# Intel® Open Source HD Graphics and Intel Iris™ Graphics

## Programmer's Reference Manual

For the 2014-2015 Intel Core™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Broadwell" Platform

Volume 8: Media VDBOX

October 2015, Revision 1.1

## Creative Commons License

**You are free to Share** - to copy, distribute, display, and perform the work under the following conditions:

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **No Derivative Works.** You may not alter, transform, or build upon this work.

## Notices and Disclaimers

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.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

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.

Implementations of the I2C bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

# Table of Contents

# VDBOX Registers

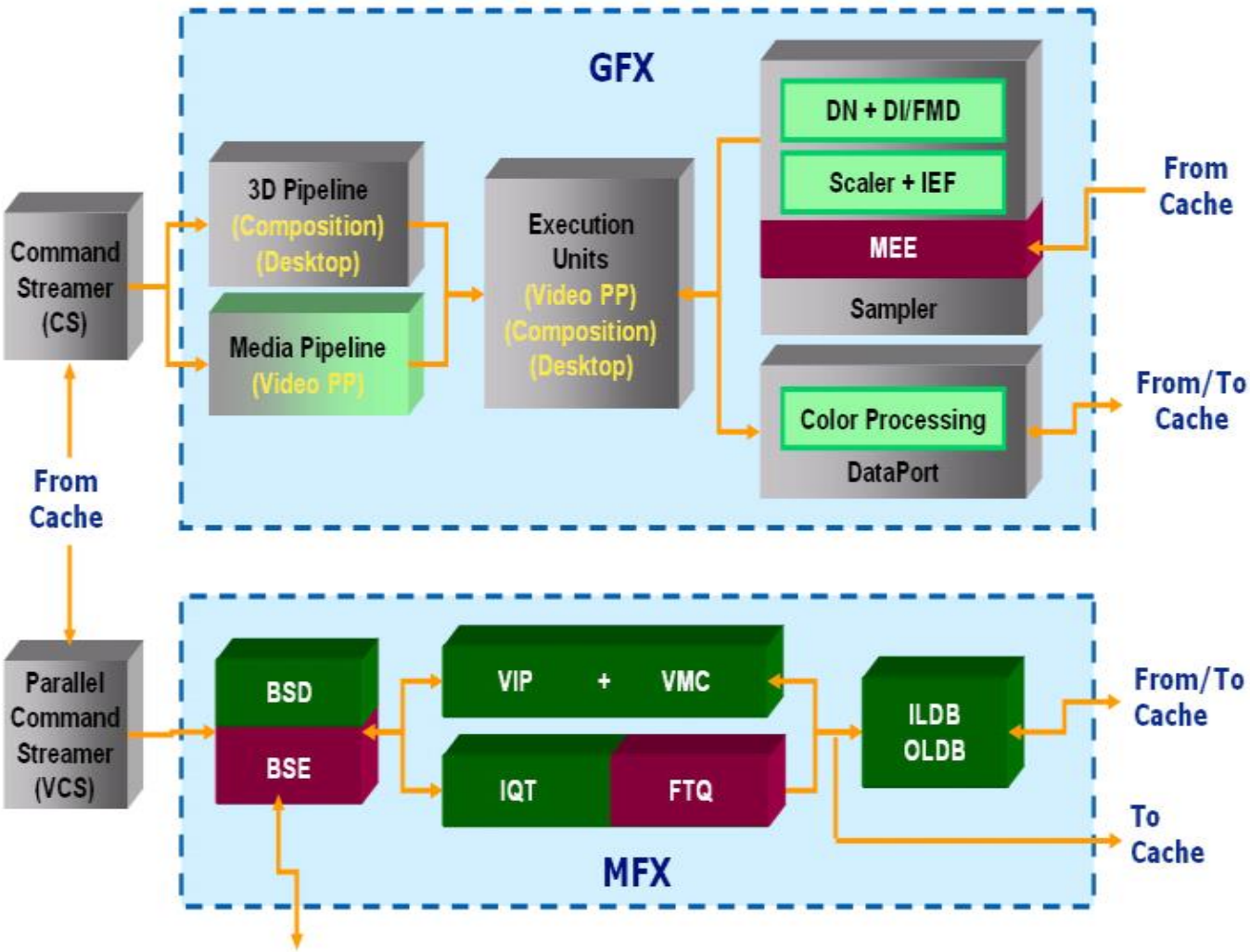
This section describes the VDBOX Command Memory Interface registers.

# MFX Architecture

This section and the following sections of Media VDBOX contain the referential documentation on the Multi-Format Codecs, or MFX for those series of chips.

## MFX Introduction

Multi-Format Codec (MFX) Engine is the hardware fixed function pipeline for decode and encoding. It includes multi-format decoding (MFD) and multi-format encoding (MFC).



# MFC Overview

Multi-Format Codec (MFX) Engine is the hardware fixed function pipeline for decode and encoding. It includes multi-format decoding (MFD) and multi-format encoding (MFC).

Many decoding function blocks in MFD such as VIP, VMC, IQT, etc, are also used in encoding mode. Two blocks, FTQ and BSE, are encoding only.

The encoding process is partitioned across host software, the GPE engine, and the MFX engine. The generation of transport layer, sequence layer, picture layer, and slice header layer must be done in the host software. GP hardware is responsible for compressing from Slice Data Layer down to all macro-block and block layers. Specifically, GPE w/ VME acceleration is for motion vector estimation, motion estimation, and code decision. The **VME**(*Video Motion Estimation*) is located next to all image processing units, such as DN (*denoise*) and DI (*deinterlace*) in sampler in GPE. MFX is for final bit packing and reconstructed picture generation.

MFC is operated concurrently with and independently from the GPE (3D/Media) pipeline with a separate command streamer. The two parallel engines have similar command protocol. They can be executed in parallel with different context. For encoding, motion search, MB mode decision, and rate control are performed using GPE pipeline resources.

MFC is implemented to achieve the following objectives:

- Compliant with next generation high definition optical video disc requirements, with sufficient performance headroom:
    - Support AVC 4:2:0 Main Profile and High Profile only (8-bit only), up to Level 4.1 resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be encoded. There is no support for Baseline, Extended, or High-10 Profiles.
- Performance requirements with MFX core frequency above 667MHz:
    - Real-time performance with 20% duty cycle or less.
    - Support concurrent decoding of two active HD bitstreams of different formats (for example, one AVC and one VC1 HD bitstream) and one active HD encoding.

As the result of this hardware partitioning, VPP and ENC are always running in GPE, and PAK is what runs exactly in MFC.

**PAK** – residue packing and entropy coding, including block transformation, quantization, data prediction, bitrate tuning and reference decoding. It delivers final packed bitstream and decoded key-frame reference:

- As the same as ENC, PAK is invoked on a Slice boundary; a single call of VPP can lead to multiple calls for PAK.
- Rate control is inside ENC and PAK only, not in VPP.
- PAK must always perform with reconstructed reference picture.

There is a general dependency of the three operation pipelines. Semaphores are inserted either according to frames or slices. The main CS will also be notified when the decoded reference is ready for the next frame set to be encoded. The detailed discussion will be found in a later section.

Host software is responsible for encoding the transport stream and all the sequence, picture, and slice layer/header in the bit-stream; the MFC system is responsible for compressing from Slice Data Layer down to all macro-block and block layers.

## Sample Algorithmic Flow

Assuming all the hardware components are given, there are infinite usage possibilities left with intention for software to decide according to its own application needs depending upon the balanced requirement of coding speed, frame latency, power-consumption, and video quality, and depending upon the usage modes and user preferences (such as low-frame-rate-high-frame-quality vs. high-frame-rate-low-frame-quality).

The last part of this chapter, we illustrate a generic sample to show how a compression algorithm can be implemented to use our hardware.

**Step 1.** Application or driver initializes the encoder with desired configuration, including speed, quality, targeted bit-rate, input video info, and output format and restrictions.

**Step 2. VPP** – Application or driver feeds VPP one frame at a time in coded order with specified frame or field type, as well as transcoding informations: motion vectors, coded complexity (i.e. bit size).
It will perform denoising and deblocking based on original and targeted bit-rate, and output additional 4 spatial variances and 2 temporal variances for each macroblock as well as the whole frame.

**Step 3. ENC** – Application or driver feeds ENC one coding slice buffer at a time including all VPP output. The frame level data is accessible to all slices.

a. Encoding setup unit (**ESE**) will set picture level quality parameters (including LUTs, and other costing functions) and set target bit-budget (TBB) and maximal bit-budget (MBB) to each macroblock based on rate-control (**RC**) scheme implemented. For B-frames, it will also make ME searching mode decision (either Fast, Slow or Uni-directional).
b. Loop over all macroblocks: calculate searching center (**MVP**) perform individual ME and IE (**MEE**). Multi-thread may be designed for HW according to a zigzag order for minimal dependency issue.
c. ENC make microblock level code decision (**CD**) outputs macroblock type, intra-mode, motion-vectors, distortions, as well as TBBs and MBBs.

**Step 4. PAK** – Application or driver feeds PAK one array of coded macroblocks covering a slice at a time, including all ENC output. Original frame buffer and reconstructed reference frame buffers are also available for PAK to access.

a. PAK may create bitstreams for all sequence, gop, picture, and slice level headers prior the first macroblock.
b. Loop over all macroblocks, accurate prediction block is constructed for either inter- or intra-predictions (**VMC** & **VIP**). If MB distortion is less than some predetermined threshold, for a B slice this step can be skipped as well as the Steps (c)-(e) and jump directly to Step (f); for a key slice the prediction calculated here will be directly used as the reference thus it jumps to Step (e) after this step.
c. Differencing the predicted block from the original block derives the residue block. Forward transformation and quantization (**FTQ**) is performed. For B slice, it will jump to Step (f) right after.

For other types of slice, Steps (d) and (e) can be performed in a thread in parallel with Step (f) and beyond.

d. This is for accurate construction of reference pictures. Inverse quantization and inverse transformation (**IQT**) are performed and added to the predictions to have the decoded blocks.
e. **ILDB** is applied accordingly to the reconstructed blocks.
f. Meanwhile macroblock codes: including its configuration info (types and modes), motion info (motion vectors and reference ids), and residual info (quantized coefficients), are collected for packing (**BSE**) in the following sub-steps:
    i. Code clean-up (in **MPR**). Check and verify Mbtype and Cbps, use Skip or Zero respectively if one can. In principal, when there are equivalent codes, use the simple one.
    ii. Drop dependency (in **MPR**). Calculate relative codes from the absolute codes by associate them with neighborhood information. All neighborhood correlations are solved in this step.
    iii. Unify symbols (in **SEC**). Translate relative codes into symbols, and table or context indices that are independent of the concept of syntax type.
    iv. Entropy coding (**VLE**) on symbols.
g. Parsing bitstream data in RBSP form (in **VLE**), and output to application or driver.
h. By the end of each picture, write out the accurate actual data size to designate buffer for ENC to access.

## Synchronization Mechanism

Encoding of a video stream can be broken down to three major steps (as explained in the previous section):

1. VPP: video-stream pre-processing
2. ENC: encoding, *that is,* code decision of inter-MVs and intra-modes
3. PAK: bit-stream packing
    a. residual calculation, transformation, and quantization
    b. code bit-stream packing
    c. reference generation of keyframes

This section describes an architectural solution to map the first two steps in the GFX engine and the last step in the MFX engine. Since this mapping involves two OS-visible engines, managing them in parallel under one application is similar to the solution in earlier generations. Each engine has its own command streamers and has mechanisms to synchronize at required levels as described in the next sub-section.

Above three steps of encoding have dependencies in processing based on

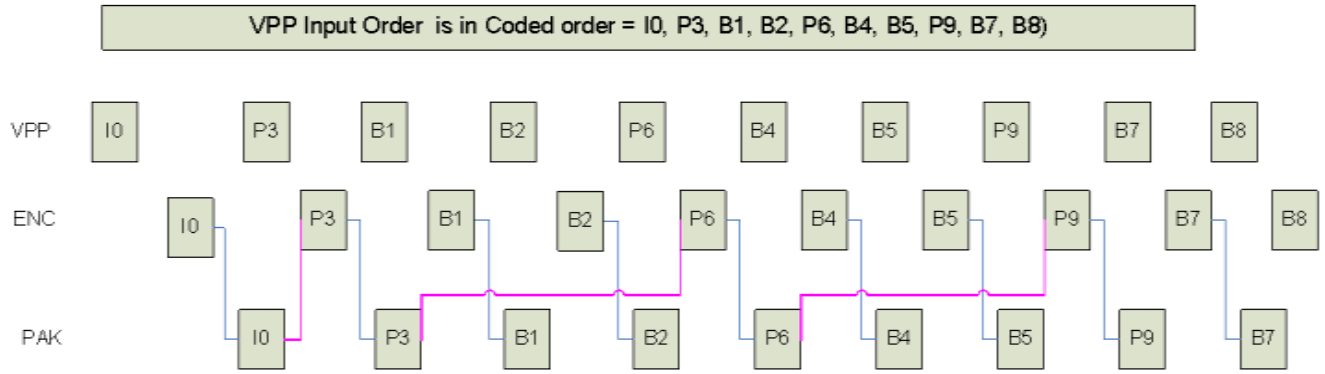
i. functional pipeline order, *i.e.* on a given frame, VPP needs to be performed first, then ENC, then PAK and finally MFD (*Multi-Format Decoding*) for key reference frame generation.
ii. I-frames are key frames for P and B, they have to be first in every pipe-stage.
iii. P-frames are key frames for B frames and therefore P frames are processed first before the dependent B frames

iv. GFX Engine is time slice to work on either VPP or ENC frame as we discussed in the previous chapter.

v. PAK + MFD are executed on the same frame in the MFX engine by macro-block level pipelining within a slice. It should be noted that for the sake of simplicity, an entire frame (potentially multiple slices) are processed in the corresponding engine and no smaller granularity of switching is allowed between the functional pipeline stages.

Three steps of the encoding can be interleaved on two engines in the following way on a frame by frame basis.

### Command Stream Synchronization

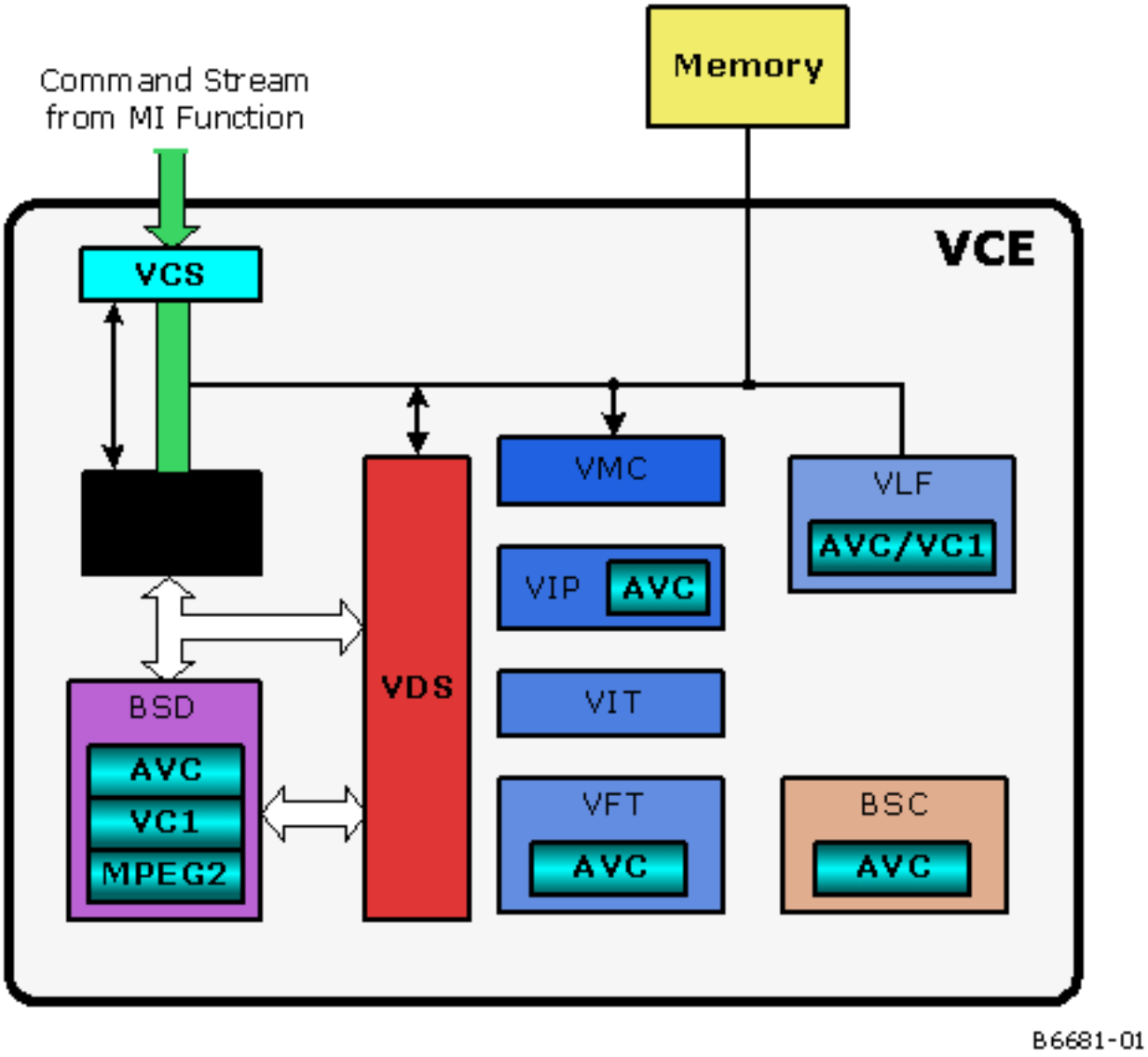



## Restrictions

MFC implementation is subject to the following limitations.

