White Paper

Muxing with Intel® Media Software Development Kit

June, 2011

Abstract: This article presents a method that can be used to perform muxing of encoded media generated by Intel® Media SDK. The article details how to use audio/video codec components part of Intel® Integrated Performance Primitives (Intel® IPP) samples to create a muxing solution based on the Intel® Media SDK "sample_encode" project.

Author: Petter Larsson (Intel Corporation)

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:
http://www.intel.com/design/literature.htm

Contents

Table of Contents

Introduction

The Intel® Media Software Development Kit (Intel® Media SDK) is a software development library that exposes the media acceleration capabilities of Intel platforms for video decoding, video encoding, and video pre/post processing. Intel® Media SDK helps developers rapidly develop software that accesses hardware acceleration for video codecs with automatic fallback on software if hardware acceleration is not available.

Intel® Media SDK is available free of charge and can be downloaded from here: www.intel.com/software/mediasdk/ (Gold release targeting current generation of platforms and Beta release targeting features for next generation platforms)

Intel® Media SDK is delivered with a wide range of application samples that illustrate how to use the SDK to encode, decode and transcode media to/from elementary video streams. Aside from the binary DirectShow* filters supplied with Intel® Media SDK, the SDK does not provide the capability of muxing encoded H.264 or MPEG-2 frames into common media containers.

This article presents a method that can be used to perform muxing (into mp4 and mpg containers) of encoded media generated by Intel® Media SDK. The article details how to use audio/video codec components part of Intel® Integrated Performance Primitives (Intel® IPP) samples to create a muxing solution based on the Intel® Media SDK "sample_encode" project.

"Muxing" is the process of, according to container type standards, combining video and audio frames with corresponding time stamps and media meta data into a file that can be played back by common media player tools such as Windows* Media Player or VLC*. While audio encoding and audio muxing is outside the scope of this article, a similar approach could be used to add that functionality.

Building the solution described in this article requires the following components:

- Intel® Media SDK 2.0
  www.intel.com/software/mediasdk/
- Intel® Integrated Performance Primitives (Intel® IPP) 7.0 (or later) for Windows*
  (http://software.intel.com/en-us/articles/intel-ipp/)
- Intel® Integrated Performance Primitives – Samples for Windows* (Main Samples)
  http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-samples-license-agreement/

In addition, the developer also needs Microsoft Visual Studio* and Microsoft Windows* SDK. (Microsoft Visual Studio* 2008 and Microsoft Windows* SDK 7.0 were used when building this solution; We suggest developers use the same configuration to minimize changes)

*Note: The same solution could easily be migrated to Intel® Media SDK 3.0 beta samples.

Download the Source Code: http://software.intel.com/file/37473

Overview

The following figure illustrates the process of encoding and muxing video frames into a media container. Note that audio encoding and muxing is out of scope of this article. However, a developer could quite easily extend the capability to also handle muxing of encoded audio samples into a container.

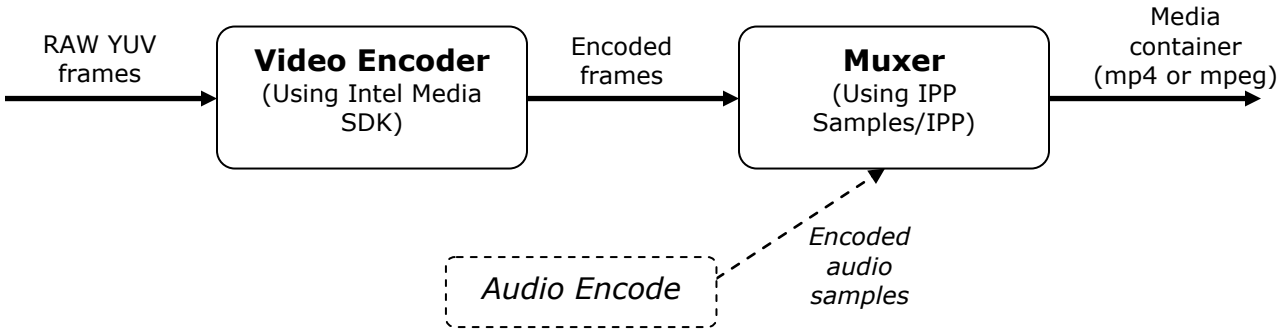

Figure 1 - Overview of Encode – Muxing process

The solution detailed in this article was created using Intel® Media SDK 2.0 with corresponding video encode console sample, "sample_encode". The solution also depends on the use of Intel® IPP with adjoining audio/video codec samples.

