FEI Encoding Sample
Design Notes

Table of Contents

Overview

FEI Encoding Sample works with **Intel® Media Server Studio 2017 R2 for Linux Server**.

It demonstrates usage of **Media Server Studio – SDK** (hereinafter referred to as "**SDK**") API for creation of a simple console application that performs encoding of an uncompressed or compressed video streams according to a H.264 video compression standard. The sample uses SDK FEI API (Flexible Encoder Interface) and provides capability to stream internal encoder information during encoding process to specified output.

- ENCODE FEI H.264. This is extension of conventional encoding functionality described in SDK API Reference Manual. It covers all stages of encoding and produces encoded bitstream from original raw frames. It is performed by ENCODE class of functions.
- PREENC FEI H.264. PreENC – pre encoding. As follow from the name it is preliminary step to gather MB level statistics, that later may be used for optimal encode configuration. This step may be used on its own for different kind of video processing, but usually it is followed by ENCODE step.
- ENC FEI H.264. This interface perform following encoding stages: Intra Prediction, Motion Estimation and Mode Decision. This step may be used on its own, but usually it is followed by PAK step.
- PAK FEI H.264. This interface perform following encoding stages: Transform; Quantization; Entropy Coding; generating Reconstruct Frames, which can be used by ENC; generating of output bitstream.

Features

FEI Encoding Sample supports the following video formats:

input (uncompressed/compressed)	YUV420, NV12, H.264 (AVC), MPEG2, VC1
output (compressed)	H.264 (AVC)

Note: For format YUV420, the **FEI Encoding Sample** assumes the order Y, U, V in the input file.

Software Requirements

See `<install-folder>/Media Samples Guide.pdf`.

How to Build the Application

See `<install-folder>/Media Samples Guide.pdf`.

Design Notes

Sample FEI allows to build pipeline not only with FEI interfaces but with VPP and Decoder as well. Class `CEncodingPipeline` manages all pipelines.

From general perspective pipeline lifecycle looks as follows:

```
ParseInputString();
Pipeline -> Init();
Pipeline -> Run();
Pipeline -> Close();
```

Initialization of pipeline

Initialization stage of the pipeline is performed by `CEncodingPipeline::Init(sInputParams *pParams)` and includes following:

- `CEncodingPipeline::InitSessions`
  Initializations of sessions with `MFXVideoSession::Init`.

Note: PREENC and ENC(+PAK) always executes in different sessions because they use same class of API functions (ENC). Normally all of the pipeline components are in `m_mfxSession`. But if PREENC and ENC(+PAK) or PREENC+DownSampling (via VPP) + VPP present in pipeline, PREENC (+VPP for DownSampling) executes in `m_preenc_mfxSession` (separate PreENC session).

Note: `m_mfxSession` also used to obtain unique id - `m_BaseAllocID` for allocator, another id - `m_EncPakReconAllocID` (which is actually `m_BaseAllocID + 1`) used for allocation of separate reconstruct surface pool for PAK.

- `CEncodingPipeline::CreateAllocator()`
  Allocator creation. D3D9 memory supported (FEI ENCODE supports system memory). Creation of HW device.

- Creation of all requested interfaces. Components passes back a `m_commonFrameInfo`, which is `mfxFrameInfo` for ENCODE / ENCPAK interfaces, i.e. it contains settings of parameters, which will be used for actual encoding. It considers resizing and info from decoder if they are present. Also each of the interfaces initializes fields in `AppConfig::PipelineCfg` and uses some information from this section to get parameters from another interfaces.

- `CEncodingPipeline::ResetMFXComponents`
  Initialization of all interfaces present in pipeline: Decoder (here extension buffer for Decode StreamOut should be passed), VPP, (VPP for PREENC input frame downsampling), PREENC, ENCODE or ENC and PAK; and frame pools allocation.

Note: Currently sample_fei works only with `AsyncDepth = 1`

- `CEncodingPipeline::AllocFrames()` (inside `ResetMFXComponents`)
  Frames allocation for all interfaces present in pipeline.

Note: ENCODE, ENC only, PREENC should have `MFX_MEMTYPE_FROM_ENC` submemtype. PAK, ENC+PAK should have `MFX_MEMTYPE_FROM_ENC | MFX_MEMTYPE_FROM_PAK` submemtype.

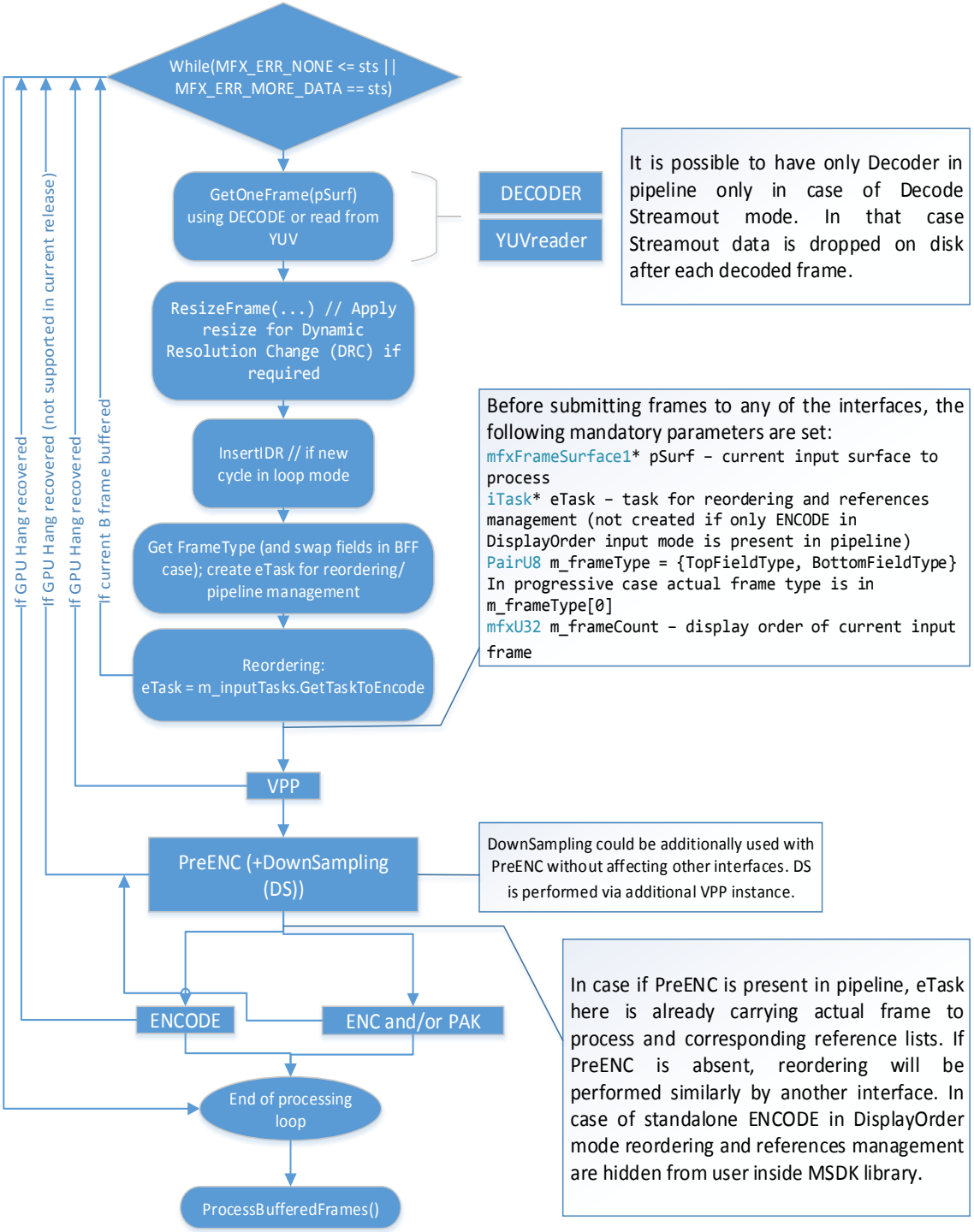
Note: Entire pipeline shares same surface pool (but VPP requires additional pool for resized surfaces), if PAK is present in pipeline, it uses another pool for reconstruct surfaces. For that purpose, all interfaces should have identical AllocId (`BaseAllocID`, but PAK uses `m_EncPakReconAllocID`).