- Context switching within MFC and with Graphics Engine occurs only at frame boundary to minimize the amount of information that needs to be tracked and maintained.

## MFD Overview




When used for decoding, we also refer to the MFX Engine as the MFD Engine.

The Multi-Format Decoder (MFD) is a hardware fixed function pipeline for decoding the three video codec format and one image compression codec format: AVC (H.264), VC-1, MPEG2, and JPEG.

- Compliant with next generation high definition optical video disc requirements, with sufficient performance headroom:
  - Support AVC 4:2:0 Main and High (8-bit only) Profile only (no support for Baseline, Extended and High-10 Profiles), up to Level 5.1 (max 983,040 MB/s, max 36,864 MB/frame, and at most one dimension can reach 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
    - Allow a B-picture (frame or field) as a reference picture

- Starting with HSW, support MVC 4:2:0 Stereoscopic Progressive Profile only, up to Level 5.1 (max 983,040 MB/s per view, max 36,864 MB/frame per view, and at most one dimension can reach 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
  - Does not support interlaced video specified in the Stereoscopic Profile
- Support VC1 4:2:0 Simple, Main and Advanced Profiles, up to Level 4 (max 491,520 MB/s and max 16,384 MB/frame; max only one dimension will be at 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
  - Allow a B-field as a reference picture only in interlaced field decoding, no other modes.
- Support MPEG2 HD Main Profile (4:2:0), up to High Level (1920x1152 pixels) and up to 80 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded. No support for SNR and spatial-scalability.
  - Does not support B-picture as a reference picture.
- Support Baseline JPEG with five choma types (4:0:0, 4:1:1, 4:2:2, 4:2:0, and 4:4:4. No support for Extended DCT-based mode, Progressive mode, Loseless mode, nor Hierarchical mode.
  - H/W support 64Kx64K, but Surface State can support only up to 16kx16k

| Features | Supported | Unsupported |
|---|---|---|
| Coding processes | Baseline sequential mode:<br>• 8-bit pixel precision of source images<br>• loadable 2 AC and 2 DC Huffman tables<br>• 3 loadable quantization matrix for Y, U, V<br>• Interleaved and non-interleaved Scans<br>• Single and multiple Scans | Extended DCT-based mode, Lossless, Hierarchical modes:<br>More than 8 bit pixel resolution, progressive mode, arithmetic coding, 4 AC and 4 DC Huffman tables (extended mode), predictive process (lossless), multiple frames (hierarchical) |
| Number of image channels | 1 for grey image<br>3 for Y, Cb, Cr color image | 4-th channel (usually alpha blending image) |
| Image resolution | Arbitrary image size up to 16K * 16K | Larger than 16K * 16K (64K * 64K is max. in the JPEG standard) |
| Chroma subsampling ratio | Chroma 4:0:0 (grey image)<br>Chroma 4:1:1<br>Chroma 4:2:0<br>Chroma horizontal 4:2:2<br>Chroma vertical 4:2:2<br>Chroma 4:4:4 | Any other arbitrary ratio, e.g., 3:1 subsampled chroma |
| Additional feature (post-processing) | Image rotation: 90/180/270 degrees | |

    - H/W does not impose restriction on picture frame aspect ratio, but is bounded by a max 256 MBs (4096 pixels) per dimension programmable at the H/W interface specifications.
        - For example, supporting HD video resolution 1920x1080/60i, 1920x1080/24p, 1280x720/60p
- Performance requirements with MFX core frequency above 1GHz
    - Real-time performance around 10% duty cycle
    - Support concurrently decoding of at least two active HD bitstreams of different formats (For example, one AVC and one VC1 HD bitstream)
- The parsing of transport layer and sequence layer is not performed in this hardware, and is required to be done in the host software.
- The MFD hardware pipeline is operated concurrently with and independently from the Graphics (3D/Media) pipeline with separate command streamer. The two parallel engines are designed with the similar command protocol. They can be executed in parallel with different context.
- Local storages and buffers along the hardware pipeline are kept at minimum. For example, there is no on-die row-store memory. They are resided on the system memory. MFD is designed to hide the memory access latency (in both the row stores and in the motion compensation units) in maximizing its decoding throughput.
- Support the following operating modes:
    - VLD mode - operation starts from entropy decoding of the compressed bit stream (parsing Slice Header and Slice Data Layer in AVC, Picture layer, Slice layer and MB Layer in VC-1, and MB-layer in MPEG2), all the way, to the reconstruction of display picture, including in-loop and out-loop deblocking, if any.
        - Streamout mode - a new feature of the VLD mode in assisting transcoding during decoding. Selected uncompressed data (e.g. per MB MV information) will be sent out to the EU and the ME engine (resided on the Sampler of the 3D Gx Pipeline) for encoding into a different format or for the purpose of transcaling and transrating. In addition, the uncompressed result may continue to be processed by the rest of pipeline as in VLD mode to generate the display picture for transcoding. That is, while intermediate data are streaming out to the memory, the MFD Engine continues its decoding as ususal.
        - For JPEG, only VLD mode is supported (No IT mode). Host software decodes Frame and Scan layers (down to Scan header in the JPEG bit stream syntax) and sends all the corresponding information and Scan payload to the MFD hardware pipeline.
    - IT mode - when host software has already performed all the bit stream parsing of the compressed data and packaging the uncompressed result into a specific format (as a sequence of per-MB record) stored in memory. The hardware pipeline will fetch one MB record at a time and perform the rest of the decoding process as in VLD mode
    - Host software (Application) is responsible for parsing and decoding all the transport and program layers, and all sequence layers. These parameters are passed to Driver and forwarded to H/W as needed through different STATE commands. Host software is also responsible for separating non-video data (audio, meta and user data) from sending to H/W.

        - MFD Engine is only responsible for macro-block and block layers decoding, plus certain level of header decoding. For AVC MFD starts decoding from Slice Header; for VC1, MFD starts decoding from Picture Header, and for MPEG2 decoding starts from MB Layer only.
        - For JPEG, MFD is responsible for ECS and block layers decoding.
- Support bitstream formats (compressed video data) for each codec
    - AVC - 2 formats
    - MVC - 2 formats
        - DXVA2 MVC Short Slice Format (HSW onwards)
        - DXVA2 AVC Long Slice Format Specification (exactly the same as AVC)
    - VC1 - 2 formats
        - Fully compliant to Picture Parameter and Slice Control Parameter interface definition
    - MPEG2
        - MB Layer only, according to DXVA 1 Specification
    - JPEG
        - ECS Layer
- The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.
    - In particularly, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition.
- To activate the AVC deblocker logic for incoming uncompressed 4:2:0-only video stream, one can pack the uncompressed video stream to compliant with the IPCM MB data format (including ILDB control information) and feed them into the MFD engine in IT mode. Since the MFD Engine is in IPCM mode, transformation, inter and intra processing are all inactive.

Start Code Detection and removal are done in the CPU, but the Start Code Emulation Prevention Byte is detected and removed by the front end logic in the MFD. The bitstream format for each codec and for each mode is specified in this document.

Codec specific information are based on the following released documents from third parties :

- Draft of Version 4 of H.264/AVC (ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 part 10) Advanced Video Coding); JVT-O205d1.doc; dated 2005-05-30
- Final Draft SMPTE Standard : VC1 Compressed Video Bitstream Format and Decoding Process, SMPTE 421M, dated 2006-1-6; PDF file.
- MPEG2 Recommendation ITU T H.262 (1995 E), ISO/IEC 13818-2: 1995 (E); doc file.
- Digital Compression and Coding of Continuous-tone Still Images, ITU-T Rec. T.81 and ISO/IEC 10918-1: Requirements and guidelines September 18 1992; itu-t81[1].pdf

## MFD Memory Interface

The Memory Arbitrator follows the pre-defined arbitration policy (as indicated in the following listing P0 to P11, in which P0 is the highest priority) to select the next memory request to service, then it will perform the TLB translation (translation to physical address in memory), and make the actual request to memory.

The Memory Arbitration unit is also responsible for capturing the return data from memory (read request) and forward it to the appropriate unit along the MFD Engine.

- Read streams: (all 64B requests)
  - Commands for BSD : linear ( including indirect data) (P0)
  - Indirect DMA (P1)
  - Row store for BSD: linear (P5)
  - Row store for MPR: linear (P6)
  - MC ref cache fetch : tiled (P2)
  - Intra row store: linear (P9)
  - ILDB row store: linear (P10)
- Write streams: (all 64B requests)
  - Row store write for BSD: linear and can avoid partial writes (P3)
  - Row store write for MPR: linear and can avoid partial writes (P4)
  - Intra row store write: linear and can avoid partial writes (P7)
  - ILDB row store write: linear and can avoid partial writes (P8)
  - Final dest writes: tiled and can potentially be partial, two ways to avoid these partials: 1) either write garbage and buffers are aligned or 2) read-modify writes for dribble end of line cases (P11)

## MFD Codec-Specific Commands

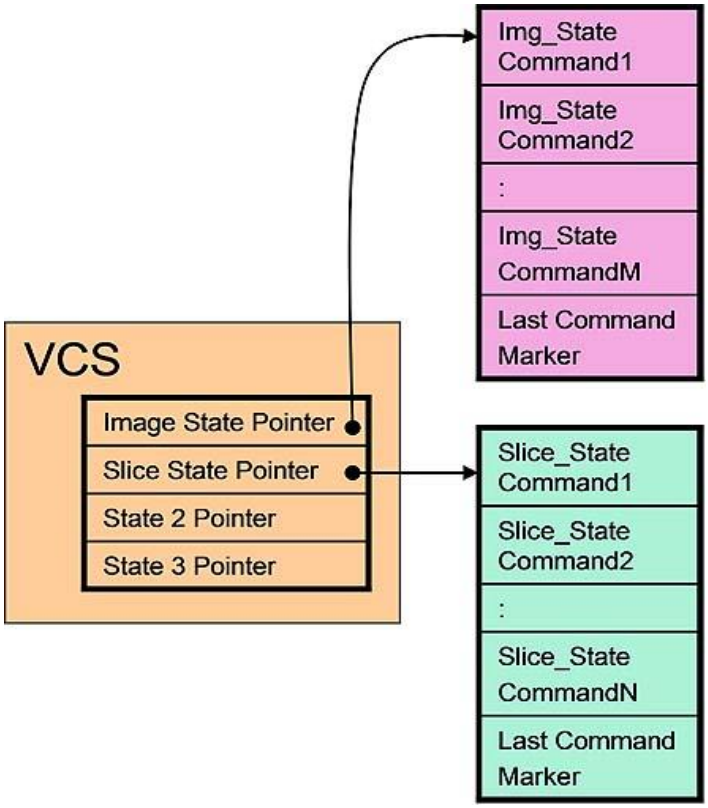
MFD hardware pipeline supports 3 different codec standards : AVC, VC1 and MPEG2. To make the interface flexible, each codec is designed with its own set of commands.

There are two categories of commands for each codec format : one set for VLD mode and one set for IT mode.

# MFX State Model

The parallel video engine (PVE) supports two state delivery models: inline state model and indirect state model. For inline state model, the state commands (*_STATE) can be issued in batch buffers or ring buffers directly preceding object commands (*_OBJECT). In the indirect state model, the state commands are not placed in the batch buffers or ring buffers. Instead Indirect State Buffers provide state information (in the form of the above mentioned state commands) for the MFX pipeline. See MFX_STATE_POINTER for more details.

VCS (aka BCS) handles the difference of the two state delivery models. Therefore, the MFX pipeline always sees the state commands in both models. However, MFX hardware supports additional context save/restore of 'dynamic states'. Dynamic states are the internal signals that are persistent. This could be the CABAC context for macroblock encoding.




## MFX State Model

The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.

- In particular, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition.

# MFX Interruptability Model

MFX encoding and the encoding pipeline do not support interruption. All operations are frame based. Interrupts can only occur between frames; the driver will submit all the states at the beginning of each frame. Any state kept across frames is in MMIO registers that should be read between frames.

Software submits without any knowledge of where the parser head pointer is located. Also there is a non-deterministic amount of time for the new context to reach the command streamer. However, the state model for the MFX engine requires software to know exactly what state the pipeline is in at all times. This introduces cases where a preemption could occur during or after a state change without software ever knowing the state saved out to memory on the context switch.

Also, preemption is only allowed during the last macroblock in a row. Hardware cannot always perform a context switch when the new context is seen by the hardware. To avoid a switch during an invalid macroblock and to keep the state synchronized with software, there are two commands available that are used. MI_ARB_ON_OFF disables and enables preemption while MFX_WAIT ensures the context switch, if needed, preempts during macroblock execution. Below illustrates an example assuming VC1 VLD mode.

| Command Ring/Batch | Notes |
|---|---|
| MI_ARB_ON_OFF = OFF | Disable preemption |
| S1 | Inline or indirect state cmd 1 |
| S2 | Inline or indirect state cmd 2 |
| S3 | Inline or indirect state cmd 3 |
| XXXX_OBJECT | Slice |
| MI_ARB_ON_OFF = ON | Enable preemption |
| MFX_WAIT | Allow preemption to occur while XXXX_OBJECT executes |
| MI_ARB_ON_OFF = OFF | Since arbitration is off again, state commands are allowed below |
| S4 | Inline or indirect state cmd 4 |
| S5 | Inline or indirect state cmd 5 |
| S6 | Inline or indirect state cmd 6 |
| XXXX_OBJECT | Slice |
| MI_ARB_ON_OFF = ON | Enable preemption |
| MFX_WAIT | Allow preemption to occur while XXXX_OBJECT executes |
| MI_ARB_ON_OFF = OFF | Since arbitration is off again, state commands are allowed below |

Note that store DW commands may execute inside the preemption enabling window if needed.

# Decoder Input Bitstream Formats

## AVC Bitstream Formats – DXVA Short

Bitstream Buffer Address starts after the 3-byte start code, i.e. starts (and includes) at the NAL Header Byte. This byte must not be included in the Emulation Byte Detection Process.

## AVC Bitstream Formats – DXVA Long

Bitstream Buffer Address starts after the 3-byte start code, i.e. starts (and includes) at the NAL Header Byte. This byte must not be included in the Emulation Byte Detection Process. Application will provide the Slice Header Skip Byte count (not including any possible Emulation Prevention Byte).

## VC1 Bitstream Formats – Intel Long

Bitstream starts right at the MB layer, with any emulation byte crossing the header and MB layer being removed by application and the data length is adjusted.

## MPEG2 Bitstream Formats – DXVA1

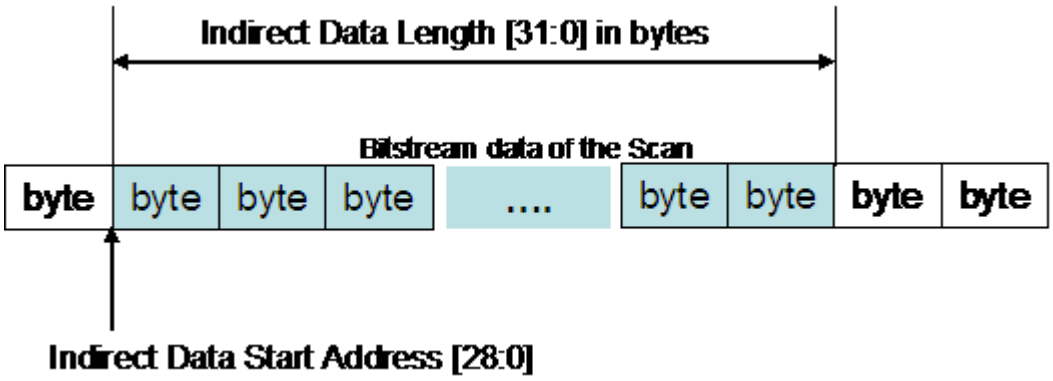
Bitstream buffer starts right at the very first MB data.

## JPEG Bitstream Formats – Intel

Bitstream buffer starts right at the very first MCU data of each Scan.

The indirect data start address in MFD_JPEG_BSD_OBJECT specifies the starting Graphics Memory address of the bitstream data that follows the Scan header. It provides the byte address for the first MCU of the Scan. Different from MFD_MPEG2_BSD_OBJECT command, First MCU Bit Offset does not need to be specified because it is always set to zero.

### Indirect data buffer for a Scan




The indirect data length in MFD_JPEG_BSD_OBJECT provides the length in bytes of the bitstream data for the Scan excluding Scan header. It includes the first byte of the first macroblock and the last byte of the last macroblock in the Scan. The Figure illustrates these parameters for a slice data.

