to be a buffer queue */
    SLDataLocator_SimpleBufferQueue recordBufferQueue =
       {SL_DATALOCATOR_SIMPLEBUFFERQUEUE, 2};
    SLDataFormat_PCM format_pcm = {SL_DATAFORMAT_PCM, 1, SL_SAMPLINGRATE_16,
        SL_PCMSAMPLEFORMAT_FIXED_16, SL_PCMSAMPLEFORMAT_FIXED_16,
        SL_SPEAKER_FRONT_CENTER, SL_BYTEORDER_LITTLEENDIAN};
    SLDataSink audioSink = {&recordBufferQueue, &format_pcm};

    /* Create audio recorder */
    const SLInterfaceID id[1] = {SL_IID_SIMPLEBUFFERQUEUE};
    const SLboolean req[1] = {SL_BOOLEAN_TRUE};
    result = (*engineEngine)->CreateAudioRecorder(engineEngine,
       &recorderObject, &audioSource, &audioSink, 1, id, req);

    /* Get the audio record interface */
    result = (*recorderObject)->GetInterface(recorderObject, SL_IID_RECORD,
       &recorderRecord);

    /* Get the audio recorder buffer queue interface */
    result = (*recorderObject)->GetInterface(recorderObject,
       SL_IID_SIMPLEBUFFERQUEUE, &recorderBufferQueue);

    /* Set callback function to be called when a buffer is full */
    result = (*recorderBufferQueue)->RegisterCallback(recorderBufferQueue,
       bqRecorderCallback, NULL);

       /* enqueue an empty buffer to be filled by the recorder */
    result = (*recorderBufferQueue)->Enqueue(recorderBufferQueue,
       recorderBuffer, RECORDER_FRAMES * sizeof(short));

       /* Start Recording */
    result = (*recorderRecord)->SetRecordState(recorderRecord,
       SL_RECORDSTATE_RECORDING);

    return SL_RESULT_SUCCESS;
}
/* Once recording has started, the buffer queue callback function will be
called as soon as a recording buffer fills with data */
```

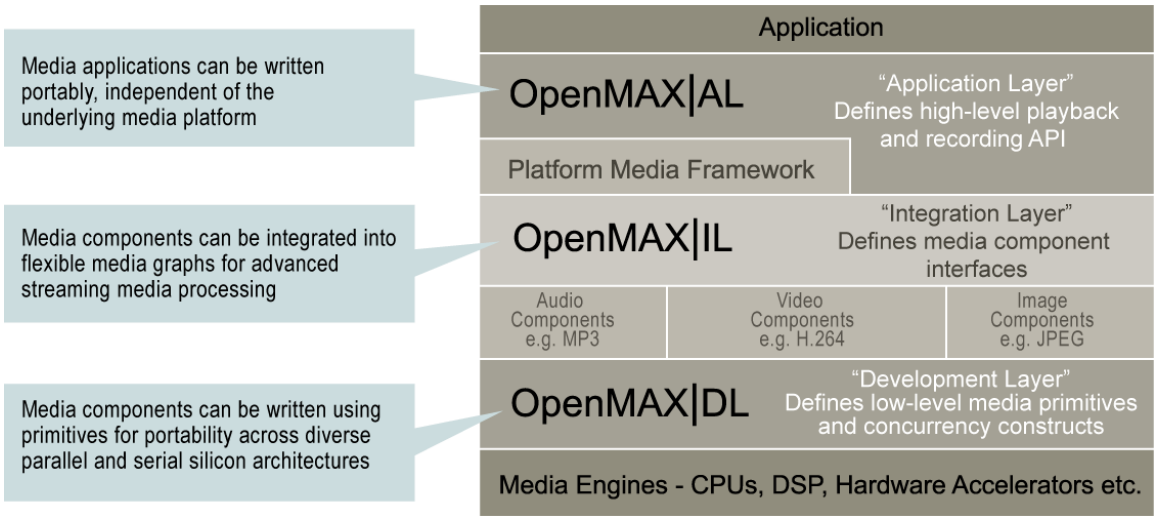

Diagram Source: http://www.khronos.org/openmax/il/

Specific TI OMX Implementation

Adaptive Digital has a unique OpenMAX Integration Layer implementation that is used in TI products such as the Blaze™ development platform. Our system follows a majority of the OMX Integration Layer specification but deviates in several key ways. There are additional header files containing new parameters, and several parts of the system have been modified to support the eXpressDSP™ Digital Media (xDM) standard. This allows the system to benefit from the OpenMAX design while reusing previously-written codecs that adhere to the xDM standard.

Sources

OpenMAX IL - The Standard for Media Library Portability http://www.khronos.org/openmax/il/

OpenMAX - Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/OpenMAX