- `CEncodingPipeline::SetSequenceParameters`
  Copy back adjusted parameters to pipeline if some of the interfaces did such adjustment. Fill encoding related parameters, i.e. number of macroblocks for ENCODE and PreENC, number of fields. Initialization of `iTaskPool m_inputTasks` and `iTaskParams m_taskInitializationParams` for task management (see Reordering and reference lists management sections for details).

- `CEncodingPipeline::AllocExtBuffers`
  Allocation of FEI Extension Buffers for interfaces.
  Sets of buffers are stored in `bufList m_preencBufs, m_encodeBufs` for PreENC and ENCODE respectively. For DECODE Streamout buffers are stored in `std::vector<mfxExtFeiDecStreamOut*> m_StreamoutBufs`.

Pipeline execution

Function `CEncodingPipeline::Run()` is responsible for direct processing of input video sequence. It consists of one main loop where new income frame obtained and then passed towards entire pipeline of FEI and other MSDK interfaces. In function `ProcessBufferedFrames()` bufferes B frames from last mini-GOP are processed.



DECODE

Implemented by `MFX_DecodeInterface` wrapper on MSDK decoder.

Provides decoder MFX interface functions proxy (Init, Close, Reset, QueryIOSurf, etc.).

`mfxStatus MFX_DecodeInterface::FillParameters()`
Fills internal `mfxVideoParam` and points `PipelineCfg.pDecodeVideoParam` to it.

`mfxStatus MFX_DecodeInterface::GetOneFrame(mfxFrameSurface1* & pSurf)`
Decodes one frame from input bitstream and points `pSurf` to it.

`mfxStatus MFX_DecodeInterface::ResetState()`
Resets decoder to beginning of input stream. Used in loop mode processing.

Note: decode uses frames from `ExtSurfPool* m_pSurfPool`, which are owned by `CEncodingPipeline`. Also these frames have attached `mfxExtFeiDecStreamOut` in case of decode streamout.

Note: in case of DECODE + VPP in one session and not mixed-picstructs content, these two interfaces will use async mode, i.e. actual sync will be performed after VPP stage.

VPP

Implemented by `MFX_VppInterface` wrapper on MSDK VPP.

Provides vpp MFX interface functions proxy (Init, Close, Reset, QueryIOSurf, etc.).

`mfxStatus MFX_VppInterface::FillParameters()`
Fills internal `mfxVideoParam` and points `m_pAppConfig->PipelineCfg.pVppVideoParam` to it.

`mfxStatus MFX_VppInterface::VPPoneFrame(mfxFrameSurface1* pSurf_in, mfxFrameSurface1* pSurf_out)`
Perform VPP using `pSurf_in` as input and `pSurf_out` as output.

Note: in case of DECODE + VPP in one session and not mixed-picstructs content, these two interfaces will use async mode, i.e. actual sync will be performed after VPP stage.

PREENC

Sample_fei uses `FEI_PreencInterface` class as wrapper on FEI PreENC interface.

Class variables:

`MFXVideoSession* m_pmfxSession` - pointer to associated MFX session.
`MFXVideoENC* m_pmfxPREENC` - PreENC FEI interface.

`MFXVideoVPP*` `m_pmfxDS` - VPP for DownSampling (if required).
`iTaskPool*` `m_inputTasks` - TaskPool managed by pipeline, used to get reference tasks during multicall.
`mfxU32` `m_allocId` - allocId associated with current interface.
`mfxVideoParam` `m_videoParams` - video parameters for PreENC.
`mfxVideoParam` `m_DSParams` - video parameters for DS VPP.
`mfxVideoParam` `m_FullResParams` - video parameters for full-res frame (i.e. same video parameters will use ENCODE)
`bufList*` `m_pExtBuffers` - PreENC extension buffers
`bufList*` `m_pEncExtBuffers` - ENCODE extension buffers (used for predictors repacking).
`AppConfig*` `m_pAppConfig` - pipeline config
`mfxSyncPoint` `m_SyncPoint` - MSDK sync point
`bool` `m_bSingleFieldMode` - single/double field mode flag
`mfxU8` `m_DSstrength` - strength of DownSampling (0 if none)

`bool` `m_bMVout` - true if Motion Vectors output present
`bool` `m_bMBStatout` - true if MB statistics output enabled