## Concurrent Multiple Video Stream Decoding Support

The natural place for switching across multiple streams is at the Slice boundary. Each Slice is a self-sustained unit of compressed video data and has no dependency with its neighbors (except for the Deblocking process). In addition, there is no interruptability within a Slice. However, when ILDB is invoked, the processing of some MBs will require neighbour MB information that crosses the Slice boundary. Hence, to limit the buffering requirement, in this version of hardware design, stream switching can only be performed at the picture boundary instead.

# MFX Codec Commands Summary

| DWord | Bit | Description |
|---|---|---|
| 0 | 31:29 | **Instruction Type =** GFXPIPE = 3h |
| | 28:16 | **3D Instruction Opcode =** PIPELINE_SELECT<br>GFXPIPE[28:27 = 1h, 26:24 = 1h, 23:16 = 04h] (Single DW, Non-pipelined) |
| | 15:1 | **Reserved:** MBZ |
| | 0 | **Pipeline Select**<br>0: 3D pipeline is selected<br>1: Media pipeline is selected |

| Pipeline Type (28:27) | Opcode (26:24) | Sub Opcode (23:16) | Command | Definition Chapter |
|---|---|---|---|---|
| **VC1 State** | | | | |
| 2h | 5h | 0h | VC1_BSD_PIC_STATE | VC1 BSD |
| 2h | 5h | 1h | Reserved | n/a |
| 2h | 5h | 2h | Reserved | n/a |
| 2h | 5h | 3h | VC1_BSD_BUF_BASE_STATE | VC1 BSD |
| 2h | 5h | 4h | Reserved | n/a |
| 2h | 5h | 5h-7h | Reserved | n/a |
| **VC1 Object** | | | | |
| 2h | 5h | 8h | VC1_BSD_OBJECT | VC1 BSD |
| 2h | 5h | 9h-FFh | Reserved | n/a |

| Pipeline Type (28:27) | Opcode (26:24) | Sub Opcode (23:16) | Command | Definition Chapter |
|---|---|---|---|---|
| **State** | | | | |
| 2h | 6h | 2h-7h | Reserved | N/A |
| **Object** | | | | |
| 2h | 6h | 9h-FFh | Reserved | N/A |

Note that it is possible for a command to appear in both IMAGE and SLICE state buffer, e.g. QM_STATE for JPEG can be issued at frame level or scan/slice level.

| Pipeline Type (28:27) | Opcode (26:24) | SubopA (23:21) | SubopB (20:16) | Command | Chapter | Recommended Indirect State Pointer Map | Interruptible? |
|---|---|---|---|---|---|---|---|
| | **MFX Common** | **Common** | | | | | |
| 2h | 0h | 0h | 0h | MFX_PIPE_MODE_SELECT | MFX | IMAGE | No |
| 2h | 0h | 0h | 1h | MFX_SURFACE_STATE | MFX | IMAGE | No |
| 2h | 0h | 0h | 2h | MFX_PIPE_BUF_ADDR_STATE | MFX | IMAGE | No |
| 2h | 0h | 0h | 3h | MFX_IND_OBJ_BASE_ADDR_STATE | MFX | IMAGE | No |
| 2h | 0h | 0h | 4h | MFX_BSP_BUF_BASE_ADDR_STATE | MFX | IMAGE | No |
| 2h | 0h | 0h | 6h | MFX_ STATE_POINTER | MFX | IMAGE | No |
| 2h | 0h | 0h | 7h | MFX_QM_STATE | MFX | IMAGE/SLICE | No |
| 2h | 0h | 0h | 8h | MFX_FQM_STATE | MFX | IMAGE | No |
| 2h | 0h | 0h | 9h | MFX_DBK_OBJECT | MFX | IMAGE | No |
| 2h | 0h | 0h | A-1Eh | Reserved | n/a | n/a | No |
| | **MFX Common** | **Dec** | | | | | |
| 2h | 0h | 1h | 0-8h | Reserved | n/a | n/a | n/a |
| 2h | 0h | 1h | 9h | MFD_ IT_OBJECT | MFX | n/a | No |
| 2h | 0h | 1h | A-1Fh | Reserved | n/a | n/a | n/a |
| | **MFX Common** | **Enc** | | | | | |
| 2h | 0h | 2h | 0-7Fh | Reserved | n/a | n/a | n/a |
| 2h | 0h | 2h | 8h | MFX_PAK_INSERT_OBJECT | MFX | n/a | No |
| 2h | 0h | 2h | 9h | Reserved | n/a | n/a | n/a |
| 2h | 0h | 2h | Ah | MFX_STITCH_OBJECT | MFX | n/a | No |
| 2h | 0h | 2h | B-1Fh | Reserved | n/a | n/a | n/a |
| | **AVC/ MVC** | **Common (State)** | | | | | |
| 2h | 1h | 0h | 0h | MFX_AVC_IMG_STATE | MFX | IMAGE | n/a |
| 2h | 1h | 0h | 1h | Reserved | n/a | n/a | n/a |
| 2h | 1h | 0h | 2h | MFX_AVC_DIRECTMODE_STATE | MFX | SLICE | n/a |
| 2h | 1h | 0h | 3h | MFX_AVC_SLICE_STATE | MFX | SLICE | n/a |
| 2h | 1h | 0h | 4h | MFX_AVC_REF_IDX_STATE | MFX | SLICE | n/a |
| 2h | 1h | 0h | 5h | MFX_AVC_WEIGHTOFFSET_STATE | MFX | SLICE | n/a |

| Pipeline Type (28:27) | Opcode (26:24) | SubopA (23:21) | SubopB (20:16) | Command | Chapter | Recommended Indirect State Pointer Map | Interruptible? |
|---|---|---|---|---|---|---|---|
| 2h | 1h | 0h | 9 | Reserved | n/a | n/a | n/a |
| 2h | 1h | 0h | D-1Fh | Reserved | n/a | n/a | n/a |
| | AVC/ MVC | **Dec** | | | | | |
| 2h | 1h | 1h | 0-5h | Reserved | MFX | n/a | n/a |
| 2h | 1h | 1h | 6h | MFD_AVC_DPB_STATE | MFX | IMAGE | n/a |
| 2h | 1h | 1h | 7h | MFD_AVC_SLICEADDR_OBJECT | MFX | n/a | n/a |
| 2h | 1h | 1h | 8h | MFD_AVC_BSD_OBJECT | MFX | n/a | No |
| 2h | 1h | 1h | 9-1Fh | Reserved | n/a | n/a | n/a |
| | **AVC/ MVC** | **Enc** | | | | | |
| 2h | 1h | 2h | 0-8h | Reserved | n/a | n/a | n/a |
| 2h | 1h | 2h | 9h | MFC_AVC_PAK_OBJECT | MFX | n/a | No |
| 2h | 1h | 2h | A-1Fh | Reserved | n/a | n/a | n/a |
| | **AVC/ MVC** | **Extension** | | | | | |
| | **VC1** | **Common (State)** | | | | | |
| 2h | 2h | 0h | 0h | Reserved | n/a | n/a | n/a |
| 2h | 2h | 0h | 1h | MFX_VC1_PRED_PIPE_STATE | MFX | IMAGE | n/a |
| 2h | 2h | 0h | 2h | MFX_VC1_DIRECTMODE_STATE | MFX | SLICE | n/a |
| 2h | 2h | 0h | 3-1Fh | Reserved | n/a | n/a | n/a |
| | **VC1** | **Dec** | | | | | |
| 2h | 2h | 1h | 0h | MFD_VC1_SHORT_PIC_STATE | MFX | IMAGE | n/a |
| 2h | 2h | 1h | 1h | MFD_VC1_LONG_PIC_STATE | MFX | IMAGE | n/a |
| 2h | 2h | 1h | 2-7h | Reserved | n/a | n/a | n/a |
| 2h | 2h | 1h | 8h | MFD_VC1_BSD_OBJECT | MFX | n/a | No |
| 2h | 2h | 1h | 9-1Fh | Reserved | n/a | n/a | n/a |
| | **VC1** | **Enc** | | | | | |
| 2h | 2h | 2h | 0-1Fh | Reserved | n/a | n/a | n/a |
| | **MPEG2** | **Common (State)** | | | | | |

| Pipeline Type (28:27) | Opcode (26:24) | SubopA (23:21) | SubopB (20:16) | Command | Chapter | Recommended Indirect State Pointer Map | Interruptible? |
|---|---|---|---|---|---|---|---|
| 2h | 3h | 0h | 0h | MFX_MPEG2_PIC_STATE | MFX | IMAGE | n/a |
| 2h | 3h | 0h | 1-1Fh | Reserved | n/a | n/a | n/a |
| | **MPEG2** | **Dec** | | | | | |
| 2h | 3h | 1h | 1-7h | Reserved | n/a | n/a | n/a |
| 2h | 3h | 1h | 8h | MFD_MPEG2_BSD_OBJECT | MFX | n/a | No |
| 2h | 3h | 1h | 9-1Fh | Reserved | n/a | n/a | n/a |
| | **MPEG2** | **Enc** | | | | | |
| 2h | 3h | 2h | 0-2h | Reserved | n/a | n/a | n/a |
| 2h | 3h | 2h | 3h | MFC_MPEG2_PAK_OBJECT | | | |
| 2h | 3h | 2h | 3-8h | Reserved | | | |
| 2h | 3h | 2h | 9h | MFC_MPEG2_SLICEGROUP_STATE | | | |
| 2h | 3h | 2h | A-1Fh | Reserved | | | |
| | **VP8** | **Common (State)** | | | | | |
| 2h | 4h | 0h | 0h | MFX_VP8_PIC_STATE | MFX | IMAGE | n/a |
| | **VP8** | **Dec** | | | | | |
| 2h | 4h | 1h | 8h | MFD_VP8_BSD_OBJECT | MFX | IMAGE | No |
| | **VP8** | **Enc** | | | | | |
| 2h | 4h | 2h | | Reserved | | | |
| | **JPEG** | **Common** | | | | | |
| 2h | 7h | 0h | 0h | MFX_JPEG_PIC_STATE | MFX | IMAGE | No |
| 2h | 7h | 0h | 1h | Reserved | n/a | n/a | n/a |
| 2h | 7h | 0h | 2h | MFX_JPEG_HUFF_TABLE_STATE | MFX | IMAGE | No |
| 2h | 7h | 0h | 3-1Fh | Reserved | n/a | n/a | n/a |
| | **JPEG** | **Common** | | | | | |
| 2h | 7h | 0h | 0h | MFX_JPEG_PIC_STATE | MFX | IMAGE | No |
| 2h | 7h | 0h | 1h | Reserved | n/a | n/a | n/a |
| 2h | 7h | 0h | 2h | MFX_JPEG_HUFF_TABLE_STATE | MFX | IMAGE | No |
| 2h | 7h | 0h | 3-1Fh | Reserved | n/a | n/a | n/a |
| | **JPEG** | **Dec** | | | | | |
| 2h | 7h | 1h | 1-7h | Reserved | MFX | n/a | n/a |

| Pipeline Type (28:27) | Opcode (26:24) | SubopA (23:21) | SubopB (20:16) | Command | Chapter | Recommended Indirect State Pointer Map | Interruptible? |
|---|---|---|---|---|---|---|---|
| 2h | 7h | 1h | 8h | MFD_JPEG_BSD_OBJECT | MFX | MCU | No |
| 2h | 7h | 1h | 9-1Fh | Reserved | MFX | n/a | n/a |
| | **JPEG** | **Enc** | | | | | |
| 2h | 7h | 2h | 0-1Fh | Reserved | MFX | n/a | n/a |

### MMIO Space Registers

| Range Start | Range End | Unit owner |
|---|---|---|
| 00002000 | 00002FFF | Render/Generic Media Engine |
| 00004000 | 00004FFF | Render/Generic Media Graphics Memory Arbiter |
| 00005000 | 0000517F | |
| 00006000 | 00007FFF | Reserved |
| 00012000 | 000123FF | MFX Control Engine (Video Command Streamer) |
| 00012400 | 00012FFF | Media Units (VIN unit) |
| 00014000 | 00014FFF | MFX Memory Arbiter |
| 00022000 | 00022FFF | Blitter Engine |
| 00024000 | 00024FFF | Blitter Memory Arbiter |
| 00030000 | 0003FFFF | Reserved |
| 00100000 | 00107FFF | Fence Registers |
| 00140000 | 0017FFFF | MCHBAR (SA) |

### Memory Interface Command Map

| | |
|---|---|
| 04h Opcode (28:23) | MI_FLUSH |

## MFX Decoder Commands Sequence

The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.

In particular, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition

## Examples for AVC

The following gives a sample command sequence programmed by a driver

a) For Intel or DXVA2 AVC Long Slice Bitstream Format

MFX_PIPE_MODE_SELECT

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

MFX_QM_STATE

**VLD mode: MFX_AVC_PICID_STATE**

MFX_AVC_IMG_STATE

MFX_AVC_DIRECTMODE_STATE

MFX_AVC_REF_IDX_STATE

MFX_AVC_WEIGHTOFFSET_STATE

MFX_AVC_SLICE_STATE

VLD mode: MFD_AVC_BSD_OBJECT

IT mode: MFD_IT_OBJECT

MI_FLUSH

b) For DXVA2 AVC Short Slice Bitstream Format (for VLD mode only)

MFX_PIPE_MODE_SELECT

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

MFD_AVC_DPB_STATE

**VLD mode: MFX_AVC_PICID_STATE**

MFX_AVC_IMG_STATE

MFX_QM_STATE

MFX_AVC_DIRECTMODE_STATE

VLD mode : MFD_AVC_SLICEADDR_OBJECT

VLD mode: MFD_AVC_BSD_OBJECT

VLD mode : MFD_AVC_BSD_SLICEADDR_OBJECT

VLD mode: MFD_AVC_BSD_OBJECT

... repeat these four commands N-1 times for a N-slice picture

VLD mode: MFD_AVC_BSD_OBJECT (for the last slice of the picture)

MI_FLUSH

## Examples for VC1

The following gives a sample command sequence programmed by a driver

a) For Intel Proprietary Long Bitstream Format

MFX_VC1_DIRECTMODE_STATE

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_LONG_PIC_STATE

VLD mode: MFD_VC1_BSD_OBJECT

IT mode: MFD_IT_OBJECT

MI_FLUSH

b) For DXVA2 VC1 Compliant Bitstream Format (for VLD mode only)

MFX_VC1_DIRECTMODE_STATE

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_SHORT_PIC_STATE

VLD mode: MFD_VC1_BSD_OBJECT

MI_FLUSH

c) For DXVA2 VC1 Compliant Bitstream Format (for VLD mode only), and field pair picture

Batch buffer for top-field

states....

Slice_objs...

MI_flush

store register immediate (if VC1 short format with interlaced field pic)

MI_flush

Batch buffer for bottom field

load register immediate (if VC1 short format with interlaced field pic)

MI_flush

states....

Slice_objs...

MI_flush

## Examples for JPEG

The following gives a sample command sequence programmed by a driver

Programmed once at the start of decoding

MFX_PIPE_MODE_SELECT

MFX_PIPE_SURFACE_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_JPEG_PIC_STATE

Programmed at the start of Frame or Scan (These commands can be sent multiple times either before MFX_JPEG_PIC_STATE or before MFD_JPEG_BSD_OBJECT)

MFX_JPEG_HUFF_TABLE

MFX_QM_STATE

Programmed per Scan (These commands can be sent multiple times depending on each bit stream)

MFD_JPEG_ BSD_OBJECT

MI_FLUSH

# MFX Pipe Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

**MFX_STATE_POINTER**

**MFX_PIPE_MODE_SELECT**

The Encoder Pipeline Modes of Operation (Per Frame):

1. PAK Mode: VCS-command driven, setup by driver. Like the IT mode of decoder, it is executed on a per-MB basis. Hence, each PAK Object command corresponds to coding of only one MB.

a. Normal Mode (including transcoding): receive per-MB control and data (MV, mb_type, cbp, etc.). It generates the output compressed bitstream as well as the reconstructed reference pictures, one MB at a time, for later use.

b. Encoder StreamOut Mode: to provide per-MB, per-Slice and per-Frame coding result and information (statistics) to the Host, Video Preprocessing Unit and ENC Unit to enhance their operations.

The Decoder Pipeline Modes of Operation (Per Frame):

1. VLD Mode: The output from the BSD (weight&offset/coeff/motion vectors record) can be sent in part (as specified) and to the remaining fixed function hardware pipeline to complete the decoding processing. The driver specifies through MFD commands of what to send out from the BSD unit and where to send the BSD output.

a. For transcoding (including transrating and transcaling), part of the BSD output (a series of per-MB record) can be sent to memory for further processing to encode into a difference output format. This function is named as StreamOut. When StreamOut is active, not all MB information needs to be sent, only MVs and selective MB coding information.

2. IT Mode: In this mode, the BSD is not invoked. Instead host performs all the bitstream decoding and parsing; and the result are saved into memory in a specific per-MB record format. The MFD Engine VCS reads in these records one at time and finish the rest of the decoding (IT, MC, IntraPred and ILDB).

a. MB information is organized into two indirect data buffers, one for MVs and one for residue coefficients. As such, two indirect base address pointers are defined.