The below sections describe the components used as part of the solution in more detail.

Intel® Media SDK

Intel® Media SDK supports hardware accelerated and software optimized media libraries for video encode, decode and processing functionality on Intel platforms. The optimized media libraries are built on top of Microsoft DirectX*, DXVA APIs and platform graphics drivers. Intel® Media SDK exposes the HW acceleration features of Intel® Quick Sync Video (Intel® QSV) built into 2nd generation Intel® Core™ processors. http://www.intel.com/technology/quicksync/index.htm

The figure below provides a high level overview of where Intel® Media SDK fits into the software stack.

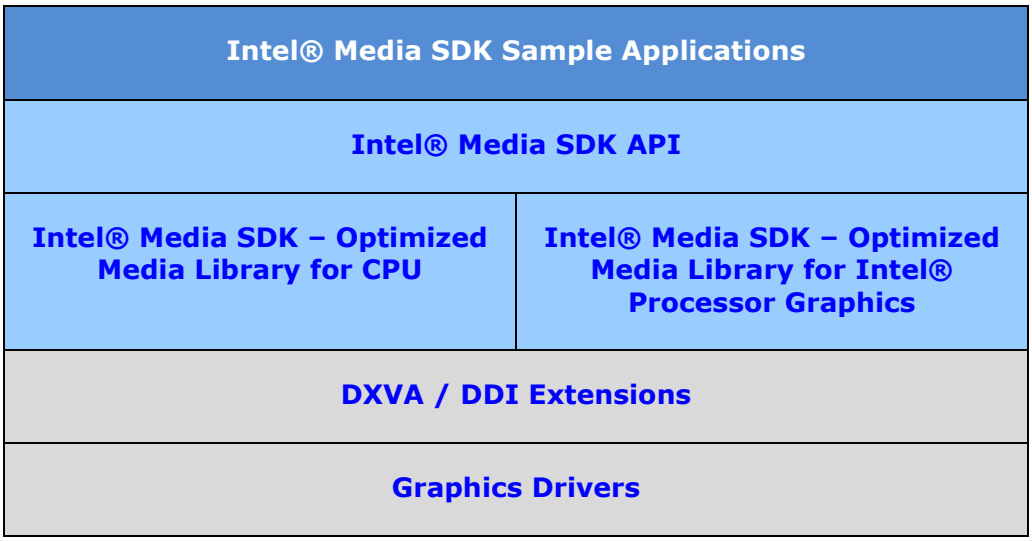

Figure 2 - Overview of Intel® Media SDK

For extensive details on all the features of Intel® Media SDK please refer to the manual provided with the SDK.

Intel® Integrated Performance Primitives (Intel® IPP)

Intel® Integrated Performance Primitives (Intel® IPP) is an extensive library of multi-core-ready, highly optimized software functions for digital media and data-processing applications. Intel® IPP offers thousands of optimized functions covering frequently-used fundamental algorithms.

The solution described in this article utilizes Intel® IPP 7.0 as part of the Intel® IPP sample code muxing components.

The following picture gives an overview of the capabilities of Intel® IPP.