`mfxExtFeiPreEncMV::mfxExtFeiPreEncMVMB m_tmpMVMB` - temporary memory for predictors repacking

`FILE* m_pMvPred_in` - for MV predictors input
`FILE* m_pMbQP_in` - for per-MB QP input
`FILE* m_pMBstat_out` - for MB statistics output
`FILE* m_pMV_out` - for Motion Vectors output

`std::vector<mfxExtBuffer*> m_InitExtParams` - extension buffers for PreENC initialization
`std::vector<mfxExtBuffer*> m_DSExtParams` - extension buffers for VPP (for DownSampling) initialization

`std::vector<mfxI16> m_tmpForMedian` - temporary memory for median computation

Class functions:

`mfxStatus FEI_PreencInterface::FillParameters()`
Fills internal `mfxVideoParam` and points `m_pAppConfig->PipelineCfg.pPreencVideoParam` to it. Also allocates all required buffers for PreENC initialization and store it in `m_InitExtParams`. Initializes all file pointers.

`mfxStatus FEI_PreencInterface::FillDSVideoParams()`
Fills internal `mfxVideoParam` for DS VPP and points `m_pAppConfig->PipelineCfg.pDownSampleVideoParam` to it.

`void FEI_PreencInterface::GetRefInfo(…)`
Copies back encoding parameters after MSDK adjustments.

`mfxStatus CEncodingPipeline::PreencOneFrame(iTask* eTask)`
Takes reordered `iTask* eTask` as input and performs PreENC multicall on related frame (if `m_DSstrength != 0` VPP invoked for downsampling and then resized frame passed for multi call processing). After that output MVs repacked to predictors (if

ENCODE interface present in pipeline) and output is flushed on disk (if requested). If GPU hang detected, `MFX_ERR_GPU_HANG` status returned.

`mfxStatus FEI_PreencInterface::DownSampleInput(iTask* eTask)`
Downsamples input frame stored in `eTask` with VPP (full resolution frame is stored in `eTask->fullResSurface`, downsampled in `eTask->in.InSurface`).

`mfxStatus FEI_PreencInterface::ProcessMultiPreenc(iTask* eTask)`
Performs PreENC multi call on available references of current `iTask* eTask`. If GPU hang is detected, `MFX_ERR_GPU_HANG` status returned. After each processing of the frame, output buffers are stored in `eTask->preenc_output`, for further repacking. See diagram for detailed function workflow.

Note: GPU Hang Recovery is not supported by PREENC in current release.

`mfxStatus FEI_PreencInterface::GetRefTaskEx(iTask *eTask, mfxU8 l0_idx, mfxU8 l1_idx, mfxU8 refIdx[2][2], mfxU8 ref_fid[2][2], iTask *outRefTask[2][2])`
This function takes current `iTask *eTask` as input and returns reference tasks `iTask *outRefTask[2][2]` ([id of field: 0 – first, 1 - second][0 - L0, 1 – L1 reference]) and its reference indexes `iTask *outRefTask[2][2]` and parity `mfxU8 ref_fid[2][2]` (this information is required for motion vectors to predictors repacking), that corresponds to `mfxU8 l0_idx` and `mfxU8 l1_idx` reference indexes (if some of the references are unavailable, corresponding values are zeroed).

`mfxStatus FEI_PreencInterface::InitFrameParams(iTask* eTask, iTask* refTask[2][2], mfxU8 ref_fid[2][2], bool isDownsamplingNeeded)`
`iTask* eTask` – current task to process.
`iTask* refTask[2][2]` – reference tasks.
`mfxU8 ref_fid[2][2]` – id of current field: 0 – first field (or frame), 1 – second field.
`bool isDownsamplingNeeded` – update or not internal driver cache (true for first call for current input frame, false for subsequent calls for same input frame).

Fills all mandatory structures for `ProcessFrameAsync` call. Also attaches free set of extension buffers (`bufSet`).

Note: for interlaces content two instances of each buffers required. First buffer corresponds to the first field, second one to second.

`mfxStatus FEI_PreencInterface::FlushOutput(iTask* eTask)`
Flushes output buffers on disk if requested and –perf option wasn't provided.

`mfxStatus FEI_PreencInterface::RepackPredictors(iTask* eTask)`
Repacks PreENC Motion Vectors output to ENCODE predictors input. It chooses up to 4 best in terms of distortion L0 and L1 (independently) predictors for each of the macroblocks (independently) with calculating median of 16 MVs to form one predictor vector (median is calculated independently for x and y coordinate).

`mfxStatus FEI_PreencInterface::RepackPredictorsPerf(iTask* eTask)`
Performs fast simplified repacking if –perf option specified. This algorithm uses up to 4 first predictors (without sorting by distortion) and first of 16 MVs (without median calculation, just taking MV with index 0).

Note: with current limitation of 4 (= max{n_Pref, n_Brefs}) maximal references and 4 predictors for ENC/ENCODE, there is no need to choose 4 best in sample_fei. Also our experiments shows that avoiding computing median and picking always first MV as predictor with `PassPreEncMVPred2EncExPerf` gives minor quality reduction against fair calculating of best predictor from 16 available MVs, with significant computing savings. Recommend to use `-perf` option for PREENC + ENC (ENCODE) pipelines.