**MFX_SURFACE_STATE**

**MFX_PIPE_BUF_ADDR_STATE >[BDW]**

**MFX_IND_OBJ_BASE_ADDR_STATE**

### MFX_BSP_BUF_BASE_ADDR_STATE

### MFX_PAK_INSERT_OBJECT

### MFX_STITCH_OBJECT

### MFX_QM_STATE

| Bits | 31:24 | 23:16 | 15:8 | 7:0 |
|---|---|---|---|---|
| Dword 1 | QuantMatrix[0][3] | QuantMatrix[0][2] | QuantMatrix[0][1] | QuantMatrix[0][0] |
| Dword 2 | QuantMatrix[0][7] | QuantMatrix[0][6] | QuantMatrix[0][5] | QuantMatrix[0][4] |
| Dword 3 | QuantMatrix[1][3] | QuantMatrix[1][2] | QuantMatrix[1][1] | QuantMatrix[1][0] |
| ... | ... | ... | ... | ... |
| Dword 16 | QuantMatrix[7][7] | QuantMatrix[7][6] | QuantMatrix[7][5] | QuantMatrix[7][4] |

### MFX_FQM_STATE

This is a frame-level state. Reciprocal Scaling Lists are always sent from the driver regardless whether they are specified by an application or the default/flat lists are being used. This is done to save the ROM (to store the default matrices) inside the PAK Subsystem. Hence, the driver is responsible for determining the final set of scaling lists to be used for encoding the current slice, based on the AVC Spec (Fall-Back Rules A and B). For encoding, there is no need to send the qm_list_flags[i], i=0 to7 and qm_present_flag to the PAK, since Scaling Lists syntax elements are encoded above Slice Data Layer.

FQM Reciprocal Scaling Lists elements are 16-bit each, conceptually equal to 1/ScaleValue. QM matrix elements are 8-bit each, equal to ScaleValue. However, in AVC spec., the Reciprocal Scaling Lists elements are not exactly equal to one-over of the corresponding Scaling Lists elements. The numbers are adjusted to simplify hardware implementation.

For all the description below, a scaling list set contains 6 4x4 scaling lists (or forward scaling lists) and 2 8x8 scaling lists (or forward scaling lists).

In MFX PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order as shown in MFX_AVC_QM_STATE. But the Forward Q scaling lists are sent in transport form, i.e. column-wise raster order (column-by-column) to simplify the H/W.

Precisely, if the reciprocal forward scaling matrix is F[4][4], then the 16 word of the matrix will be set as the following:

| | bits 0-15 | bits 16-31 |
|---|---|---|
| DW0 | F[0][0] | F[1][0] |
| DW1 | F[2][0] | F[3][0] |
| DW2 | F[0][1] | F[1][1] |
| DW3 | F[2][1] | F[3][1] |
| DW4 | F[0][2] | F[1][2] |
| DW5 | F[2][2] | F[3][2] |
| DW6 | F[0][3] | F[1][3] |
| DW7 | F[2][3] | F[3][3] |

## Bitplane Buffer

Bitplane coding is used in seven different cases in VC-1, although not all the seven syntax elements are present in the same picture header at the same time. The following list shows which syntax elements are coded as bitplanes in each picture header:

Progressive I and BI picture headers in AP: ACPRED, OVERFLAGS
Field interlace I and BI picture headers in AP: ACPRED, OVERFLAGS
Frame interlace I and BI picture headers in AP: FIELDTX, ACPRED, OVERFLAGS

Frame interlace P picture headers in AP: SKIPMB
Progressive P picture headers in SP and MP: MVTYPEMB, SKIPMB
Progressive P picture headers in AP: MVTYPEMB, SKIPMB

Field interlace B picture headers in AP: FORWARDMB
Frame interlace B picture headers in AP: DIRECTMB, SKIPMB
Progressive B picture headers in AP: DIRECTMB, SKIPMB
Progressive B picture headers in MP: DIRECTMB, SKIPMB

There are also seven different modes of coding the bitplane information. Except when the bitplane is coded in raw mode, the bitplane is decoded by the host and provided to the hardware in the bitplane buffer.

Since at most three bitplanes are encoded in any picture header, instead of using a complete byte for signaling the values of all the seven possible bitplanes for each MB, a more efficient approach is used with each byte divided in two nibbles and each nibble carries the data of up to four bitplanes for one MB.

| PictureType | Bits 3, 7 | Bit 2, 6 | Bits 1, 5 | Bits 0, 4 |
|---|---|---|---|---|
| **I or BI** | 0 | OVERFLAGS | ACPRED | FIELDTX |
| **P** | 0 | MVTYPEMB | SKIPMB | 0 |
| **B** | 0 | FORWARDMB | SKIPMB | DIRECTMB |

The bytes containing the above defined nibbles are stored in the bitplane buffer in raster scan order. The bitplane buffer is a linear buffer with a buffer pitch (as defined by Bitplane Buffer Pitch field in VC1_BSD_PIC_STATE command). When the number of macroblock in a row is odd, the last byte of the row containing the last macroblock in bits 0-3. The first macroblock of the next row starts at the next pitch offset from the first macroblock of the current row.

The bitplane buffer structure must be sent once per picture only if there is one or more syntax elements coded as bitplanes in the picture header.

# Video Codecs

The following sections contain the various registers for video codec support. Specifically, the codec types supported are:

| Supported Codec Types |
|---|
| Advanced Video Coding (AVC)/ H.264/MPEG-4 Part 10 (MVC) |
| MPEG-2 (H.222/H.262) — Used in Digital Video Broadcast and DVDs |
| VC1 — SMPTE 421M, known informally as VC-1 |
| VP8 — Video compression format |
| JPEG and MJPEG — A video format in which video gram or interlaced field of a digital video sequence is compressed separately as a JPEG image |
| Other Codec Functions |

**Internal Media Rowstore table** – An internal Media Rowstore Storage is added to reduce memory read/write to save power. If the internal Media Rowstore exists, driver should use the storage as the following table indicates.

## AVC (H.264)

### AVC Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

**MFX_AVC_IMG_STATE**

A new command is added to support MPEG transport stream encapsulation of AVC bitstream in Encoder mode. This command should be used only when MPEG transport stream is needed.

**MFX_AVC_DIRECTMODE_STATE >[BDW]**

**MFX_AVC_SLICE_STATE**

**MFX_AVC_REF_IDX_STATE**

**MFX_AVC_WEIGHTOFFSET_STATE**

## AVC Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

### MFD_AVC_DPB_STATE

NOTE modified from DXVA2 – The values in RefFrameList and UsedForReference_Flag are the primary means by which the H/W can determine whether the corresponding entries in RefFrameList, POCList, LTSTFrameNumList, and Non-ExistingFrame_Flag should be considered valid for use in the decoding process of the current picture or not. When RefFrameList[i] is marked to be invalid, the values of POCList[i][0], POCList[i][1], LTSTFrameNumList[i], UsedForReference_Flag[i], and Non-ExistingFrame_Flag[i] must all be equal to 0. When UsedForReference_Flag[i] = 0, the value of RefFrameList[i] must be marked invalid.

### MFD_AVC_SLICEADDR

### MFD_AVC_BSD_OBJECT

### Inline Data Description for MFD_AVC_BSD_Object

### MFD_AVC_PICID_STATE

NOTE 1: In AVC short format, PictureIDList has one-to-one corresponding to LongTermFrame_Flag list, Non-ExistingFrame_flag list, UsedForReference_Flag list, FrameNumList list in MFD_AVC_DPB_STATE.

NOTE 2: PictureIDList is only used to identify reference picture across frames. Hardware will convert the picture in the RefFrameList to PictureID before writing out DMV data and convert back to RefFrameList Index after reading out DMV data. The reference pictures and their orders in the RefFrameList can be changed across frames.

## Session Decoder StreamOut Data Structure

When StreamOut is enabled, per MB intermediated decoded data (MVs, mb_type, MB qp, etc.) are sent to the memory in a fixed record format (and of fixed size). The per-MB records must be written in a strict raster order and with no gap (i.e. every MB regardless of its mb_type and slice type, must have an entry in the StreamOut buffer). Therefore, the consumer of the StreamOut data can offset into the StreamOut Buffer (**StreamOut Data Destination Base Address**) using individual MB addresses.

A StreamOut Data record format is detailed as follows:

| DWord | Bit | Description |
|---|---|---|
| 0 | 23 | **Reserved MBZ** |
| | 22-20 | **EdgeFilterFlag** (AVC) / **OverlapSmoothFilter** (VC1) |
| | 19:17 | **CodedPatternDC** (for AVC only, **111b** for others)<br>The field indicates whether DC coefficients are sent.<br>1 bit each for Y, U and V. |
| | 16 | **Reserved MBZ** |
| | 15 | **Transform8x8Flag**<br>When it is set to 0, the current MB uses 4x4 transform. When it is set to 1, the current MB uses 8x8 |

| DWord | Bit | Description |
|---|---|---|
| | | transform. The transform_szie_8x8_flag syntax element, if present in the output bitstream, is the same as this field. However, whether transform_szie_8x8_flag is present or not in the output bitstream depends on several conditions:<br>This field is only allowed to be set to 1 for two conditions:<br>It must be 1 if **IntraMbFlag** = INTRA and **IntraMbMode** = INTRA_8x8<br>It may be 1 if **IntraMbFlag** = INTER and there is no sub partition size less than 8x8<br>Otherwise, this field must be set to 0.<br>0: 4x4 integer transform<br>1: 8x8 integer transform |
| | 14 | **MbFieldFlag**<br>This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode.<br>This field is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode.<br>Same as the mb_field_decoding_flag syntax element in AVC spec.<br>0 = Frame macroblock<br>1 = Field macroblock |
| | 13 | **IntraMbFlag**<br>This field specifies whether the current macroblock is an Intra (I) macroblock.<br>I_PCM is considered as Intra MB.<br>For I-picture MB (IntraPicFlag =1), this field must be set to 1.<br>This flag must be set in consistent with the interpretation of MbType (inter or intra modes).<br>0: INTER (inter macroblock)<br>**1: INTRA (intra macroblock)** |
| | 12:8 | **MbType5Bits**<br>This field is encoded to match with the best macroblock mode determined as described in the next section. It follows AVC encoding for inter and intra macroblocks. |
| | 7 | **MbPolarity**<br>FieldMB Polarity - vctrl_vld_top_field AVC |
| | 6 | **Reserved MBZ** |
| | 5:4 | **IntraMbMode**<br>This field is provided to carry information partially overlapped with MbType.<br>This field is only valid if **IntraMbFlag** = INTRA, otherwise, it is ignored by hardware. |
| | 3 | **Reserved MBZ** |
| | 2 | **MbSkipFlag**<br>Reserved MBZ (DXVA Encoder). HW (VDSunit) doesn't have skip MB info.<br>It sets to 1 if any of the sub-blocks is inter, uses predicted MVs, and skips sending MVs explicitly in the code stream. Currently H/W can provide this flag and is defaulted to 0 always. |

| DWord | Bit | Description |
|---|---|---|
| | 1:0 | **InterMbMode**<br>This field is provided to carry redundant information as that in MbType. It also carries additional information such as skip.<br>This field is only valid if **IntraMbFlag** =INTER, otherwise, it is ignored by hardware. |
| 1 | 31:16 | **MbYCnt (Vertical Origin).**<br>This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks.<br>Format = U8 in unit of macroblock. |
| | 15:0 | **MbXCnt (Horizontal Origin).**<br>This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks.<br>Format = U8 in unit of macroblock. |
| 2 | 31 | **Conceal MB Flag.**<br>This field specifies if the current MB is a conceal MB, use in AVC/VC1/MPEG2 mode. |
| | 30 | **Last MB of the Slice Flag.**<br>This field indicate the current MB is a last MB of the slice. Use in AVC/VC1/MPEG2 mode. |
| | 29:24 | **Reserved** |
| | 23:20 | **CbpAcV**<br>0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero)<br>1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding). |
| | 19:16 | **CbpAcU**<br>0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero)<br>1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding). |
| | 15:0 | **CbpAcY**<br>0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero)<br>1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).<br>Bit15=Y0Sub0, Bit0=Y3Sub3 |
| 3 | 31:28<br>AVC | **Skip8x8Pattern** (AVC)<br>This field indicates whether each of the four 8x8 sub macroblocks is using the predicted MVs and will not be explicitly coded in the bitstream (the sub macroblock will be coded as direct mode). It contains four 1-bit subfields, corresponding to the 4 sub macroblocks in sequential order. The whole macroblock may be actually coded as B_Direct_16x16 or B_Skip, according to the macroblock type conversion rules described in a later sub section.<br>This field is only valid for a B slice. It is ignored by hardware for a P slice. Hardware also ignores this |

| DWord | Bit | Description |
|---|---|---|
| | | field for an intra macroblock.<br>0 in a bit – Corresponding MVs are sent in the bitstream<br>1 in a bit – Corresponding MVs are not sent in the bitstream |
| | 27:25 | **Reserved** |
| | 24:16 | **NzCoefCountMB** – all coded coefficients input including AC/DC blocks in current MB.<br>Range 0 to 384 (9 bits) |
| | 15:8 | **[BDW] MbClock16** – MB compute clocks in 16-clock unit. |
| | 7 | **mbz** (AVC) **/ QScaleType** (MPEG2) |
| | 6:0 | **QpPrimeY** (AVC) **/ QScaleCode** (MPEG2)<br>The luma quantization index. This is the per-MB QP value specified for the current MB. |
| 4 to 6 | 31:0 Each | For intra macroblocks, definition of these fields are specified in 1<br>For inter macroblocks, definition of these fields are specified in 2 |
| 7 | 31:24 | **Reserved** |
| | 23:20 | **MvFieldSelect** (Ref polarity top or bottom bits) for VC1 and MPEG2<br>vcp_vds_mvdataR[162:159] VC1<br>vmd_vds_mt_vert_fld_selR[3:0] MPEG2 |
| | 19:12 | **Reserved** |
| | 11:10 | **SubBlockCodeType V**<br>(If 8x8, 8x4, 4x8, 4x4 type) |
| | 9:8 | **SubBlockCodeType U**<br>(specifies 8x8, 8x4, 4x8, 4x4 type) VC1 |
| | 7:6 | **SubBlockCodeType Y3**<br>(specifies 8x8, 8x4, 4x8, 4x4 type) VC1 |
| | 5:4 | **SubBlockCodeType Y2**<br>(specifies 8x8, 8x4, 4x8, 4x4 type) VC1 |
| | 3:2 | **SubBlockCodeType Y1**<br>(specifies 8x8, 8x4, 4x8, 4x4 type) VC1 |
| | 1:0 | **SubBlockCodeType Y0**<br>(specifies 8x8, 8x4, 4x8, 4x4 type) VC1 |
| **Inter Cases:** | | |
| 8 | 31:16 | **MvFwd[0].y** – y-component of the forward motion vector of the 1st 8x8 or 1st 4x4 subblock |
| | 15:0 | **MvFwd[0].x** – x-component of the forward motion vector of the 1st 8x8 or 1st 4x4 subblock |
| 9 | 31:0 | **MvBck[0]** – the backward motion vector of the 1st 8x8 or 1st 4x4 subblock |
| 10 | 31:0 | **MvFwd[1]** – the forward motion vector of the 2nd 8x8 or 4th 4x4 subblock |

| DWord | Bit | Description |
|---|---|---|
| 11 | 31:0 | **MvBck[1]** – the backward motion vector of the 2nd 8x8 or 4th 4x4 subblock |
| 12 | 31:0 | **MvFwd[2]** – the forward motion vector of the 3rd 8x8 or 8th 4x4 subblock |
| 13 | 31:0 | **MvBck[2]** – the backward motion vector of the 3rd 8x8 or 8th 4x4 subblock |
| 14 | 31:0 | **MvFwd[3]** – the forward motion vector of the 4th 8x8 or 12th 4x4 subblock |
| 15 | 31:0 | **MvBck[3]** – the backward motion vector of the 4th 8x8 or 12th 4x4 subblock |
| **Intra Cases:** | | |
| 8 to 15 | 31:0 | **Reserved MBZ** |

The inline data content of Dwords 4 to 6 is defined either for intra prediction or for inter prediction, but not both.