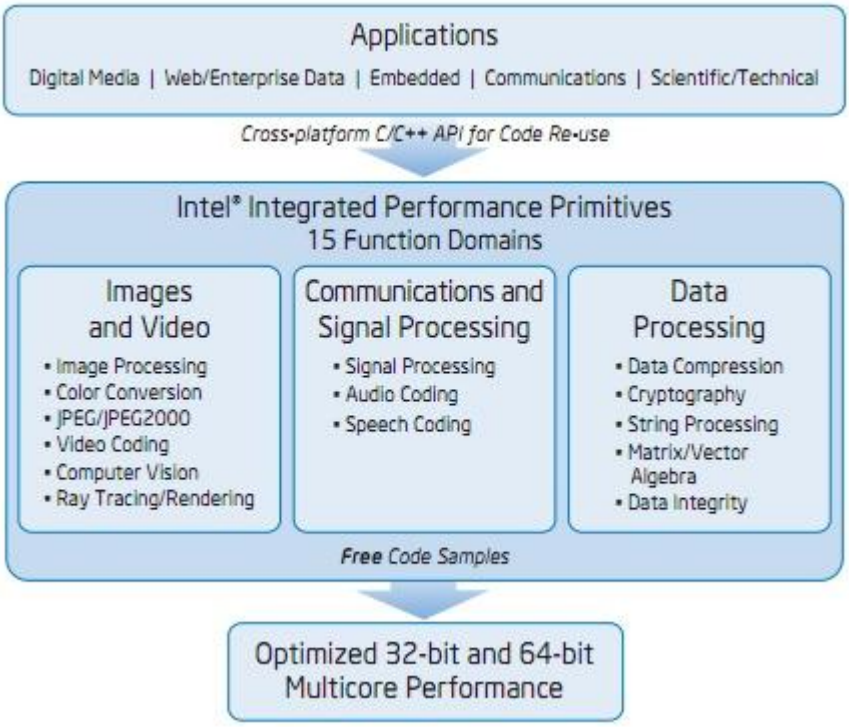

Figure 3 - Overview of Intel® IPP

For further details on the Intel® IPP licensing terms, please refer to the Intel® IPP product page or the following link:
http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-faq/

Intel® IPP Samples

This article uses the audio/video codec samples provided as a separate download with the core Intel® IPP package. Only the "audio-video-codecs" sub folder part of the samples is used in the following solution.

Inside the "audio-video-codecs" sub folder there is a "doc" folder containing a document with further details on how to use the range of sample components (including the mp4 and mpeg muxers used in this article).

Implementation

In this chapter we will detail all the steps required to implement muxing support for encoded Intel® Media SDK MPEG-2 and H.264 streams. This includes instructions on how to make the required changes to build projects, library/include file dependencies, source code and how to build the Intel® IPP sample components.

*Note that muxing could also be performed after complete Intel® Media SDK encode using third party tool such as ffmpeg, but this naturally allows for much less flexibility. For instance, to mux H.264 elementary stream into mp4 container the following ffmpeg command could be used: "`ffmpeg –f h264 –i stream.264 –vcodec copy out.mp4`"

Building Intel® IPP sample muxer components

First we will build the Intel® IPP audio video codec sample components (these components are all in the "UMC" name space) that we will access as libraries in our solution.

Make sure you have installed at least version 7.0 of Intel® IPP (or Intel® Composer XE that includes Intel® IPP) and the Intel® IPP samples.

Also please set "IPPROOT" system environment variable to the Intel® IPP root folder in your file system. For instance: `IPPROOT = C:\Program Files\Intel\ComposerXE-2011\ipp\`

Execute the following steps to build the Intel® IPP sample components required for the muxing capability:

1) Open Command prompt
   (depending on where you execute from you may have to open in Administrative mode)
2) Change directory to Intel® IPP environment setup folder
   For instance: `C:\Program Files\Intel\ComposerXE-2011\ipp\bin\`
3) Execute the Intel® IPP environment script
   `ippvars.bat <arch>`
   (where <arch> is "ia32" or "intel64")
4) Change directory to Intel® IPP samples audio video codecs folder
   `<location of Intel® IPP samples>\audio-video-codecs\`
5) In this folder you will find a Makefile. Modify the Makefile compile flag M_FLAG and set it to "/MTd" instead of the default "/MD". (Multi-threaded debug linked library)
   (This is required to match the project settings of the Intel® Media SDK sample_encode project we base the development on)
6) Run the build script
   For 32 bit: `build_ia32.bat cl9`
   For 64 bit: `build_intel64.bat cl9`
   "cl9" indicates the version of Microsoft Visual Studio*. In the example above "9" indicates version 9 of the tool, which equals Microsoft Visual Studio* 2008. *Modify according to your setup (there is more info on this in \audio-video-codecs\ReleaseNotes.htm).*
7) Note that the build is somewhat time-consuming. After compilation make sure all the base libraries compiled ok; don't worry if the application projects failed to compile.
   The binary libraries are created in:
   `<location of Intel® IPP samples>\audio-video-codecs\_bin\<arch>_cl9\lib\`