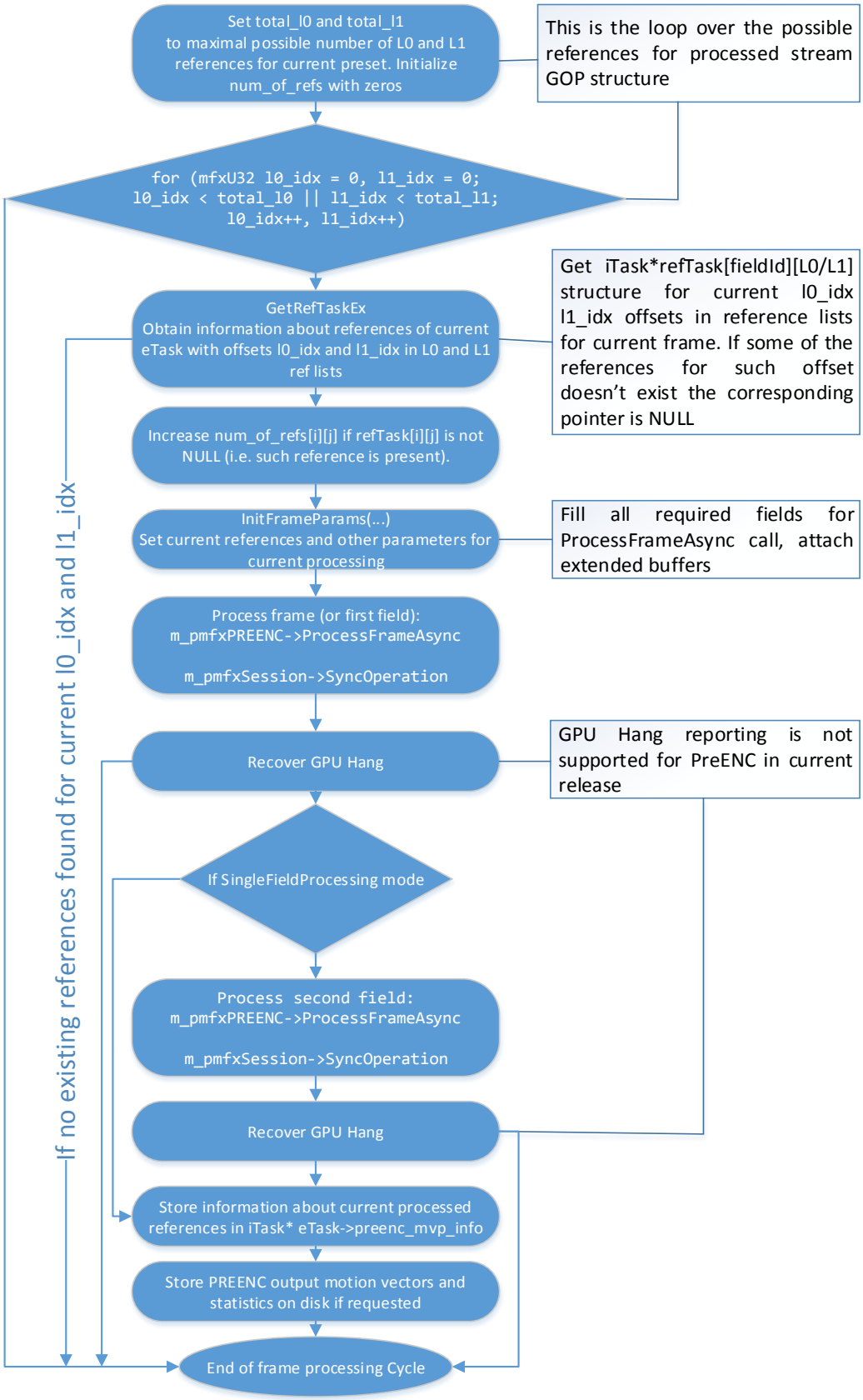
```
void FEI_PreencInterface::UpsampleMVP(mfxExtFeiPreEncMV::mfxExtFeiPreEncMVMB *
preenc_MVMB, mfxU32 MBindex_DS, mfxExtFeiEncMVPredictors* mvp, mfxU32 predIdx,
mfxU8 refIdx, mfxU32 L0L1)
```

Upscales output PreENC motion vectors to corresponding full-resolution frame in case of processing on downsampled frames.

preenc_MVMB - current MB on downsampled frame
MBindex_DS - index of current MB
mvp - motion vectors predictors array for full-resolution frame
predIdx - index of current predictor [0-3] (up to 4 is supported)
refIdx - index of current reference (for which predictor is applicable) in reference list
L0L1 - indicates list of references being processed [0-1] (0 - L0 list, 1- L1 list)

```
mfxStatus FEI_PreencInterface::ResetState()
```

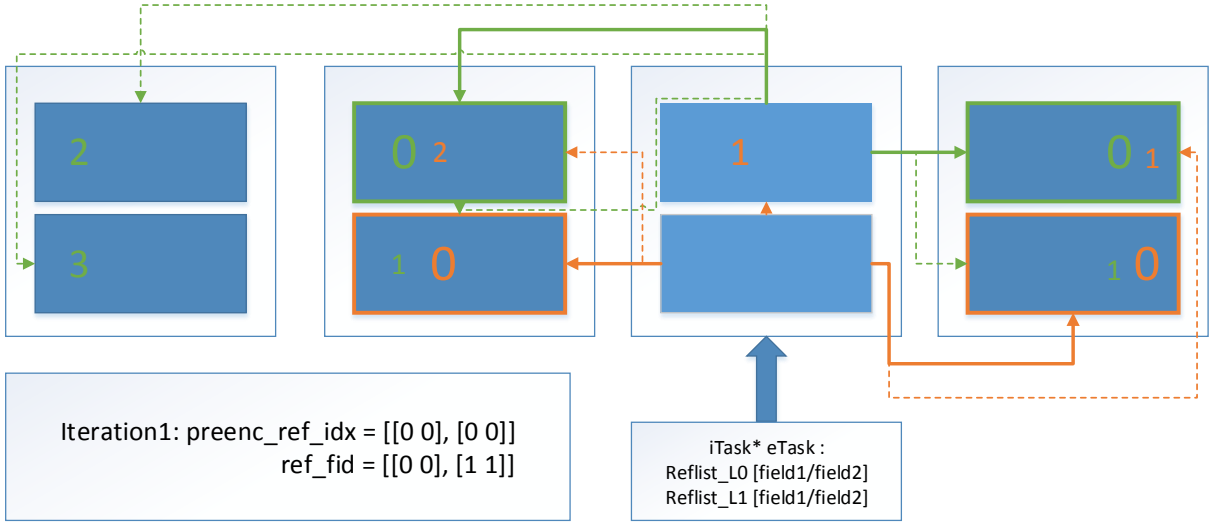
Reset state of PreENC to initial, i.e. before processing of first frame. Used in loop mode processing.