### Inline data subfields for an Intra Macroblock

| DWord | Bit | Description |
|---|---|---|
| 4 | 31:16 | **LumaIndraPredModes[1]**<br>Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.<br>AVC: See the bit assignment table later in this section.<br>VC1: MBZ.<br>MPEG2: MBZ. |
| | 15:0 | **LumaIndraPredModes[0]**<br>Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB.<br>4-bit per 4x4 sub-block (Transform8x8Flag=0, Mbtype=0 and intraMbFlag=1) or 8x8 block (Transform8x8Flag=1, Mbtype=0, MbFlag=1), since there are 9 intra modes.<br>4-bit for intra16x16 MB (Transform8x8Flag=0, Mbtype=1 to 24 and intraMbFlag=1), but only the LSBit[1:0] is valid, since there are only 4 intra modes.<br>AVC: See the bit assignment table later in this section.<br>VC1: MBZ.<br>MPEG2: MBZ. |
| 5<br>AVC INTRA | 31:16 | **LumaIndraPredModes[3]**<br>Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.<br>AVC: See the bit assignment table later in this section.<br>VC1: MBZ.<br>MPEG2: MBZ. |
| | 15:0 | **LumaIndraPredModes[2]** |

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.<br>AVC: See the bit assignment later in this section.<br>VC1: MBZ.<br>MPEG2: MBZ.</td></tr>
<tr><td rowspan="2">6</td><td>31:8</td><td>Reserved (Reserved for encoder turbo mode IntraResidueDataSize, when this is not 0, optional residue data are provided to the PAK; Reserved for decoder)</td></tr>
<tr><td>7:0</td><td>MbIntraStruct<br>The IntraPredAvailFlags[4:0] have already included the effect of the constrained_intra_pred_flag. See the diagram later for the definition of neighbors position around the current MB or MB pair (in MBAFF mode).<br>1 – IntraPredAvailFlagX, indicates the values of samples of neighbor X can be used in intra prediction for the current MB.<br>0 – IntraPredAvailFlagX, indicates the values of samples of neighbor X is not available for intra prediction of the current MB.<br>IntraPredAvailFlag-A and -E can only be different from each other when constrained_intra_pred_flag is equal to 1 and mb_field_decoding_flag is equal to 1 and the value of the mb_field_decoding_flag for the macroblock pair to the left of the current macroblock is equal to 0 (which can only occur when MbaffFrameFlag is equal to 1).<br>IntraPredAvailFlag-F is used only if<br>○ it is in MBAFF mode, i.e. MbaffFrameFlag = 1<br>○ the current macroblock is of frame type, i.e. MbFieldFag = 0, and<br>○ the current macroblock type is Intra8x8,<br>that is, IntraMbFlag = INTRA, IntraMbMode = INTRA_8x8, and Transform8x8Flag = 1.<br>In any other cases IntraPredAvailFlag-A shall be used instead.
<table>
<tr><th>Bits</th><th>IntraPredAvailFlags[4:0] Definition</th></tr>
<tr><td>7</td><td>IntraPredAvailFlagF – F (Left 8th row (-1,7) neighbor)</td></tr>
<tr><td>6</td><td>IntraPredAvailFlagA – A<br>(Left neighbor top half)</td></tr>
<tr><td>5</td><td>IntraPredAvailFlagE – E<br>(Left neighbor bottom half)</td></tr>
<tr><td>4</td><td>IntraPredAvailFlagB – B<br>(Top neighbor)</td></tr>
<tr><td>3</td><td>IntraPredAvailFlagC – C<br>(Top right neighbor)</td></tr>
<tr><td>2</td><td>IntraPredAvailFlagD – D<br>(Top left corner neighbor)</td></tr>
<tr><td>1:0</td><td>ChromaIntraPredMode –<br>2 bits to specify 1 of 4 chroma intra prediction mode, see the table in later section.</td></tr>
</table></td></tr>
</table>

## Inline data subfields for an Inter Macroblock

| DWord | Bit | Description |
|---|---|---|
| 4 | 31:24 | Reserved: MBZ (DXVA Decoder) |
| | 23:16 | Reserved: MBZ (DXVA Decoder) |
| | 15:8 | **SubMbPredModes**[bit 7:0] (Sub Macroblock Prediction Mode)<br>This field describes the prediction mode of the sub macroblocks (four 8x8 blocks). It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order.<br>This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA)<br>This field is derived from MbType for a non-BP_8x8 inter macroblock, and carries redundant information as MbType)<br>Bits [1:0]: SubMbPredMode[0] – for 8x8 Block 0<br>Bits [3:2]: SubMbPredMode[1] – for 8x8 Block 1<br>Bits [5:4]: SubMbPredMode[2] – for 8x8 Block 2<br>Bits [7:6]: SubMbPredMode[3] – for 8x8 Block 3<br>Blocks of the MB is numbered as follows :<br>0 1<br>2 3<br>Each 2-bit value [1:0] is defined as :<br>00 – Pred_L0<br>01 – Pred_L1<br>10 – BiPred<br>**For VC1:**<br>Bits [1:0]: "00"= Y0 Forward only, "01"= Y0 Backward only, "10"= Y0 Bi direction<br>Bits [3:2]: SubMbPredMode[1] – for 8x8 Block 1<br>Bits [5:4]: SubMbPredMode[2] – for 8x8 Block 2<br>Bits [7:6]: SubMbPredMode[3] – for 8x8 Block 3 |
| | 7:0 | **SubMbShape**[bit 7:0] (Sub Macroblock Shape)<br>This field describes the sub-block partitioning of each sub macroblocks (four 8x8 blocks). It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order.<br>This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA)<br>This field is forced to 0 for a non-BP_8x8 inter macroblock, and effectively carries redundant |

| DWord | Bit | Description |
|---|---|---|
| | | information as MbType).<br>Bits [1:0]: SubMbShape[0] – for 8x8 Block 0<br>Bits [3:2]: SubMbShape[1] – for 8x8 Block 1<br>Bits [5:4]: SubMbShape[2] – for 8x8 Block 2<br>Bits [7:6]: SubMbShape[3] – for 8x8 Block 3<br>Blocks of the MB is numbered as follows :<br>0 1<br>2 3<br>Each 2-bit value [1:0] is defined as :<br>00 – SubMbPartWidth=8, SubMbPartHeight=8<br>01 – SubMbPartWidth=8, SubMbPartHeight=4<br>10 – SubMbPartWidth=4, SubMbPartHeight=8<br>11 – SubMbPartWidth=4, SubMbPartHeight=4<br>For VC-1, This field indicates the transformation types used for luma components, 2 bits for each 8x8. |
| 5 | 31:24 | **Frame Store ID L0[3]**<br>Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table.<br>Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID)<br>1: indicate it is in Frame store ID format.<br>0: indicate it is in Reference Index format.<br>Bit 6:5: reserved MBZ<br>Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) |
| | 23:16 | **Frame Store ID L0[2]**<br>Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table.<br>Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID)<br>1: indicate it is in Frame store ID format.<br>0: indicate it is in Reference Index format.<br>Bit 6:5: reserved MBZ |

| DWord | Bit | Description |
|---|---|---|
| | | Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) |
| | 15:8 | **Frame Store ID L0[1]**<br>Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table.<br>Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID)<br>1: indicate it is in Frame store ID format.<br>0: indicate it is in Reference Index format.<br>Bit 6:5: reserved MBZ<br>Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation). |
| | 7:0 | **Frame Store ID L0[0]**<br>Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table.<br>Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID)<br>1: indicate it is in Frame store ID format.<br>0: indicate it is in Reference Index format.<br>Bit 6:5: reserved MBZ<br>Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) |
| 6 | 31:24 | **Frame Store ID L1[3]**<br>Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table.<br>Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID)<br>1: indicate it is in Frame store ID format.<br>0: indicate it is in Reference Index format.<br>Bit 6:5: reserved MBZ<br>Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) |
| | 23:16 | **Frame Store ID L1[2]**<br>Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later |

| DWord | Bit | Description |
|---|---|---|
| | | section. This field specifies the frame Store ID into the Reference Picture List0 Table.<br>Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID)<br>1: indicate it is in Frame store ID format.<br>0: indicate it is in Reference Index format.<br>Bit 6:5: reserved MBZ<br>Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) |
| | 15:8 | **Frame Store ID L1[1]**<br>Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table.<br>Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID)<br>1: indicate it is in Frame store ID format.<br>0: indicate it is in Reference Index format.<br>Bit 6:5: reserved MBZ<br>Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) |
| | 7:0 | **Frame Store ID L1[0]**<br>Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table.<br>Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID)<br>1: indicate it is in Frame store ID format.<br>0: indicate it is in Reference Index format.<br>Bit 6:5: reserved MBZ<br>Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation) |

# AVC Encoder PAK Commands

Each PAK Commands is composed of a command op-code DW and one or more command data DWs (inline data). The size of each command is specified as part of the op-code DW. Most of the commands have fixed size, except some are allowed to be of variable length.

There is an inherent order of executing MFC PAK commands that driver must follow.

**MFC_AVC_PAK_OBJECT**

## Indirect Data Description

For each macroblock, an ENC-PAK data set consists of two types of data blocks: indirect **MV data block** and **inline MB information**.

The indirect MV data block may be in two modes: **unpackedmode** and **packed-size mode**.

## Unpacked Motion Vector Data Block

In the **unpacked** mode, motion vectors are expanded (or duplicated) to either bidirectional 8x8 8MV major partition format, or bidirectional 4x4 32MV format. Thus either 32 bytes or 128 bytes is assigned to each MB.

Motion Vector block contains motion vectors in an intermediate format that is partially expanded according to the sub- macroblock size. During the expansion, a place that does not contain a motion vector is filled by replicating the relevant motion vector according to the following motion vector replication rules. If the relevant motion vector doesn't exist (for the given L0 or L1), it is zero filled.

Motion Vector Replication Rules:

- Rule #1
    - #1.1: For L0 MV, for any sub-macroblock or sub-partition where there is at least one motion vector
        - If L0 inter prediction exists, the corresponding L0 MV is used
        - Else it must be zero
    - #1.2: For L1 MV, for any sub-macroblock or sub-partition where there is at least one motion vector
        - If L1 inter prediction exists, the corresponding L1 MV is used
        - Else it must be zero
- For a macroblock with a 16x16, 16x8 or 8x16 sub-macroblock, MvSize = 8. The eight MV fields follow Rule #1.
    - The 16x16 is broken down into 4 8x8 sub-macroblocks. The 16x16 MVs (after rule #1) are replicated into all 8x8 blocks.

- For an 8x16 partition, each 8x16 is broken down into 2 8x8 stacking vertically. The 8x16 MVs (after rule #1) are replicated into both 8x8 blocks.
- For a 16x8 partition, each 16x8 is broken down into 2 8x8 stacking horizontally. The 16x8 MVs (after rule #1) are replicated into both 8x8 blocks.

- For macroblock with sub-macroblock of 8x8 without minor partition (SubMbShape[0...3] = 0), MvSize = 8, (e.g. mb_type equal to P_8x8, P_8x8ref0, or B_8x8)
  - There is no motion vector replication
- For macroblock with sub-macroblock of 8x8 with at least one minor partition (if any SubMbShape[i] != 0), MvSize = 32, (e.g. mb_type equal to P_8x8, P_8x8ref0, or B_8x8)
  - For an 8x8 sub-partition, the 8x8 MVs (after rule #1) is replicated into all the four 4x4 blocks.
  - For an 4x8 sub-partition within an 8x8 partition, each 4x8 is broken down into 2 4x4 stacking vertically. The 4x8 MVs (after rule #1) are replicated into both 4x4 blocks.
  - For an 8x4 sub-partition within an 8x8 partition, each 8x4 is broken down into 2 4x4 stacking horizontally. The 8x4 MVs (after rule #1) are replicated into both 4x4 blocks.
  - For a 4x4 sub-partition within an 8x8 partition, each 4x4 has its own MVs (after rule #1).

### Motion Vector block and MvSize

| DWord | Bit | MvSize | |
|---|---|---|---|
| | | 8 | 32 |
| W1.0 | 31:16 | **MV_Y0_L0.y** | **MV_Y0_0_L0.y** |
| | 15:0 | **MV_Y0_L0.x** | **MV_Y0_0_L0.x** |
| W1.1 | 31:16 | **MV_Y0_L1.y** | **MV_Y0_0_L1.y** |
| | 15:0 | **MV_Y0_L1.x** | **MV_Y0_0_L1.x** |
| W1.2 | 31:0 | **MV_Y1_L0** | **MV_Y0_1_L0** |
| W1.3 | 31:0 | **MV_Y1_L1** | **MV_Y0_1_L1** |
| W1.4 | 31:0 | **MV_Y2_L0** | **MV_Y0_2_L1** |
| W1.5 | 31:0 | **MV_Y2_L1** | **MV_Y0_2_L0** |
| W1.6 | 31:0 | **MV_Y3_L0** | **MV_Y0_3_L0** |
| W1.7 | 31:0 | **MV_Y3_L1** | **MV_Y0_3_L1** |

| DWord | Bit | MvSize | |
|---|---|---|---|
| | | 8 | 32 |
| W2.0 | 31:0 | n/a | MV_Y1_0_L1 |
| W2.1 | 31:0 | n/a | MV_Y1_0_L0 |
| W2.2 | 31:0 | n/a | MV_Y1_1_L1 |
| W2.3 | 31:0 | n/a | MV_Y1_1_L0 |
| W2.4 | 31:0 | n/a | MV_Y1_2_L1 |
| W2.5 | 31:0 | n/a | MV_Y1_2_L0 |
| W2.6 | 31:0 | n/a | MV_Y1_3_L0 |
| W2.7 | 31:0 | n/a | MV_Y1_3_L1 |
| W3.0 | 31:0 | n/a | MV_Y2_0_L1 |
| W3.1 | 31:0 | n/a | MV_Y2_0_L0 |
| W3.2 | 31:0 | n/a | MV_Y2_1_L1 |
| W3.3 | 31:0 | n/a | MV_Y2_1_L0 |
| W3.4 | 31:0 | n/a | MV_Y2_2_L1 |
| W3.5 | 31:0 | n/a | MV_Y2_2_L0 |
| W3.6 | 31:0 | n/a | MV_Y2_3_L0 |
| W3.7 | 31:0 | n/a | MV_Y2_3_L1 |
| W4.0 | 31:0 | n/a | MV_Y3_0_L1 |
| W4.1 | 31:0 | n/a | MV_Y3_0_L0 |
| W4.2 | 31:0 | n/a | MV_Y3_1_L1 |
| W4.3 | 31:0 | n/a | MV_Y3_1_L0 |
| W4.4 | 31:0 | n/a | MV_Y3_2_L1 |

| DWord | Bit | MvSize | |
|---|---|---|---|
| | | 8 | 32 |
| W4.5 | 31:0 | **n/a** | **MV_Y3_2_L0** |
| W4.6 | 31:0 | **n/a** | **MV_Y3_3_L0** |
| W4.7 | 31:0 | **n/a** | **MV_Y3_3_L1** |

The motion vector(s) for a given sub-macroblock or a sub-partition are uniquely placed in the output message as shown by the non-duplicate fields in *Unpacked Motion Vector Data Block* and *Unpacked Motion Vector Data Block*.

MV_Yx_L0 and MV_Yx_L1 may be present individually or both. If one is not present, the corresponding field must be zero. Subsequently, the duplicated fields will be zero as well.

### Motion Vector duplication by sub-macroblocks for a 16x16 macroblock, whereas the 8x8 column is for 4x(8x8) partition without minor shape

| DWord | Bit | 16x16 | 16x8 | 8x16 | 8x8 |
|---|---|---|---|---|---|
| W1.0 | 31:16 | **MV_Y0_L1 (A)** | **MV_Y0_L1 (A)** | **MV_Y0_L1** | **MV_Y0_L1** |
| | 15:0 | **MV_Y0_L0 (A)** | **MV_Y0_L0 (A)** | **MV_Y0_L0** | **MV_Y0_L0** |
| W1.1 | 31:16 | **Duplicate (A)** | **Duplicate (A)** | **MV_Y1_L1** | **MV_Y1_L1** |
| | 15:0 | **Duplicate (A)** | **Duplicate (A)** | **MV_Y1_L0** | **MV_Y1_L0** |
| W1.2 | 31:16 | **Duplicate (A)** | **MV_Y2_L1 (B)** | **Duplicate (A)** | **MV_Y2_L1** |
| | 15:0 | **Duplicate (A)** | **MV_Y2_L0 (B)** | **Duplicate (A)** | **MV_Y2_L0** |
| W1.3 | 31:16 | **Duplicate (A)** | **Duplicate (B)** | **Duplicate (B)** | **MV_Y3_L1** |
| | 15:0 | **Duplicate (A)** | **Duplicate (B)** | **Duplicate (B)** | **MV_Y3_L0** |

### Motion Vector duplication by sub-partitions for the first 8x8 sub-macroblock Y0 if any Y0-Y3 contains minor shape (Y1_ to Y3_ have the same format in W2 to W4)

| DWord | Bit | 8x8 | 8x4 | 4x8 | 4x4 |
|---|---|---|---|---|---|
| W1.0 | 31:16 | **MV_Y0_L1** | **MV_Y0_0_L1 (A)** | **MV_Y0_0_L1 (A)** | **MV_Y0_0_L1** |
| | 15:0 | **MV_Y0_L0** | **MV_Y0_0_L0 (A)** | **MV_Y0_0_L0 (A)** | **MV_Y0_0_L0** |
| W1.1 | 31:16 | **Duplicate (A)** | **Duplicate (A)** | **MV_Y0_1_L1 (B)** | **MV_Y0_1_L1** |
| | 15:0 | **Duplicate (A)** | **Duplicate (A)** | **MV_Y0_1_L0 (B)** | **MV_Y0_1_L0** |
| W1.2 | 31:16 | **Duplicate (A)** | **MV_Y0_2_L1 (B)** | **Duplicate (A)** | **MV_Y0_2_L1** |
| | 15:0 | **Duplicate (A)** | **MV_Y0_2_L0 (B)** | **Duplicate (A)** | **MV_Y0_2_L0** |
| W1.3 | 31:16 | **Duplicate (A)** | **Duplicate (B)** | **Duplicate (B)** | **MV_Y0_3_L0** |
| | 15:0 | **Duplicate (A)** | **Duplicate (B)** | **Duplicate (B)** | **MV_Y0_3_L1** |

## Packed-Size Motion Vector Data Block

In the packed case, no redundant motion vectors are sent. So the number of motion vectors sent, as specified by **MvQuantity** is the same as the motion vectors that will be packed (**MvPacked**).