Set "IPPSAMPLESROOT" system environment variable to the root folder of the Intel® IPP samples. For instance: `IPPSAMPLESROOT = C:\Program Files\Intel\MediaSDK\ipp_samples`
The folder "audio-video-codecs" should be located in the folder of the above path.

Integrating muxer components with encoder

Note that the code below follows the same programming conventions and macros from the existing Intel® Media SDK samples. Please refer to the Intel® Media SDK sample code for further details.

For exact details on the code changes please refer to the provided Microsoft Visual Studio* solution/projects.

Changes to Microsoft Visual Studio* "sample_encode" project

Since the following solution is based on the existing "sample_encode" project part of the Intel® Media SDK sample code, please first make a backup copy of the "sample_encode" and "sample_common" project folders before making the changes.

Make sure environment variables IPPROOT and IPPSAMPLESROOT are set as described in the previous chapter.

Please add the following to your Microsoft Visual Studio* project target configuration (or configurations if you plan to compile for several targets) part of the "sample_encode" solution (.sln file).

Add the following to your list of project (sample_common and sample_encode) include directories:

```
$(IPPSAMPLESROOT)\audio-video-codecs\codec\mpeg2_mux\include
$(IPPSAMPLESROOT)\audio-video-codecs\codec\mpeg4_mux\include
$(IPPSAMPLESROOT)\audio-video-codecs\io\media_buffers\include
$(IPPSAMPLESROOT)\audio-video-codecs\io\umc_io\include
$(IPPSAMPLESROOT)\audio-video-codecs\core\vm\include
$(IPPSAMPLESROOT)\audio-video-codecs\core\umc\include
$(IPPSAMPLESROOT)\audio-video-codecs\core\vm_plus\include
$(IPPROOT)\include
```

Add the following to your list of project (sample_encode) library directories:

```
$(IPPSAMPLESROOT)\audio-video-codecs\_bin\ia32_cl9\lib
```

(Replace ia32_cl9 with appropriate folder name reflecting your architecture and tool version, as described in the previous chapter)

```
$(IPPROOT)\lib\ia32
```

(ia32 or intel64 depending on selected architecture)

Add the following to your list of project (sample_encode) library dependencies:

```
mpeg4_mux.lib
mpeg2_mux.lib
umc_io.lib
vm.lib
umc.lib
media_buffers.lib
```

```
common.lib
vm_plus.lib
ippcore_l.lib
ippdc_l.lib
```

Add the following to your list of project (sample_encode) ignored libraries:

```
ippcore.lib
ippdc.lib
```

For this specific solution we are using statically linked of Intel® IPP libraries which will make the executable a bit larger, but also more portable, compared to a solution using Intel® IPP DLL.

Usage scenario / command-line input

To expose the new muxing feature we add a new "-mux" command line option.

Using this new option we encode raw YUV video into a H.264 video stream muxed into a mp4 container with the following example command:

```
sample_encode.exe h264 –i raw.yuv –o out.mp4 –w 640 –h 480 –mux
```

*Note: The same options available as part of the original encode sample are also available in the new muxer solution.

If the encode target is set to "mpeg2" instead of "h264" the output container will be of the type mpeg.