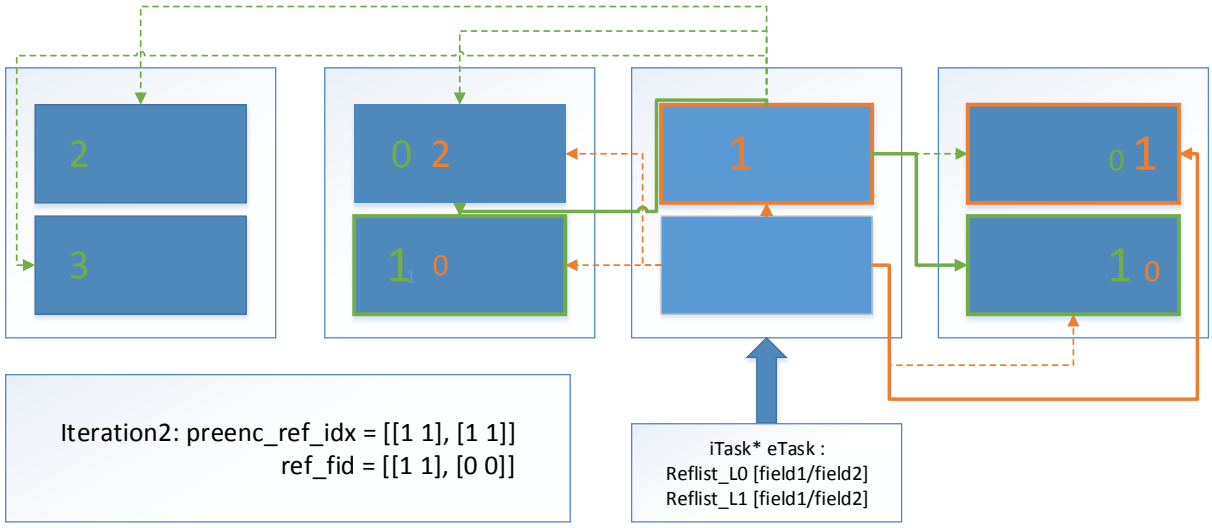


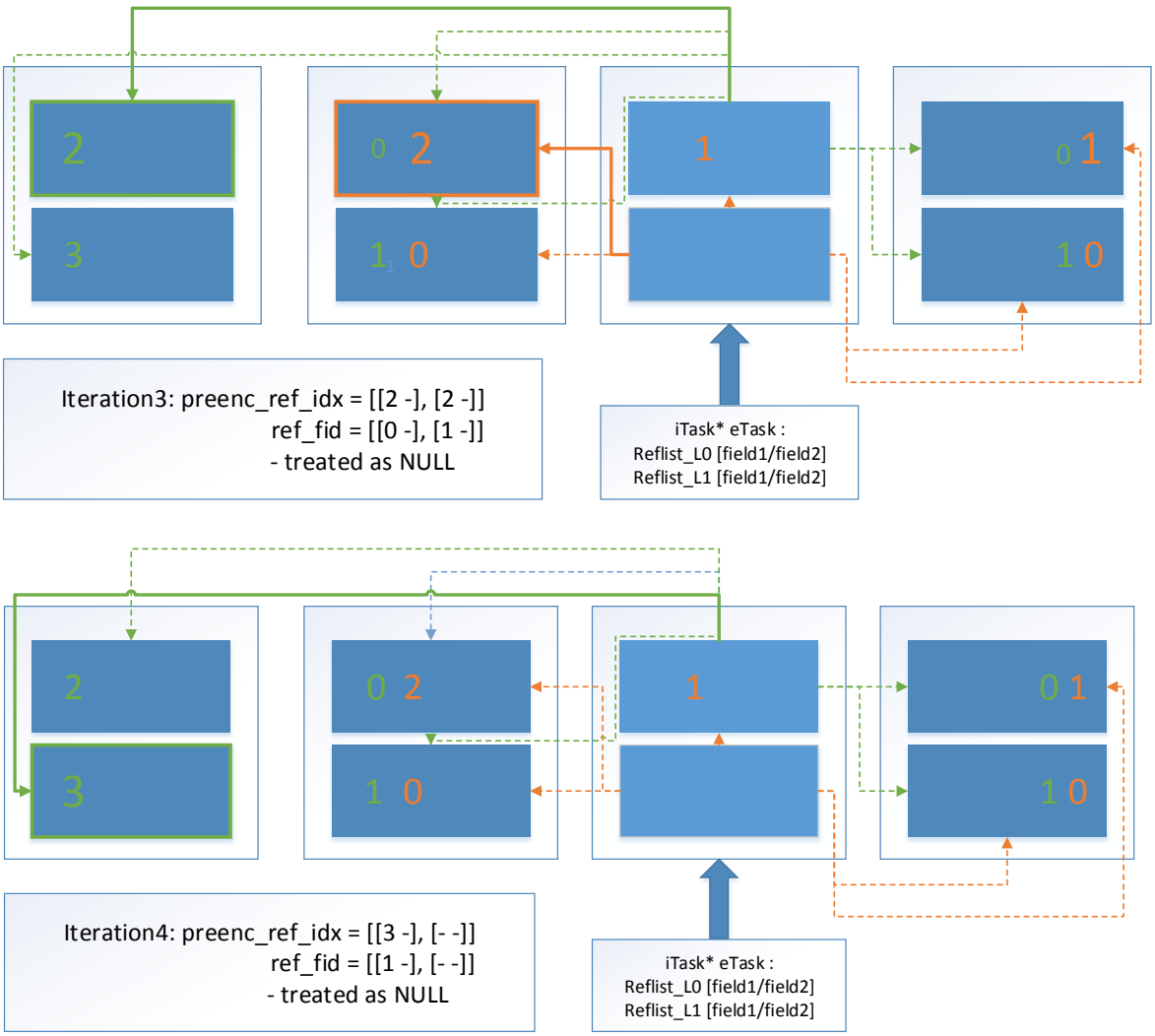
The diagram below shows workflow of references estimation on interlaced stream with two backward reference frames and one forward (each frame-slot contains of two possible references).

For such preset the following steps will be taken:

Note: dashed lines indicates inactive references in current PREENC call, solid – active. Green color is for to field references, orange – for bottom field references. Current frame is marked by arrow.







At each iteration one L0 and/or one L1 reference processed.

Note: all L1 references processed at iterations 1 and 2, iterations 3 and 4 processing only L0 references.

ENCODE

Sample_fei uses `FEI_EncodeInterface` class as wrapper on FEI ENCODE interface.