The following tables are for information only. Fields like MvQuantity and MvPacked are not required interface fields.

| MbSkipFlag | MbType | Description | Mv Quantity | MvSize | (Minimal MvSize) |
|---|---|---|---|---|---|
| 1 | 1 | P_Skip_16x16 | 0 | 8 | 1 |
| 0 | 1 | BP_L0_16x16 | 1 | 8 | 1 |
| 0 | 2 | B_L1_16x16 | 1 | 8 | 1 |
| 0 | 3 | B_Bi_16x16 | 2 | 8 | 2 |
| 0 | 4 | BP_L0_L0_16x8 | 2 | 8 | 4 |
| 0 | 5 | BP_L0_L0_8x16 | 2 | 8 | 4 |
| 0 | 6 | B_L1_L1_16x8 | 2 | 8 | 8 |
| 0 | 7 | B_L1_L1_8x16 | 2 | 8 | 8 |
| 0 | 8 | B_L0_L1_16x8 | 2 | 8 | 8 |
| 0 | 9 | B_L0_L1_8x16 | 2 | 8 | 8 |
| 0 | 0Ah | B_L1_L0_16x8 | 2 | 8 | 8 |
| 0 | 0Bh | B_L1_L0_8x16 | 2 | 8 | 8 |

| MbSkipFlag | MbType | Description | Mv Quantity | MvSize | (Minimal MvSize) |
|---|---|---|---|---|---|
| 0 | 0Ch | B_L0_Bi_16x8 | 3 | 8 | 8 |
| 0 | 0Dh | B_L0_Bi_8x16 | 3 | 8 | 8 |
| 0 | 0Eh | B_L1_Bi_16x8 | 3 | 8 | 8 |
| 0 | 0Fh | B_L1_Bi_8x16 | 3 | 8 | 8 |
| 0 | 10h | B_Bi_L0_16x8 | 3 | 8 | 8 |
| 0 | 11h | B_Bi_L0_8x16 | 3 | 8 | 8 |
| 0 | 12h | B_Bi_L1_16x8 | 3 | 8 | 8 |
| 0 | 13h | B_Bi_L1_8x16 | 3 | 8 | 8 |
| 0 | 14h | B_Bi_Bi_16x8 | 4 | 8 | 8 |
| 0 | 15h | B_Bi_Bi_8x16 | 4 | 8 | 8 |
| 0 | 16h | **BP**_8x8 | [3]4 | 8 or 32 | 8 or 32 |

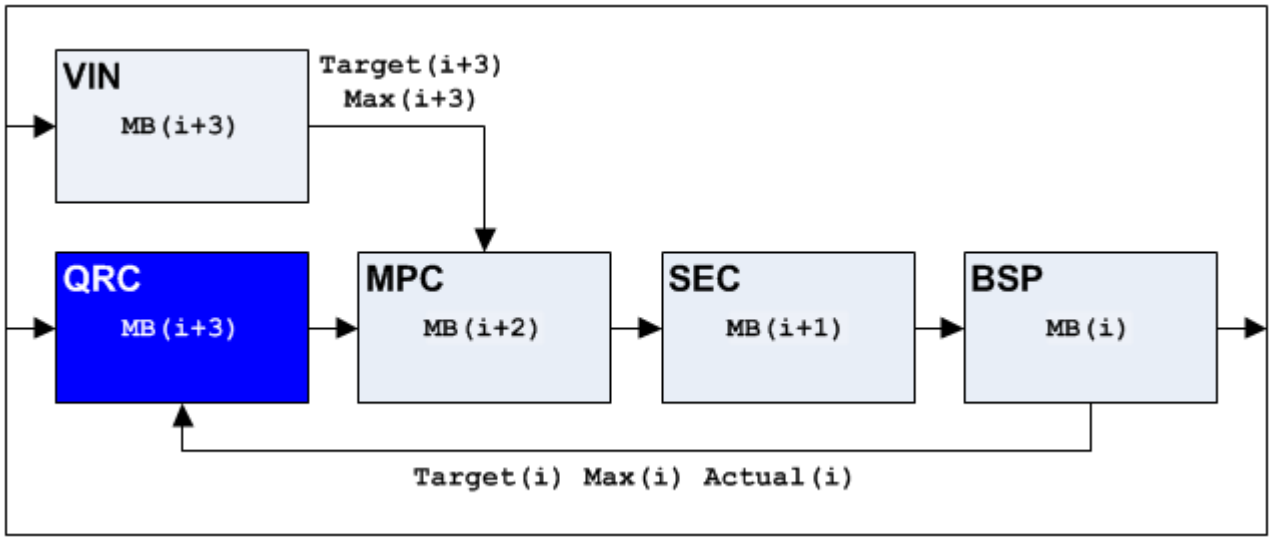
When MbType = 22, BP_8x8, take the sum of four individual 8x8 subblocks

| Direct8x8Pattern | SubMb Shape | SubMb PredMode | Description | Mv Quantity | Mv Size | (Min MvSize) |
|---|---|---|---|---|---|---|
| OR | OR | OR | | ADD | ADD | ADD |
| 1 | 0 | 0 | P_Skip_8x8<br>B_Direct_L0_8x8<br>(B-Skip_ L0_8x8) | 0 | 2 | 1 |
| 1 | 0 | 1 | B_Direct_L1_8x8<br>(B-Skip_ L1_8x8) | 0 | 2 | 1 |
| 1 | 0 | 2 | B_Direct_Bi_8x8<br>(B-Skip_ Bi_8x8) | 0 | 2 | 2 |
| 1 | 3 | 0 | P_Skip_4x4<br>B_Direct_L0_4x4<br>(B-Skip_ L0_4x4) | 0 | 8 | 4 |
| 1 | 3 | 1 | B_Direct_L1_4x4<br>(B-Skip_ L1_4x4) | 0 | 8 | 4 |
| 1 | 3 | 2 | B_Direct_Bi_4x4<br>(B-Skip_ Bi_4x4) | 0 | 8 | 8 |
| 0 | 0 | 0 | BP_L0_8x8 | 1 | 2 | 1 |

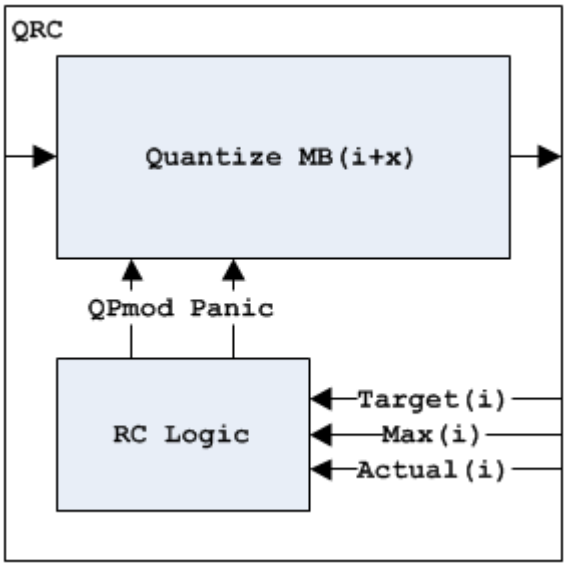
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | B_L1_8x8 | 1 | 2 | 1 |
| 0 | 0 | 2 | B_BI_8x8 | 2 | 2 | 2 |
| 0 | 1 | 0 | BP_L0_8x4 | 2 | 8 | 4 |
| 0 | 1 | 1 | B_L1_8x4 | 2 | 8 | 4 |
| 0 | 1 | 2 | B_BI_8x4 | 4 | 8 | 8 |
| 0 | 2 | 0 | BP_L0_4x8 | 2 | 8 | 4 |
| 0 | 2 | 1 | B_L1_4x8 | 2 | 8 | 4 |
| 0 | 2 | 2 | B_BI_4x8 | 4 | 8 | 8 |
| 0 | 3 | 0 | BP_L0_4x4 | 4 | 8 | 4 |
| 0 | 3 | 1 | B_L1_4x4 | 4 | 8 | 4 |
| 0 | 3 | 2 | B_BI_4x4 | 8 | 8 | 8 |

## Macroblock Level Rate Control

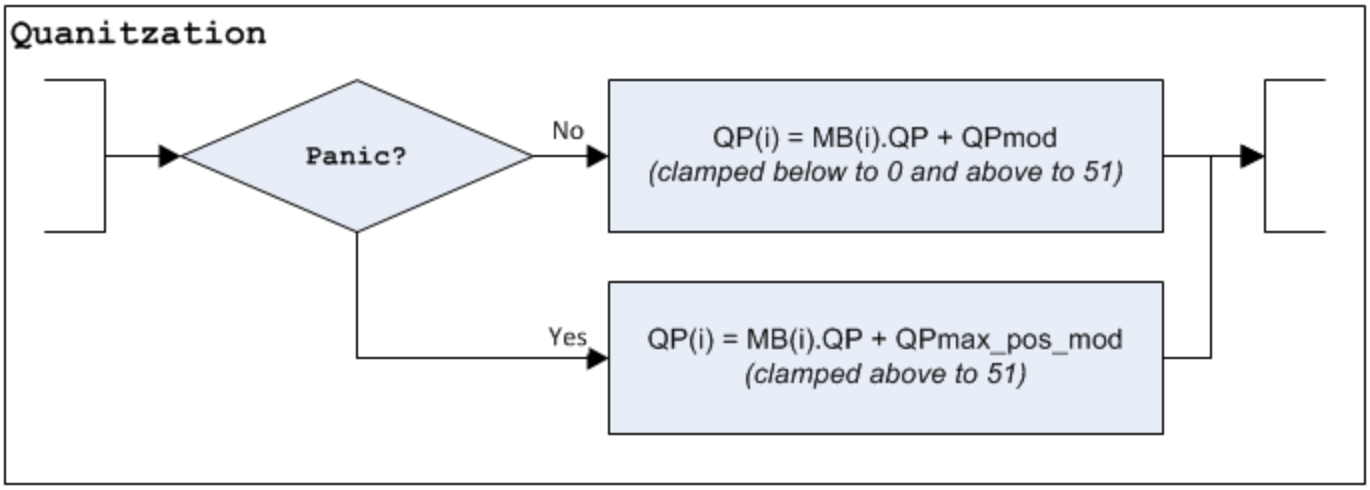
The QRC (Quantization Rate Control) unit receives data from BSP (Bit Serial Packer) and VIN (Video In) and generates adjustments to QP values across macroblocks.



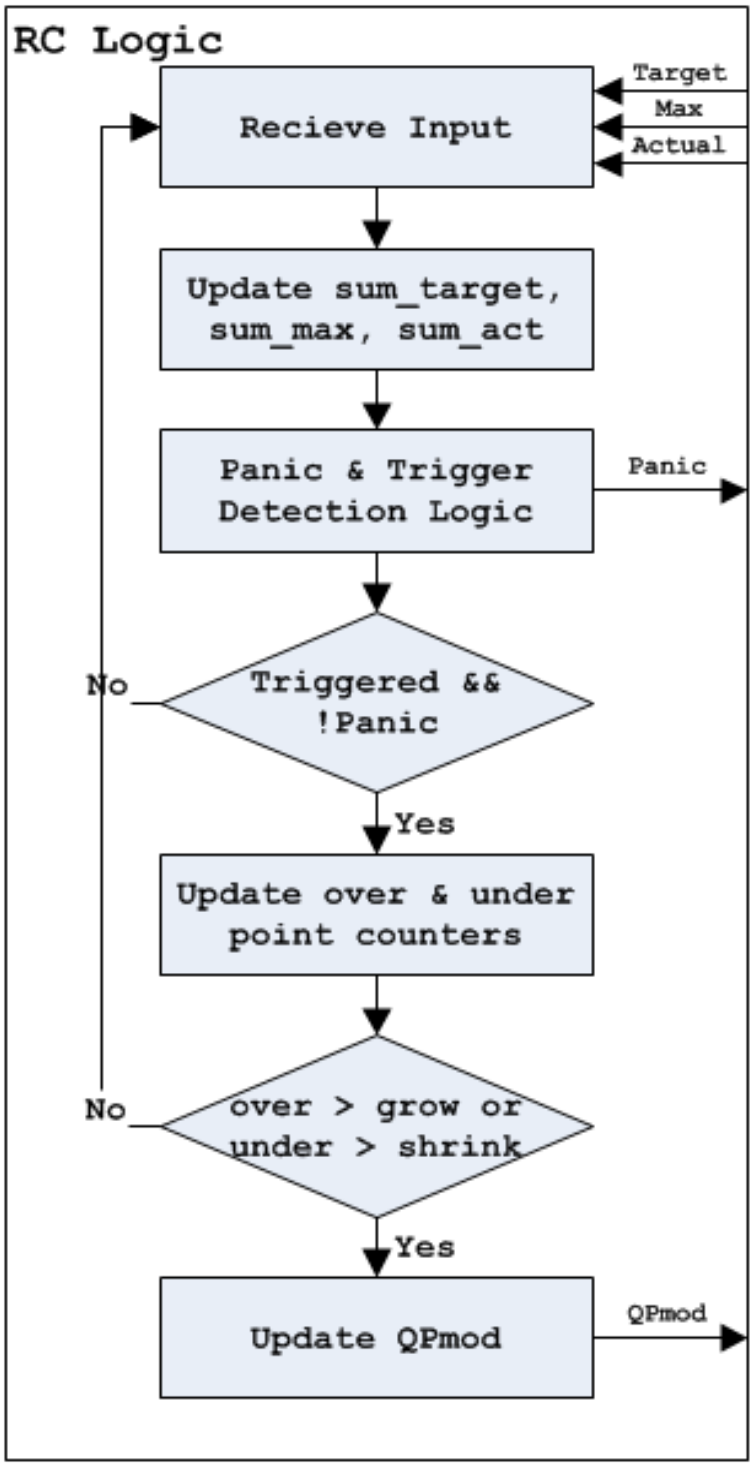
QRC can be logically partitioned into two units as shown below.




Macroblock level rate control is handled by the RC logic and the quantization logic.



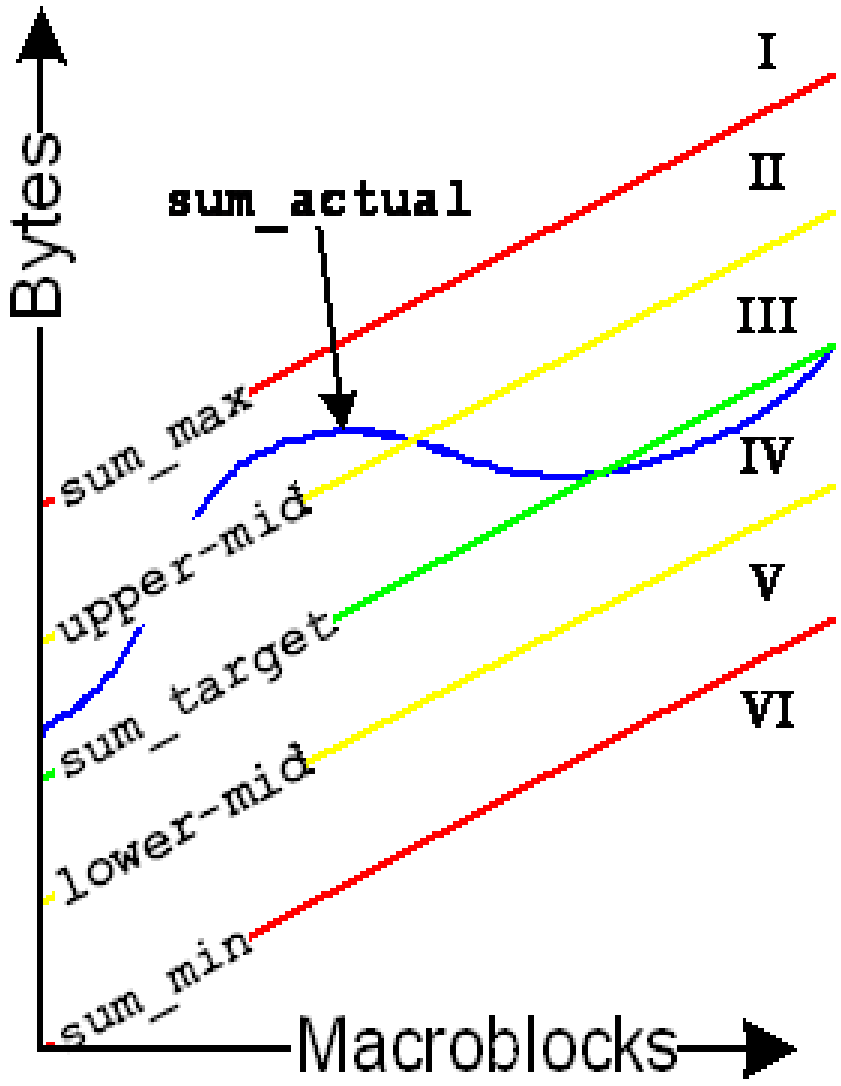
The signals QPmod and panic are generated by the RC logic based on data feedback from BSP. A flowchart of the RC logic is given below.



## Theory of Operation Overview

BSP will generate a byte estimate for each macroblock packed. Additionally, the user will specify a target and max size per macroblock. The running sum of these signals (actual, target, max) creates “curves” which are used to identify when QP adjustments are necessary (see figure below). Three more curves are symmetrically generated by QRC (upper_midpt, lower_midpt, sum_min) from target and max. The values of target and max are specified by the user will dictate the shape of these curves.