The following code in "sample_encode.cpp" implements support for the new option:

```
…
else if (0 == _tcscmp(strInput[i], _T("-mux")))
{
        pParams->bMux = true;
}
…
```

Integration of new Muxer class

The muxing capability is integrated into the exiting sample_encode project by sub classing the `CSmplBitStreamWriter` class. The new subclass is named `MuxWriter` and it implements the complete mp4 and mpeg muxing functionality.

To integrate with the new muxer class the following changes are made to "pipeline_encode.cpp" and "pipeline_user.cpp". The code checks if the muxing option is enabled and instantiates the appropriate object type:

```
if(!pParams->bMux)
{
        m_pFileWriter = new CSmplBitstreamWriter();
```

```
        sts = m_pFileWriter->Init(pParams->strDstFile);
    }
    else
    {
        m_pFileWriter = new MuxWriter();
        sts = static_cast<MuxWriter*>(m_pFileWriter)->Init(
                    pParams->strDstFile,
                    pParams->nDstWidth,
                    pParams->nDstHeight,
                    pParams->dFrameRate,
                    pParams->nBitRate,
                    pParams->CodecId);
    }
    CHECK_RESULT(sts, MFX_ERR_NONE, sts);
```

As can be seen in the code above, the file writer object has changed to a pointer type instead of a directly accessed object in "pipeline_encode.h".

```
    CSmplBitstreamWriter *m_pFileWriter;
```

Also note the new muxing related parameters required by the `MuxWriter` class.

Muxer class

The majority of the code changes are in the "sample_common" project. The new muxer class `MuxWriter` is a sub class of the `CSmplBitStreamWriter` and is implemented in the existing "sample_utils.cpp" file.

In the sample_utils.h include file we must first include all the required header files from the Intel® IPP samples audio video codec components package.

```
    #include "umc_structures.h"
    #include "umc_muxer.h"
    #include "umc_mp4_mux.h"
    #include "umc_file_writer.h"
    #include "umc_mpeg2_muxer.h"
    #include "umc_mpeg2_muxer_chunk.h"
```

Then we need to define the new `MuxWriter` class as follows:

```
    class MuxWriter : public CSmplBitstreamWriter
    {
    public :
        MuxWriter();
        virtual ~MuxWriter();

        virtual mfxStatus Init(const TCHAR *strFileName,
                               const mfxU16 nWidth,
                               const mfxU16 nHeight,
                               const mfxF64 dFrameRate,
                               const mfxU16 nBitRate,
                               const mfxU32 nCodecId);
        virtual mfxStatus WriteNextFrame(
                               mfxBitstream *pMfxBitstream,
                               bool isPrint = true);
        virtual void Close();

    private:
```

```
        UMC::Muxer                  *m_pMuxer;
        UMC::MuxerParams            *m_pMuxerParams;
        UMC::FileWriter             m_writer;
        UMC::FileWriterParams       m_fwp;
        UMC::VideoStreamInfo        m_videoInfo;
        UMC::MediaData              m_videoData;
        Ipp32s                      m_videoTrackID;
        Ipp64f                      m_frameDuration;
    };
```

From the above code you can see that we utilize the base, mp4 and mpeg2 muxer capabilities from the Intel® IPP sample code. Also note that the `MuxWriter` initialization parameters such as bit rate, dimensions, and frame rate, are all required as meta-data to the muxer.

We implement the full muxer behavior in the `MuxWriter` class part of "sample_utils.cpp". There are three key operations involved in using the Intel® IPP Samples muxer components.

1. Init: Initialize the muxer with stream information such as type, bit rate, dimensions and buffer size
2. PutVideoData: This operation provides the muxer with data for one frame (alternatively LockBuffer and UnlockBuffer could be used to achieve the same result)
3. PutEndOfStream: This operation effectively completes the muxing into the media container

The code below shows how these operations are used in the context of the `MuxWriter` class.

```
MuxWriter::MuxWriter():
        m_pMuxer(NULL),
        m_pMuxerParams(NULL),
        m_videoTrackID(0)
{
}

MuxWriter::~MuxWriter()
{
    Close();
}

void MuxWriter::Close()
{
        if(m_pMuxer){
                m_pMuxer->PutEndOfStream(m_videoTrackID);
        }

        SAFE_DELETE(m_pMuxer);
        SAFE_DELETE(m_pMuxerParams);

        m_bInited = false;
        m_nProcessedFramesNum = 0;
}

mfxStatus MuxWriter::Init( const TCHAR *strFileName,
```