Class variables:

`MFXVideoSession*`	`m_pmfxSession`	- pointer to ENCODE's MFX session.
`MFXVideoENCODE*`	`m_pmfxENCODE`	- FEI ENCODE.
`mfxEncodeCtrl`	`m_encodeControl`	- ENCODE control structure.
`mfxVideoParam`	`m_videoParams`	- ENCODE's video parameters.

```
mfxU32       m_allocId          - allocId used to allocate ENCODE's input surfaces.
bufList*     m_pExtBuffers      - list of ENCODE's extension buffers sets.
AppConfig*   m_pAppConfig       - pointer to CencodingPipeline's config.
mfxBitstream m_mfxBS            - output bitstream.
mfxSyncPoint m_SyncPoint        - ENCODE's sync point.
bool         m_bSingleFieldMode - true if ENCODE is in sigle-field processing mode.

CSmplBitstreamWriter m_FileWriter - bitstream writer.

FILE* m_pMvPred_in     - MV predictors input file.
FILE* m_pENC_MBCtrl_in - per-MB control input file.
FILE* m_pMbQP_in       - per-MB QP input file.
FILE* m_pRepackCtrl_in - repacking control input file.
FILE* m_pMBstat_out    - macroblock statistics output file.
FILE* m_pMV_out        - MV output file.
FILE* m_pMBcode_out    - MB code output file.

std::vector<mfxExtBuffer*> m_InitExtParams - vector of ENCODE's init extension
buffers.
SyncList sync_list - list of surface-extension buffers set. It used in display order
mode for extension buffers management.

std::vector<mfxI16> m_tmpForMedian - temporary array for median calculation.
std::vector<mfxExtFeiPreEncMV::mfxExtFeiPreEncMVMB> m_tmpForReading - temporary
array for PreENC MVs input, which will be repacked to predictors using fast repacking
algorithm.

mfxExtFeiEncMV::mfxExtFeiEncMVMB m_tmpMBencMV - temporary memory initialized with
0x8000 to output as default value for I frames MVs.
```

Class functions:

```
mfxStatus FEI_EncodeInterface::FillParameters()
```
Fills internal `mfxVideoParam` and points `m_pAppConfig->PipelineCfg.pEncodeVideoParam` to it. Also allocates all required buffers for ENCODE initialization and store it in `m_InitExtParams`. Initializes all file pointers.

Note: If some custom deblocking parameters specified, they are passed with `mfxExtFeiSliceHeader` (one for each field) to Init() function.

```
void FEI_EncodeInterface::GetRefInfo(…)
```
Copies back encoding parameters after MSDK adjustments.

```
mfxStatus FEI_EncodeInterface::InitFrameParams(mfxFrameSurface1* encodeSurface,
PairU8 frameType, iTask* eTask)
```
Fills all mandatory structures for `EncodeFrameAsync` call. Also attaches free set of extension buffers (`bufSet`).

`iTask* eTask` – current task to process. If ENCODE is in display order mode or if we drain buffered frames, this pointer is `NULL`.
`mfxFrameSurface1* encodeSurface` – surface to encode, ignored if `eTask != NULL`
`PairU8 frameType` – type of current frame, ignored if `eTask != NULL`

Note: if ENCODE works in EncodedOrder mode, frame type for current input surface should be set in `m_encodeControl->FrameType`.

Note: output buffers in case of mixed picstructs streams can be used only without internal ENCODE reordering: i.e. in presets without B frames or with ENCODE in EncodedOrder mode, otherwise output buffers layout would be miss aligned with submitted frames.

Note: `SyncList` which owns `std::list<std::pair<bufSet*, mfxFrameSurface1*> > sync_list` used for synchronization: when input surface locker became 0 (frame processing complete), corresponding set of extension buffers will be released. In case of ENCODE in display order such correspondence may be impaired (when ENCODE uses internal reordering), but the output on disk would be correct and no buffers leak would happen.

`mfxStatus FEI_EncodeInterface::ResetState()`
Reset state of ENCODE to initial, i.e. before encoding of first frame. Used in loop mode processing.

`mfxStatus FEI_EncodeInterface::FlushOutput()`
Flushes output buffers on disk if requested and –perf option wasn't provided.

`mfxStatus FEI_EncodeInterface::EncodeOneFrame(iTask* eTask, mfxFrameSurface1* pSurf, PairU8 runtime_frameType)`
Encodes current frame. If current frame encoded successfully, writes new chunk of data to bitstream and flushes output buffers on disk.

`iTask* eTask` – current task to process. If ENCODE is in display order mode or if we drain buffered frames, this pointer is `NULL`.
`mfxFrameSurface1* pSurf` – surface to encode, ignored if `eTask != NULL`
`PairU8 runtime_frameType` – type of current frame, ignored if `eTask != NULL`

Note: `SyncList::Update()` is called after encoding. In display order mode it releases `bufSet` associated with encoded frame, otherwise `iTaskPool` in `CEncodingPipeline` is responsible for buffers management.

`mfxStatus FEI_EncodeInterface::AllocateSufficientBuffer()`
Allocates memory to extend bitstream, if required.

ENCPAK

Sample_fei uses `FEI_EncPakInterface` class as wrapper on FEI ENC / FEI PAK interface. Implementation supports only ENC and only PAK cases, if some of the interfaces is not requested, corresponding pointer (`m_pmfxENC or m_pmfxPAK`) will be initialized as NULL.

Class variables:

`MFXVideoSession* m_pmfxSession` – pointer to ENC / PAK MFX session

`MFXVideoENC*` `m_pmfxENC` – FEI ENC
`MFXVideoPAK*` `m_pmfxPAK` – FEI PAK
`iTaskPool*` `m_inputTasks` – list of tasks (for proper reference surfaces detection)
`mfxVideoParam` `m_videoParams_ENC` – ENC's video parameters.
`mfxVideoParam` `m_videoParams_PAK` – PAK's video parameters.
`mfxU32` `m_allocId` – allocId used to allocate reconstruct surfaces (`CencodingPipeline::m_EncPakReconAllocID`).
`bufList*` `m_pExtBuffers` – ENC / PAK extension buffers.
`AppConfig*` `m_pAppConfig` – pointer to `CencodingPipeline`'s config.
`mfxBitstream` `m_mfxBS` – output bitstream.
`mfxSyncPoint` `m_SyncPoint` – ENC / PAK sync point.
`bool` `m_bSingleFieldMode` – true if ENC / PAK is in sigle-field processing mode.
`RefInfo` `m_RefInfo` – structure that holds current DPB / Reflists configuration (required for proper filling of PPS/SliceHeader extension buffers).