The difference between sum_actual and sum_target (called ‘bytediff’) identifies the margin of error between the target and actual sizes. The difference between the current bytediff and the previously calculated bytediff represents the rate of change in this margin over time. The sign of this rate is used to identify if the correction is trending in the appropriate direction (towards bytediff = 0).



## QPmod

Each macroblock will have a requested QP (which could vary across macroblocks or remain constant). QPmod is to be added to the QP requested. QPmod will be positive when the target was under-predicted and negative when the target is over-predicted.

QPmod is incremented or decremented when internal counters (called 'over' and 'under') reach tripping points (called 'grow' and 'shrink'). For each MB processed and based on which region (1-6) sum_actual falls in, various amounts of points are added to either counters. If over exceeds grow, QPmod is incremented whereas if under exceeds shrink, QPmod is decremented.

To dampen the effect of repeated changes in the same direction, an increase in resistance for that direction and decrease in resistance for the complementary direction occurs (called 'grow_resistance' and 'shrink_resistance'). This resistance is added to grow or shrink, which then requires more points to trip the next correction in that direction.

The user can specify guard-bands that limit the amount QPmod can be modified. QPmod cannot exceed QPmax_pos_mod or become less than -QPmax_neg_mod_abs.

## Triggering

The RC unit begins to modify QPmod occurs only when it is triggered.

Three levels of triggering exist: always, gentle, loose. Always means that RC will be active once sum_actual reaches regions 3 or 4. Gentle will trigger RC once sum_actual reaches regions 2 or 5. Loose waits to trigger RC when sum_actual reaches regions 1 or 6.

RC will deactivate (triggered = false) once sum_actual begins to track sum_target over a series of macroblocks. Specifically, the sign of the rate of change for bytediff is monitored over a window of macroblocks. When the sum of these signs over the window falls within a tolerance value (called 'stable'), triggered will reset to false.

## Panic

When enabled, panic mode will occur whenever sum_actual reaches region 1 and will remain so until sum_actual reaches region 4. When panicking, all macroblocks will be quantized with QP = MB(n).QP + QPmax_pos_mod, clamped to 51.

## User Controls

This unit achieves a large flexibility by allowing the user to define various key parameters. At the per-macroblock level, the values of target and max are specified and will create various shapes of curves that sum_actual will be compared against.

Per-slice, the user can specify the triggering sensitivity and the tolerance required to disable the trigger. Additionally, the user can enable panic detection.

The point values assigned to each of the 6 regions are exposed to the user which allow for asymmetrical control for over and under predictions amongst other things. Additionally, the user can specify the initial values of grow and shrink along with the resistance values applied when correction is invoked.

Lastly, the maximum and minimum values for QPmod are also exposed to the user.

## AVC Encoder MBAFF Support

1. **Algorithm**

   Prediction of current macroblock motion vector is possible from neighboring macroblocks mbAddrA/mbAddrD/mbAddrB/mbAddrC/mbAddrA+1/mbAddrD+1/mbAddrB+1/mbAddrC+1. The selection of these macroblocks depends on coding type(field/frame) of current macroblock pair and the coding of neighboring macroblock pair.

   Selection of these macroblock pairs is described in detail in following sections.

   **1.1 Selection of Top LeftMB pair:** The selection of Top Left MB pair depends on coding type of current and also top left macroblock pair.

   **1.2 Selection of LeftMB pair:** The selection of Left MB pair depends on coding type of current and also left macroblock pair.

   **1.3 Selection of Top MB pair:** The selection of Top MB pair depends on coding type of current and also top macroblock pair.

   **1.4 Selection of Top RightMB pair:** The selection of Top Right MB pair depends on coding type of current and also top right macroblock pair.

   **1.5 Motion Vector and refIdx Scaling:** Motion vectors and reference index of neighboring macroblocks (mbAddrA/mbAddrB/mbAddrC/mbAddrD) should be scaled before using them into prediction equations. Again the scaling depends on coding type of current and neighboring macroblock pair which is described as follows

   - If the current macroblock is a field macroblock and the macroblock mbAddrN is a frame macroblock ...

     ```
     mvLXN[ 1 ] = mvLXN[ 1 ] / 2 (8-214)
     refIdxLXN = refIdxLXN * 2   (8-215)
     ```

   - Otherwise, if the current macroblock is a frame macroblock and the macroblock mbAddrN is a field macroblock ...

     ```
     mvLXN[ 1 ] = mvLXN[ 1 ] * 2 (8-216)
     refIdxLXN = refIdxLXN / 2   (8-217)
     ```

   - Otherwise, the vertical motion vector component mvLXN[ 1 ] and the reference index refIdxLXN remain unchanged.

# MPEG-2

## MPEG2 Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

**MFX_MPEG2_PIC_STATE**

## MPEG2 Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

**MFD_MPEG2_BSD_OBJECT**

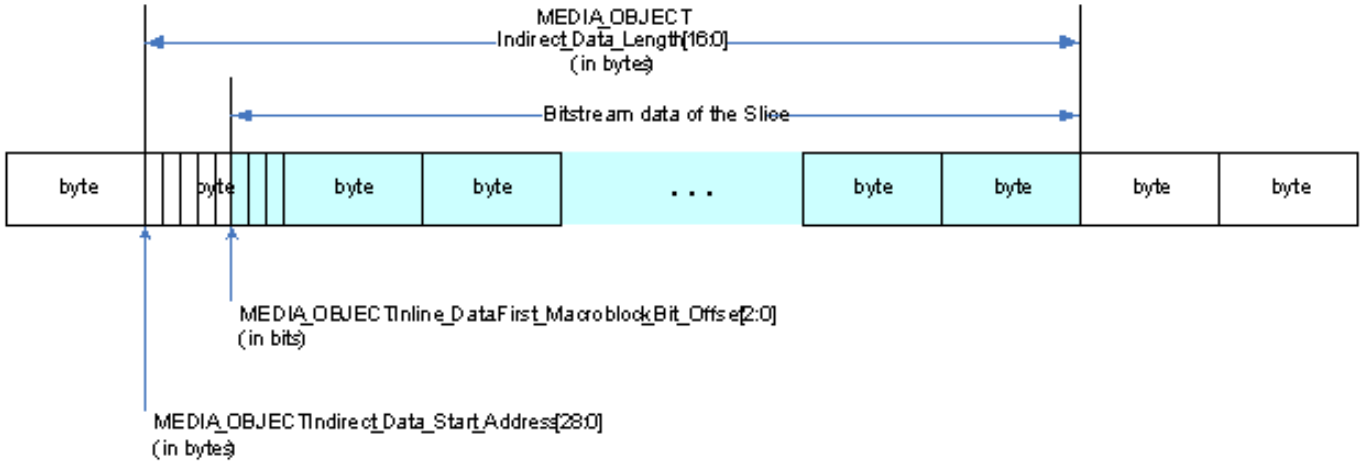
**MFD_MPEG2_BSD_OBJECT Inline Data Description**

## Indirect Data Description

The indirect data start address in MFD_MPEG2_BSD_OBJECT specifies the starting Graphics Memory address of the bitstream data that follows the slice header. It provides the byte address for the first macroblock of the slice. Together with the First Macroblock Bit Offset field in the inline data, it provides the bit location of the macroblock within the compressed bitstream.

The indirect data length in MFD_MPEG2_BSD_OBJECT provides the length in bytes of the bitstream data for this slice. It includes the first byte of the first macroblock and the last **non-zero** byte of the last macroblock in the slice. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. Hardware ignores the contents after the last non-zero byte. *Indirect Data Description* illustrates these parameters for a slice data.

### Indirect data buffer for a slice



## MPEG2 Encoder PAK Commands

The MFC_MPEG2_PAK_INSERT_OBJECT Command is identical to the MFC_AVC_PAK_INSERT_OBJECT command as described in this document.

The MFC_MPEG2_STITCH_OBJECT Command is identical as MFC_AVC_STITCH_OBJECT command as described in this document.

**MFC_MPEG2_SLICEGROUP_STATE**

**MFC_MPEG2_PAK_OBJECT**

## PAK Object Inline Data Description – MPEG-2

The Inline Data includes all the required MB encoding states, constitute part of the Slice Data syntax elements, MB Header syntax elements and their derivatives. It provides information for the following operations:

1. Forward and Inverse Transform
2. Forward and Inverse Quantization
3. Advanced Rate Control (QRC)
4. MB Parameter Construction (MPC)
5. VLC encoding
6. Bit stream packing
7. Internal error handling

These state/parameter values may subject to change on a per-MB basis, and must be provided in each MFC_MPEG2_PAK_OBJECT command. The values set for these variables are retained internally, until they are reset by hardware Asynchronous Reset or changed by the next MFC_MPEG2_PAK_OBJECT command.

The inline data has been designed to match AVC MB structure for efficient transcoding.

Current MB [x,y] address is not sent, it is assumed that the H/W will keep track of the MB count and current MB position internally.

| DWord | Bit | Description |
|---|---|---|
| 1 | 31:27 | Reserved: MBZ |
| | 22-20 | **MvFormat (Motion Vector Size)**. This field specifies the size and format of the input motion vectors.<br>This field is reserved (MBZ) when the **IntraMbFlag** = 1.<br>The valid encodings are:<br>**011 = Unpacked: Two motion vector pairs**<br>Others are reserved.<br>*(The following encodings are intended for other formats:*<br>*001 = 1MV: one 16x16 motion vector*<br>*010 = 2MV: One 16x16 motion vector pair*<br>*011 = 4MV: Four 8x8 motion vectors, or Two 16x8 motion vector pairs*<br>*100 = 8MV: Four 8x8 motion vector pairs*<br>*101 = 16MV: 16 4x4 motion vectors*<br>*110 = 32MV: 16 4x4 motion vector pairs*<br>*111 = Packed, number of MVs is given by **packedMvNum**.)* |
| | 19 | **CbpDcY.** This field specifies if the Luma DC coded. Must be 1 for MPEG-2. |
| | 18 | **CbpDcU.** This field specifies if the Chroma Cb DC coded. Must be 1 for MPEG-2**.** |
| | 17 | **CbpDcV.** This field specifies if the Chroma Cb DC coded. Must be 1 for MPEG-2**.** |
| | 16 | Reserved: MBZ |
| | 15 | **TransformFlag**<br>Used to indicate transformation type for MPEG-2.<br>0 = Frame DCT transformation<br>1 = Field DCT transformation |
| | 14 | **FieldMbFlag**<br>For MPEG-2, this flag is set to 1 if<br>either the picture is in field type<br>or the MB is INTER of field type, i.e. split into two 16x8 field blocks. |

| | | |
|---|---|---|
| | 13 | **IntraMbFlag**<br>This field specifies whether the current macroblock is an Intra (I) macroblock.<br>For I-picture MB (IntraPicFlag =1), this field must be set to 1.<br>This flag must be set in consistent with the interpretation of MbType (inter or intra modes).<br>0: INTER (inter macroblock)<br>1: INTRA (intra macroblock) |
| | 12:8 | **MbType**<br>This field is encoded to match with the best macroblock mode determined as described in the next section. It follows an unified encoding for inter and intra macroblocks according to MFX Encoding reference as shown in Figure A. |
| | 7:3 | Reserved : MBZ |
| | 2 | **SkipMbFlag**<br>By setting it to 1, this field forces an inter macroblock to be encoded as a skipped macroblock. It is equivalent to mb_skip_flag in AVS spec, Hardware honors input MVs for motion prediction and forces CBP to zero.<br>By setting it to 0, an inter macroblock will be coded as a normal inter macroblock. The macroblock may still be coded as a skipped macroblock, according to the macroblock type conversion rules described in the later sub sections.<br>This field can only be set to 1 for certain values of MbType. See details later.<br>This field is only valid for an inter macroblock. Hardware ignores this field for an intra macroblock.<br>0 = not a skipped macroblock<br>1 = is coded as a skipped macroblock<br>**Note:** When this flag is set to 1, the correct MVs are assumed for HW decoder to generate decoded reconstruction frame. |
| | 1:0 | **InterMbMode**<br>This field is provided to carry redundant information as that encoded in MbType.<br>This field is only valid if **IntraMbFlag** =0, otherwise, it is ignored by hardware. |
| 2 | 31:16 | **MbYCnt (Vertical Origin).** This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks.<br>Format = U16 in unit of macroblock. |
| | 15:0 | **MbXCnt (Horizontal Origin).** This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks.<br>Format = U16 in unit of macroblock. |

| | | |
|---|---|---|
| 3 | 31:24 | **MaxSizeInWord**<br>PAK should not exceed this budget accumulatively, otherwise it will trickle the PANIC mode. |
| | 23:16 | **TargetSizeInWord**<br>PAK should use this budget accumulatively to decide if it needs to limit the number of non-zero coefficients. |
| | 15:13 | **MBZ** |
| | 12:0 | **Cbp –** Coded Block Pattern. This field specifies whether blocks are present or not.<br>Format = 6-bit mask (or 8-bit, & 12-bit, for 422 and 444).<br>Bit 11: Y0Bit 10: Y1Bit 9: Y2Bit 8: Y3<br>Bit 7: Cb4Bit 6: Cr5Bits 0-5: MBZ |
| 4 | 31 | **LastMbInSlice –** the last MB in a slice. |
| | 30 | **FirstMbInSlice –** the first slice in a slice, it requires slice header insertion. |
| | 29:28 | **MBZ** |
| | 27 | **EnableCoeffClamp**<br>1 = the magnitude of coefficients of the current MB will be clamped based on the clamping matrix after quantization<br>0 = no clamping |
| | 26 | **LastMbInSG**<br>1 – the current MB is the last MB in the current slice group. |
| | 25 | **MbSkipConvDisable**<br>This is a per-MB level control to enable and disable skip conversion. This field is ORed with SkipConvDisable field. This field is only valid for a P or B slice. It must be zero for other slice types. Rules are provided in Macroblock Type Conversion Rules<br>0 - Enable skip type conversion for the current macroblock<br>1 – Disable skip type conversion for the current macroblock |
| | 24 | **FirstMbInSG**<br>1 – the current MB is the last MB in the current slice group. |
| | 23:20 | **MBZ** |

<table>
<tr><td></td><td>19:16</td><td>MvFieldSelect – Motion Vertical Field Select. A bit-wise representation of a long [2][2] array as defined in §6.3.17.2 of the ISO/IEC 13818-2 (see also §7.6.4).
<table>
<tr><th>Bit</th><th>MVector[r]</th><th>MVector[s]</th><th>MotionVerticalFieldSelect Index</th></tr>
<tr><td>16</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>17</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>18</td><td>1</td><td>0</td><td>2</td></tr>
<tr><td>19</td><td>1</td><td>1</td><td>3</td></tr>
</table>
Format = MC_MotionVerticalFieldSelect.<br>0 = The prediction is taken from the top reference field.<br>1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>15:5</td><td>MBZ Reserved</td></tr>
<tr><td></td><td>4:0</td><td>QpScaleCode</td></tr>
<tr><td>5</td><td>31:16</td><td>MV[0][0].y – the y coordinate of the first forward MV<br>if Mv[0][0] n/a:<br>if Mv[1][0] available, it MUST be set to the same value as Mv[1][0].<br>else it MUST be set to the value 0</td></tr>
<tr><td></td><td>15:0</td><td>MV[0][0].x – the x coordinate of the first forward MV<br>if Mv[0][0] n/a:<br>if Mv[1][0] available, it MUST be set to the same value as Mv[1][0].<br>else it MUST be set to the value 0</td></tr>
<tr><td>6</td><td>31:0</td><td>MV[1][0] – the first backward MV<br>if Mv[1][0] n/a: it MUST be set to the same value as Mv[0][0]</td></tr>
<tr><td>7</td><td>31:0</td><td>MV[0][1] – the second forward MV<br>if Mv[0][1] n/a:<br>if Mv[1][1] available, it MUST be set to the same value as Mv[1][1].<br>else it MUST be set to the same value as Mv[0][0]</td></tr>
<tr><td>8</td><td>31:0</td><td>MV[1][1] – the second backward MV<br>if Mv[1][1] n/a: it MUST be set to the same value as Mv[1][0]</td></tr>
</table>

The mapping between MPEG-2 spec and MfxMbCode can be achieved according to the following:

1) Renamed variables with identical meaning:

| MPEG-2 Spec | MFX API | Value |
|---|---|---|
| **macroblock_quant** | **MbQuantPresent** | 0 or 1 |
| **macroblock_intra** | **IntraMbFlag** | 0 or 1 |
| **dct_type** | **Transform8x8Flag** | 0 or 1 |
| **macroblock_pattern** | **Cbp8x8** | remapped |

2) Macroblock type remapping:

| | | B-spec Entry | | | | | MPEG-2 Spec | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame Type | Mb Type | Intra Mb Flag | Skip Mb Flag | Mb Type 5Bits | Field Mb Flag | Inter Mb Mode | macroblock_intra | motion_type_bit 0 | motion_type_bit 1 | motion_forward | motion_backward |
| IPB | Intra | 1 | 0 | 1Ah | 0/1 | - | 1 | - | - | - | - |
| **P**<br>**B**<br>B | Skip | 0 | 1 | 01h<br>02h<br>03h | 0/1 | 0 | 0 | - | - | 1<br>0<br>1 | 0<br>1<br>1 |
| P | 0-MV* | 0 | 0 | 01h | 0/1 | 0 | 0 | - | - | 0 | 0 |
| P Frame | Frame type | 0 | 0 | 01h | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| P Frame | Field type | 0 | 0 | 04h | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| P Frame | dual prime | 0 | 0 | 19h | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| P Field | One 16x16 | 0 | 0 | 01h | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

| | | B-spec Entry | | | | | MPEG-2 Spec | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame Type | Mb Type | Intra Mb Flag | Skip Mb Flag | Mb Type 5Bits | Field Mb Flag | Inter Mb Mode | macroblock_intra | motion_type_bit 0 | motion_type_bit 1 | motion_forward | motion_backward |
| P Field | Two 16x8 | 0 | 0 | 04h | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| P Field | dual prime | 0 | 0 | 19h | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| B Frame | Frame type | 0 | 0 | 01h<br>02h<br>03h | 0 | 0 | 0 | 0 | 1 | 1<br>0<br>1 | 0<br>1<br>1 |
| B Frame | Field type | 0 | 0 | 04h<br>06h<br>14h | 1 | 1 | 0 | 1 | 0 | 1<br>0<br>1 | 0<br>1<br>1 |
| B Field | One 16x16 | 0 | 0 | 01h<br>02h<br>03h | 1 | 0 | 0 | 1 | 0 | 1<br>0<br>1 | 0<br>1<br>1 |
| B Field | Two 16x8 | 0 | 0 | 04h<br>06h<br>14h | 1 | 1 | 0 | 0 | 1 | 1<br>0<br>1 | 0<br>1<br>1 |

- Notice that there is no special way to indicate 0 motion vector case for P frame. It is for PAK to handle internally by checking up the motion vector values.
- Notice also, the MbType5bits is adapted from AVC DXVA macroblock types. It may seems awkward from MPEG-2 perspective, but provides a common VME interface for us for simpler HW design and help the advanced transcoding solution.