Mark end of muxing. Close file

```
                                const mfxU16 nWidth,
                                const mfxU16 nHeight,
                                const mfxF64 dFrameRate,
                                const mfxU16 nBitRate,
                                const mfxU32 nCodecId)
{
        UMC::Status umcRes = UMC::UMC_OK;

        Close();

        // Select muxer type
        if(MFX_CODEC_AVC == nCodecId)
        {
                m_pMuxer            = new UMC::MP4Muxer();
                m_pMuxerParams      = new UMC::MuxerParams();

                m_videoInfo.stream_type           = UMC::H264_VIDEO;
                m_pMuxerParams->m_SystemType      = UMC::MPEG4_SYSTEM_STREAM;
        }
        else if(MFX_CODEC_MPEG2 == nCodecId)
        {
                m_pMuxer            = new UMC::MPEG2Muxer();
                m_pMuxerParams      = new UMC::MPEG2MuxerParams();

                m_videoInfo.stream_type           = UMC::MPEG2_VIDEO;
                m_pMuxerParams->m_SystemType      = UMC::MPEG2_PURE_VIDEO_STREAM;
        }
        else
                return MFX_ERR_UNSUPPORTED;

        // Initialize file writer
        char tempStr[UMC::MAXIMUM_PATH];
        wcstombs(tempStr, strFileName, UMC::MAXIMUM_PATH);
        strcpy((char *)m_fwp.m_file_name, tempStr);
        m_fwp.m_portion_size = 0;
        umcRes = m_writer.Init(&m_fwp);
        CHECK_NOT_EQUAL(umcRes, UMC::UMC_OK, MFX_ERR_UNKNOWN);

        // Video info
        m_videoInfo.clip_info.height      = nHeight;
        m_videoInfo.clip_info.width       = nWidth;
        m_videoInfo.bitrate               = nBitRate;
        m_videoInfo.framerate             = dFrameRate;
        m_videoInfo.streamPID             = UMC::IdBank::NO_ID;

        // Muxer config
        m_pMuxerParams->m_lpDataWriter    = &m_writer;
        m_pMuxerParams->m_nNumberOfTracks = 1;
        m_pMuxerParams->pTrackParams      =
                new UMC::TrackParams[m_pMuxerParams->m_nNumberOfTracks];
        m_pMuxerParams->pTrackParams[0].type           = UMC::VIDEO_TRACK;
        m_pMuxerParams->pTrackParams[0].info.video     = &m_videoInfo;
        m_pMuxerParams->pTrackParams[0].bufferParams.m_prefOutputBufferSize = 1000000;
        m_pMuxerParams->pTrackParams[0].bufferParams.m_prefInputBufferSize  = 1000000;
        m_pMuxerParams->pTrackParams[0].bufferParams.m_numberOfFrames       = 30;

        m_frameDuration = (Ipp64f)1/dFrameRate;

        // Initialize muxer
        umcRes = m_pMuxer->Init(m_pMuxerParams);
        CHECK_NOT_EQUAL(umcRes, UMC::UMC_OK, MFX_ERR_UNKNOWN);
```

Initialize appropriate muxer object and set overall video and stream type of the container

Configure video stream information such as bit rate and resolution

Since we are only muxing video into the container there is only one track. Configure muxer internal buffer parameters.