`CSmplBitstreamWriter m_FileWriter` – bitstream writer

`FILE* m_pMvPred_in` - MV predictors input file.
`FILE* m_pENC_MBCtrl_in` - per-MB control input file.
`FILE* m_pMbQP_in` - per-MB QP input file.
`FILE* m_pRepackCtrl_in` - repacking control input file.
`FILE* m_pMBstat_out` - macroblock statistics output file.
`FILE* m_pMV_out` - MV output file.
`FILE* m_pMBcode_out` - MB code output file.

`std::vector<mfxExtBuffer*> m_InitExtParams_ENC, m_InitExtParams_PAK` - vectors of ENC and PAK extension buffers used on Init stage.

`std::vector<mfxI16> m_tmpForMedian` - temporary array for median calculation.
`std::vector<mfxExtFeiPreEncMV::mfxExtFeiPreEncMVMB> m_tmpForReading` - temporary array for PreENC MVs input, which will be repacked to predictors using fast repacking algorithm.

`mfxExtFeiEncMV::mfxExtFeiEncMVMB m_tmpMBencMV` - temporary memory initialized with 0x8000 to output as default value for I frames MVs.

Class functions:

`mfxStatus FEI_EncodeInterface::FillParameters()`
Fills internal `mfxVideoParam` for ENC and PAK, points `m_pAppConfig->PipelineCfg.pEncVideoParam and m_pAppConfig->PipelineCfg.pPakVideoParam` to them. Also allocates all required buffers for ENC and PAK initialization and store it in `m_InitExtParams_ENC and m_InitExtParams_PAK`. Initializes all file pointers.

Note: if `mfxExtFeiSPS` and `mfxExtFeiPPS` FEI extension buffers passed to Init, they override default MSDK settings for SPS and PPS buffers. `mfxExtFeiSPS` is unsupported during runtime. `mfxExtFeiPPS` is supported during runtime, but parameters from Init should be preserved. Init settings of `mfxExtFeiSPS` and `mfxExtFeiPPS` should be identical for ENC and PAK. Only one instance of each buffer should be passed during Init.

`void FEI_EncodeInterface::GetRefInfo(…)`
Copies back encoding parameters after MSDK adjustments.

`mfxStatus FEI_EncodeInterface::InitFrameParams(iTask* eTask)`
Fills all mandatory structures for `ProcessFrameAsync` call. Also attaches free set of extension buffers (`bufSet`) to `eTask`.

Note: `mfxExtFeiPPS` and `mfxExtFeiSliceHeader` extension buffers are mandatory for encoding. `mfxExtFeiPPS` holds current DPB state (state for the moment before current frame encoding), and `mfxExtFeiSliceHeader` holds current L0 and L1 references lists (different reference lists per-slice within frame is unsupported). One buffer per-field should be passed.

Note: Any other information except DPB state in `mfxExtFeiPPS` should be preserved from Init stage.

Note: DPB state in `mfxU16 mfxExtFeiPPS::ReferenceFrames[16]` are simply indexes in `mfxFrameSurface1* mfxPAKInput::L0Surface` array, `0xffff` indicates end of DPB.

Note: Reference lists are stored in `mfxExtFeiSliceHeader::Slice::RefL0 (RefL1)` in terms of indexes in `mfxFrameSurface1* mfxPAKInput::L0Surface` array.

`mfxStatus FEI_EncodeInterface::ResetState()`
Reset state of ENC and/or PAK to initial, i.e. before encoding of first frame. Used in loop mode processing.

`mfxStatus FEI_EncodeInterface::FlushOutput()`
Flushes output buffers on disk if requested and –perf option wasn't provided.

`mfxStatus FEI_EncPakInterface::EncPakOneFrame(iTask* eTask)`
Encodes current frame from `eTask`. If encoding succeed, writes new chunk of data to bitstream (if PAK is present) and flushes output buffers on disk.

`mfxStatus FEI_EncodeInterface::AllocateSufficientBuffer()`
Allocates memory to extend bitstream, if required.

`mfxStatus FEI_EncPakInterface::FillRefInfo(iTask* eTask)`
Extracts DPB state and references lists configuration from `eTask` and stores it in `RefInfo m_RefInfo` for further filling `mfxExtFeiSliceHeader` and `mfxExtFeiPPS`.

Note: ENC + PAK in double-field mode may produce heavy second field in case of inter-field references within frame, because reconstruct for first field is not generated yet. To avoid such issue single-field mode can be used. In general it is strictly recommended to use single-field mode for interlaced encoding.

Application-level reordering and iTask management

Sample_fei uses `iTaskPool m_inputTasks` for general pipeline encoding management: reordering and references lists construction. It consists of both: already processed tasks, which could be used as references (DPB), and reordered tasks awaiting encoding. `iTask`s are created if application-level reordering required, i.e. the only exception is ENCODE in DisplayOrder mode. They store all information required for encoding, reordering and reference lists management.

The lifecycle of `iTask* eTask` is the following:

- If manual reordering required, new `iTask*` is created and stored into `m_inputTasks`.
- To get reordered task to encode `iTask* GetTaskToEncode(bool buffered_frames_processing)` is called. If such task is found, its DPB and reflists are configured and task is returned to main encoding loop. If task is not found (B frames are buffering at the moment) `NULL` pointer is returned and new income frame requested. `buffered_frames_processing` – indicates processing of previously buffered task: last B frame could be converted to P.

Note: at this moment `iTask* eTask` holds some of the previously submitted frames and may not correspond to current income frame for presets with B frames because of reordering.

- Then obtained task is passed towards the pipeline to all of the created interfaces.
- After successful processing by all of the interfaces, `iTaskPool::UpdatePool()` is invoked to remove already processed tasks that is not in DPB of last processed task.

Note: VPP stage is performed before new task creation.

Note: last frame could be a B frame, so it won't have forward reference. The default behavior is to convert such frame to P frame. If `MFX_GOP_STRICT` flag set, conversion won't be applied and such frame will be encoded as B frame with only L0 reference.

Extension buffers management

Sample_fei uses extension buffers management for flexible control of input and output for FEI interfaces.

Buffers for current frames are held in `iTask`'s fields: `bufSet* bufs` for ENCPAK and `bufSet* preenc_bufs` for PREENC.