Frame duration; required for time stamps

```
        // Initial alloc of memory for muxer data buffer. extended runtime if needed
        m_videoData.Alloc(400000);

        m_bInited = true;

        return MFX_ERR_NONE;
}

mfxStatus MuxWriter::WriteNextFrame(mfxBitstream *pMfxBitstream, bool isPrint)
{
        UMC::Status umcRes = UMC::UMC_OK;

        umcRes = m_videoData.SetDataSize(pMfxBitstream->DataLength);
        if(UMC::UMC_OK != umcRes) {
                // If larger buffer is needed
                umcRes = m_videoData.Alloc(pMfxBitstream->DataLength);
                if(UMC::UMC_OK == umcRes)
                        umcRes = m_videoData.SetDataSize(pMfxBitstream->DataLength);
        }
        CHECK_NOT_EQUAL(umcRes, UMC::UMC_OK, MFX_ERR_UNKNOWN);

        memcpy(m_videoData.GetDataPointer(),
                pMfxBitstream->Data + pMfxBitstream->DataOffset,
                pMfxBitstream->DataLength);

        // Muxer requires time stamps
        // (Adding timestamp offset to oblige Windows Media Player)
        Ipp64f tss = m_nProcessedFramesNum * m_frameDuration + m_frameDuration;

        umcRes = m_videoData.SetTime(tss, tss + m_frameDuration);
        CHECK_NOT_EQUAL(umcRes, UMC::UMC_OK, MFX_ERR_UNKNOWN);

        umcRes = m_pMuxer->PutVideoData(&m_videoData, 0);
        CHECK_NOT_EQUAL(umcRes, UMC::UMC_OK, MFX_ERR_UNKNOWN);

        // mark that we don't need bit stream data any more
        pMfxBitstream->DataLength = 0;

        m_nProcessedFramesNum++;
        if (isPrint && 0 == (m_nProcessedFramesNum - 1) % 100){
                _tcprintf(_T("Frame number: %hd\r"), m_nProcessedFramesNum);
        }

        return MFX_ERR_NONE;
}
```

Buffer to hold encoded video frame

Copy Intel® Media SDK encoded bitstream (frame) into muxer frame buffer

Time stamps required by container

Mux frame buffer into container

Note that the above code provides an example on how to use the Intel® IPP sample muxer components together with Intel® Media SDK. There are several additional muxer options a developer can explore. For further details refer to the Intel® IPP samples source code or document located in the "doc" folder under "audio-video-codecs".

The "audio-video-codecs" (sometimes also named Unified Media Classes, or UMC) also provides a wide range of other codec components such as demuxers(splitters) and audio codecs that could be used in connection with Intel® Media SDK.

Additional changes

To ensure smooth playback (and quick repositioning) for playback on most common media players we also recommend changing the default GOP encoder settings in `InitMfxEncParams` (in "pipeline_encode.cpp").

As an example we choose an interval of 30 frames between each I frame.

```
if(pInParams->CodecId == MFX_CODEC_MPEG2)  // MPEG2
{
       // Required for MPEG to insert sequence header before every I frame
       m_mfxEncParams.mfx.IdrInterval = 1;

       m_mfxEncParams.mfx.GopPicSize = 16;
       m_mfxEncParams.mfx.GopRefDist = 3;
}
else   // H.264
{
       m_mfxEncParams.mfx.GopPicSize = 30;
       m_mfxEncParams.mfx.GopRefDist = 1;
}
```

Conclusion

In this article we presented a method to integrate Intel® Media SDK with Intel® IPP samples to enable developers to enhance their Intel® Media SDK solutions with muxing capabilities. A developer can use this method to quickly implement muxing support into a Intel® Media SDK solution.

This is not the only way of integrating muxing capabilities with Intel® Media SDK. There are many other third-party tools or methods that can be used to perform muxing. However the presented method provides good flexibility and tight integration.

Source Code: http://software.intel.com/file/37473

For developer questions on how to use Intel® Media SDK in general please refer to our Intel® Media SDK forum on the Intel® Software Network site: http://software.intel.com/en-us/forums/intel-media-sdk/

References

- Intel® Media Software Development Kit: www.intel.com/software/mediasdk
- Intel® Integrated Performance Primitives (Intel® IPP): http://software.intel.com/en-us/intel-ipp/
- Intel® IPP samples: http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-samples-license-agreement/
- Microsoft Windows* SDK: http://msdn.microsoft.com/en-us/windows/bb980924

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

(Notice revision #20101101)