All sets are allocated in `CEncodingPipeline::AllocExtBuffers()` according to user's request and stored in `CencodingPipeline:: bufList m_preencBufs` (for PREENC), `m_encodeBufs` (for ENCODE/ENCPAK). Pointers to these buffers passed to `FEI_PreencInterface` and `FEI_EncodeInterface` / `FEI_EncPakInterface` constructors.

`bufSet` uses boolean field `vacant` for indication whether current buffer set is in use.

Buffers obtained by components with `bufSet* bufList::GetFreeSet()` inside `InitFrameParams`. Such set will be marked as occupied and added to current `iTask`.

Inside `iTask` destructor (which will be invoked during `iTaskPool::UpdatePool()`) all associated `bufSets` will be marked as vacant and will be reused in next encodings.

Note: for FEI ENCODE in display order mode buffers set will be released when corresponding input surface is free (i.e. `mfxFrameSurface1::Data.Locked == 0`).

Frames allocation

Allocation of frames performed in `CEncodingPipeline::AllocFrames()`

The logic of surface pools allocation takes into account following rules:

- PREENC input surfaces requires memory type `MFX_MEMTYPE_FROM_ENC`
- VPP requires two pools: input and output
- DECODE surfaces requires memory type `MFX_MEMTYPE_FROM_DECODE`
- Input surfaces for FEI interfaces requires memory type `MFX_MEMTYPE_EXTERNAL_FRAME | MFX_MEMTYPE_VIDEO_MEMORY_PROCESSOR_TARGET`
- Without VPP (and DownSampling) whole pipeline shares same surfaces pool
- PAK requires separate pool for reconstruct surfaces
- Reconstruct/Input surfaces for ENC and/or PAK requires memory type `MFX_MEMTYPE_FROM_ENC | MFX_MEMTYPE_FROM_PAK`

Examples:

- Pipeline YUVreader/DECODE + PREENC + ENCODE will have one shared pool
- Pipeline YUVreader/DECODE + VPP + PREENC + ENCODE will have two surfaces pools: one for part before VPP and second one for part after VPP
- Pipeline YUVreader/DECODE + VPP1 + VPP2(DS)+PREENC + ENCODE will have three surfaces pools: one for the part before VPP1; second one for VPP1 output, VPP2 and ENCODE input; the third one is for VPP2 output and PreENC
- Pipeline YUVreader/DECODE + VPP + DS+PREENC + ENCPAK will have three surfaces pools: one for the part before VPP1; second one for VPP1 output, VPP2 and ENCODE input; the third one is for VPP2 output and PreENC; forth pool to store PAK reconstruct surfaces

Reordering and references lists management

Reordering

Reordering of frames performed inside `iTaskPool::GetReorderedTask(bool buffered_frames_processing)` in accordance with H.264 coding standard.

`bool buffered_frames` – indicates processing of last GOP, last B frame without L1 reference will be converted to P, if GOP optimization option MFX_GOP_STRICT not provided.

Returned `iTask*` holds current task to process. During buffering of B frames from first mini-GOP `NULL` returned and new income frame requested.

Reference lists management

`iTask` holds information about DPB state before and after encoding of current frame, this information used to construct L0 and L1 reference lists according to H.264 standard, with respect to limitations of reference frames counts and DPB-size, provided by user.

The reflists construction itself is performed by four functions: `UpdateDpbFrames, InitRefPicList, ModifyRefPicLists, MarkDecodedRefPictures.`

Whole configuration of `iTask` for encoding is performed in `iTask* iTaskPool::GetTaskToEncode(bool buffered_frames_processing)` with following steps:

- `iTask* iTaskPool::GetReorderedTask(bool buffered_frames_processing)` Where performed reordering and obtained `iTask` to encode.
- Filled required for reordering information:

```
ArrayDpbFrame   m_dpb[2]; // DPB state before encoding first and second
fields
ArrayDpbFrame   m_dpbPostEncoding; // DPB after encoding a frame (or 2
fields)
ArrayU8x33      m_list0[2]; // L0 list for first and second field
ArrayU8x33      m_list1[2]; // L1 list for first and second field
PairU8          m_type;     // type of first and second field
mfxU8           m_fid[2];   // progressive fid=[0,0]; tff fid=[0,1]; bff
fid=[1,0]
mfxU8           m_fieldPicFlag; // is interlaced frame
PairI32         m_poc;          // POC of first and second field

mfxU32  m_frameOrderIdr;    // most recent IDR frame in display order
mfxU32  m_frameOrderI;      // most recent I frame in display order
mfxU32  m_frameOrder;       // current frame order in display order

mfxU16 NumRefActiveP;   // limits of active
mfxU16 NumRefActiveBL0; // references for
mfxU16 NumRefActiveBL1; // reflists management
```

- `void UpdateDpbFrames(iTask& task, mfxU32 field, mfxU32 frameNumMax)`
  Update m_picNum for each frame in DPB.

- `void InitRefPicList(iTask& task, mfxU32 field)`
  Initialize reference lists form current DPB state according to H.264 standard.

- `void ModifyRefPicLists(mfxVideoParam& video, iTask& task, mfxU32 fieldId)`
  Adjust lists to current limitations: remove references to previous POC (i.e. each I frame is entry point; that's not true for interlaced content due to issue in MSDK relists management logic, fix is upcoming); adjust size of each list to user limitation of active references.

  > Note: for interlaced case field of same parity should be the first entry in reference list. Swap of first two entries performed if required. This is hardware limitation for HSW.

- `void MarkDecodedRefPictures(mfxVideoParam& video, iTask& task, mfxU32 fid)`
  Update DPB if processing frame is reference frame.

- Repeat call of UpdateDpbFrames, InitRefPicList, ModifyRefPicLists for second field in case of interlaced content encoding.

  > Note: no need for another call of MarkDecodedRefPictures as second field processing doesn't change DPB.

Closing pipeline

Inside of `CEncodingPipeline::Close()`:

- Delete all created interfaces.
- Close MFX sessions.
- Delete allocated frames.
- Close and delete allocator.
- Release all allocated resources: Clear `iTaskPool`, delete all allocated extension buffers (inside `CEncodingPipeline::ReleaseResources()`).
- Delete HW device.

Legal Information

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 by visiting Intel's Web Site.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Optimization Notice

Intel's compilers 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 SSE2, SSE3, and SSE3 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. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804