# MFX HW Interface and DXVA Conversion

## Map DXVA to HW PRM

| | | HW | |
|---|---|---|---|
| Location | | PRM | |
| BYTE | | MPEG-2 | DXVA |
| | DW0 | | |
| 0 | | MbMode | |
| 0.0-1 | 0[0-1] | InterMbMode | see (A) |
| 0.2 | 0[2] | SkipMbFlag | <-MBskipsFollowing |
| 0.3 | 0[3] | mbz | |
| 0.4-5 | 0[4-5] | IntraMbMode | IntraMacroblock |
| 0.6 | 0[6] | mbz | |
| 0.7 | 0[7] | FieldMbPolarity | derived |
| 1 | | MbType | |
| 1.0-4 | 0[8-12] | MbType5Bits | see (A) |
| 1.5 | 0[13] | IntraMbFlag | IntraMacroblock |
| 1.6 | 0[14] | FieldMbFlag | see (A) |
| 1.7 | 0[15] | TransformFlag | FieldResidual |
| 2 | | MbFlag | |
| 2.0 | 0[16] | ResidDataFlag | HostResDiff |
| 2.1 | 0[17] | CbpDcV | PAK control |
| 2.2 | 0[18] | CbpDcU | PAK control |
| 2.3 | 0[19] | CbpDcY | PAK control |
| 2.4-6 | 0[20-22] | MvFormat | = 3, derived |
| 2.7 | 0[23] | mbz | |
| 3 | 0[24-31] | PackedMvNum | see (A) |
| | DW1 | | |
| 4-5 | 1[0-15] | MbXCnt | wMBaddress |
| 6-7 | 1[16-31] | MbYCnt | wMBaddress |
| | DW2 | | |
| 8 | 2[0-7] | | bNumCoef[0] |
| 8.0-5 | 2[0-5] | mbz | |
| 8.6-7 | 2[6-7] | CbpAcUV | PAK control |
| 9 | 2[8-11] | CbpAcY | PAK control |
| | 2[12-15] | mbz | |
| 10 | 2[16-23] | TargetedSzInWord | |

| | | HW | |
|---|---|---|---|
| Location | | PRM | |
| BYTE | | MPEG-2 | DXVA |
| 11 | 2[24-31] | **MaxSzInWord** | |
| | DW3 | | |
| 12 | | **Qscale** | derived |
| 12.0-6 | 3[0-6] | **QScaleCode** | |
| 12.7 | 3[7] | **QScaleType** | |
| 13 | 3[8-15] | **mbz** | |
| 14 | 3[16-19] | **MvFieldSelect** | `MvertFieldSel` |
| | 3[20-23] | **mbz** | |
| 15 | | **MbExtFlag** | |
| 15.0 | 3[24] | **mbz** | |
| 15.1 | 3[25] | **SkipMvConvDisable** | |
| 15.2 | 3[26] | **LastMbFlag** | PAK control |
| 15.3 | 3[27] | **EnableCoeffClamp** | PAK control |
| 15.4-5 | 3[28-29] | **MbScanMethod** | `MBscanMethod` |
| 15.6 | 3[30] | **NewSliceFlag** | PAK control |
| 15.7 | 3[31] | **EndSliceFlag** | PAK control |
| | DW4-7 | | |
| 16-32 | 4-7[all] | **MV[2][2][2]** | `MVector[4][2]` |

**(A):** Set **InterMbMode, MbType5bits, FieldMbFlag**, and **PackedMvNum** from DXVA parameters:

```
 if(IntraMacroblock) return (TYPE_INTRA);
 else if(MotionType==3){ // dual prime
    MbType5bits = 0x19; FieldMbFlag = 0; InterMbMode = 0; PackedMvNum = 2;     return
(DUAL_PRIME);
 }
 else{
    IsFieldFrame = a PicState derivative;     switch(MotionType+IsFieldFrame{
       case 1: // Two 16x8 field in Frame Frame
       case 3: // Two 16x8 field in Field Frame
          FieldMbFlag = 1; InterMbMode= 1;          switch(MotionForward |Motionbackward «1)){
             case 1:
                MbType5bits = 4; PackedMvNum = 2;                break;
             case 2:
                MbType5bits = 6; PackedMvNum = 2;                break;
             case 3:
                MbType5bits = 0x14; PackedMvNum = 4;                break;
          }
          break;
       case 2: // 16x16 block in either case
          FieldMbFlag = IsFieldFrame; InterMbMode = 0;
switch(MotionForward|(Motionbackward«1)){
             case 1:
                MbType5bits = 1; PackedMvNum = 1;                break;
             case 2:
                MbType5bits = 2; PackedMvNum = 1;                break;
             case 3:
                MbType5bits = 3; PackedMvNum = 2;                break;
          }
          break;
    }
 }
```

## Map HW PRM to DXVA

| | DXVA | PRM |
|---|---|---|
| | | MPEG-2 |
| 0-1 | wMBaddress | = **MbYCnt*MbW + MbXCnt** |
| 2-3 | wMBtype | |
| 2.0 | IntraMacroblock | = **IntraMbFlag** |
| 2.1 | MotionForward | see (B) |
| 2.2 | MotionBackward | see (B) |
| 2.3 | Motion4MV | VC-1 only, MBZ for Mpeg-2 |
| 2.4 | Reserved | |
| 2.5 | FieldResidual | = **TranformFlag** |
| 2.6-7 | MBscanMethod | = **MbScanMethod** |
| 3.0-1 | MotionType | see (B) |
| 3.2 | HostResDiff | = **ResidDataFlag** |
| 3.3 | Reserved | |
| 3.4-7 | MvertFieldSel | = **MvFieldSelect** |
| | | |
| 4 | MBskipsFollowing | count **SkipMbFlag** |
| 5-7 | MBdataLocation | n/a |
| 8-9 | wPatternCode | = **CbpAcY\|UV** |
| 10-15 | bNumCoef[6] | **n/a** |
| 16-32 | MVector[4][2] | = **MV[2][2][2]** |

**(B):** Set **MBtype** and **MotionType** from PRM interface

```
if(MbIntraFlag) return (TYPE_INTRA);
else if(MbType5Bits&8){ // dual prime
        MotionForward= 1;
        MotionBackward= 0;
        MotionType= 3;
              return (DUAL_PRIME);
        }
         else{
        // redundant: InterMbMode = !!(MbType5Bits&4);
        if(InterMbMode){
                    MotionForward= !(MbType5Bits&2);
                    MotionBackward= !!(MbType5Bits&0x12);
              }
               else{
                    MotionForward= (MbType5Bits&1);
                    MotionBackward= !!(MbType5Bits&2);
              }
              MotionType = 2-(InterMbMode^FieldMbFlag);
              // equivalently the 2 bits are:
              // MotionType0 = (InterMbMode^FieldMbFlag);
              // MotionType1 = ~MotionType0;
               return (TYPE_INTER);
}
```

## VC1 Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

**MFX_VC1_PRED_PIPE_STATE**

**MFX_VC1_DIRECTMODE_STATE**

## VC1 Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

**MFD_VC1_LONG_PIC_STATE**

**AltPQuantConfig** and **AltPQuantEdgeMask** are derived based on the following variables: *DQUANT*, *DQUANTFRM*, *DQPROFILE*, *DQSBEDGE*, *DQDBEDGE*, and *DQBILEVEL* defined in the VC1 standard, as shown in the following table.

### Definition of AltPQuantConfig and AltPQuantEdgeMask

| Inputs | | | | | | Outputs | | Description |
|---|---|---|---|---|---|---|---|---|
| DQUANT | DQUANT FRM | DQ PROFILE | DQDB EDGE | DQSB EDGE | DQBI LEVEL | AltPQuant Config | AltPQuant EdgeMask | |
| 0 | - | - | - | - | - | 00b | 0000b | No **AltPQuant** |
| 1 | 0 | - | - | - | - | 00b | 0000b | No **AltPQuant** |
| 1 | 1 | 11b | - | - | 0 | 10b | 0000b | All MBs are different with *MQDIFF* and *ABSMQ* |
| 1 | 1 | 11b | - | - | 1 | 11b | 0000b | All MBs may switch with 1-bit *MQDIFF* |
| 2 | - | - | - | - | - | 01b | 1111b | All edge MBs |
| 1 | 1 | 00b | - | - | - | 01b | 1111b | All edge MBs |
| 1 | 1 | 01b | 00b | - | - | 01b | 0011b | Left and top MBs |
| 1 | 1 | 01b | 01b | - | - | 01b | 0110b | Top and right MBs |
| 1 | 1 | 01b | 10b | - | - | 01b | 1100b | Right and bottom MBs |
| 1 | 1 | 01b | 11b | - | - | 01b | 1001b | Bottom and left MBs |
| 1 | 1 | 10b | - | 00b | - | 01b | 0001b | Left MBs |
| 1 | 1 | 10b | - | 01b | - | 01b | 0010b | Top MBs |
| 1 | 1 | 10b | - | 10b | - | 01b | 0100b | Right MBs |
| 1 | 1 | 10b | - | 11b | - | 01b | 1000b | Bottom MBs |

#### MFD_VC1_SHORT_PIC_STATE

Intel HW does not use the MVMODE and MVMODE2 provided at the revised DXVA2 VC1 VLD interface, instead, HW will decode them directly from the bitstream picture header.

#### MFD_VC1_BSD_OBJECT

For VC1, a slice/picture is always started with MB x positon equal to 0. Hence, no need to include in the Object Command.

## Handling Emulation Bytes

In general, VC1 BSD unit is capable of handling emulation prevention bytes. However, there is a corner case that requires host software's intervention. Host software needs to overwrite the emulation byte if it overlaps the macroblock layer decode and there is not enough information for the hardware to detect the emulation byte.

The emulation bytes might have an overlap between the picture states and the first macroblock data. If the emulation bytes are 0x00 **0x000x03** 0x00 and the macroblock data starts in the middle of byte1 (**0x00**), then the host software needs to overwrite the **0x03** byte location with the previous byte (**0x00**) and change the byte offset accordingly. The hardware wouldn't know what the 1st byte was and will miss this **0x03** removal.

# VP8

## VP8 Common Commands

Following are VP8 Common Commands:

### MFX_VP8_PIC_STATE

For VP8 HW PAK, there are four VP8 versions supported and their programming is shown in Table1 below.

| Version | MC Filter Select | Chroma Full Pixel MC Filter Mode | DBLK FilterType | DBLK FilterLevel for Segment0 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Any FilterLevel |
| 1 | 1 | 0 | 1 | Any FilterLevel |
| 2 | 1 | 0 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 |

#### VP8 Version

**MC Filter Select:** MFX_VP8_PIC_STATE.DW2.Bit0

**Chroma Full Pixel MC Filter Mode:** MFX_VP8_PIC_STATE.DW2.Bit1

**DBKL Filter Type:** MFX_VP8_PIC_STATE.DW2.Bit4

**DBLK Filter Level for Segment0:** MFX_VP8_PIC_STATE.DW3.Bit5:0

1. Note that when multiple segment is enabled, if Segment0 DBLK Filter is programmed to 0, Segment1,2,3 DBLK Filter should be set to 0 as well.
2. Note that MFX_VP8_Encoder_CFG.BW22.Bit22:20 (Bitstream Format Version). This field is used for generating Uncompressed header only. It is not used to control any Filter.

## VP8 Decoder Commands

Following are VP* Decoder Commands:

### MFD_VP8_BSD_OBJECT

# JPEG and MJPEG

## JPEG Decoder Commands

Following are JPEG Decoder Commands:

### MFD_JPEG_BSD_OBJECT

For JPEG decoding, the following program note is informative.

For **Rotation**, it is important to note that rotation of 90 or 270 degrees also requires exchanging **FrameWidthInBlksMinus1** with **FrameHeightInBlksMinus1** in the command. In addition, the rotation of 90 or 270 degrees also requires transportation of the quantization matrix will be transposed into the position (y, x).

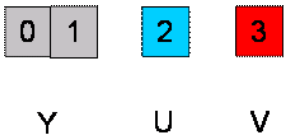
**Chroma type** is determined by the values of horizontal and vertical sampling factors of the components (*Hi* and Vi where *i* is a component id) in the Frame header as shown in the following table.

| | H1 | H2 | H3 | V1 | V2 | V3 |
|---|---|---|---|---|---|---|
| 0: YUV400 | r | Not available | Not available | r | Not available | Not available |
| 1: YUV420 | 2 | 1 | 1 | 2 | 1 | 1 |
| 2: YUV422H_2Y | 2 | 1 | 1 | 1 | 1 | 1 |
| 3: YUV444 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4: YUV411 | 4 | 1 | 1 | 1 | 1 | 1 |
| 5: YUV422V_2Y | 1 | 1 | 1 | 2 | 1 | 1 |
| 6: YUV422H_4Y | 2 | 1 | 1 | 2 | 2 | 2 |
| 7: YUV422V_4Y | 2 | 2 | 2 | 2 | 1 | 1 |

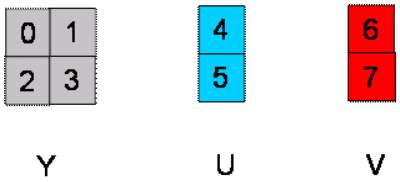
For YUV400, the value of *V1* can be 1, 2, or 3 and will be same as the value of *H1*, and the Minimum coded unit (MCU) is one 8x8 block. For the other chroma formats, if non-interleaved data, the MCU is one 8x8 block. For interleaved data, the MCU is the sequence of block units defined by the sampling factors of the components.

For example, the following figures show the MCU structures of interleaved data and the decoding order of blocks in the MCU:

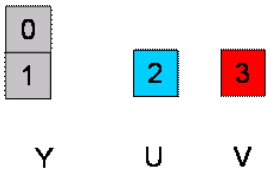
**422H_2Y**



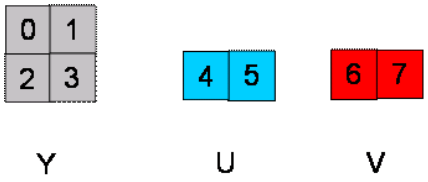

**422H_4Y**




**422V_2Y**



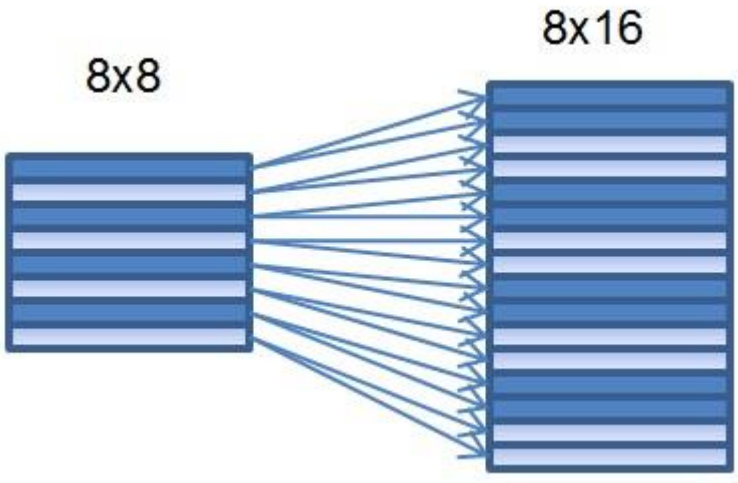

**422V_4Y**



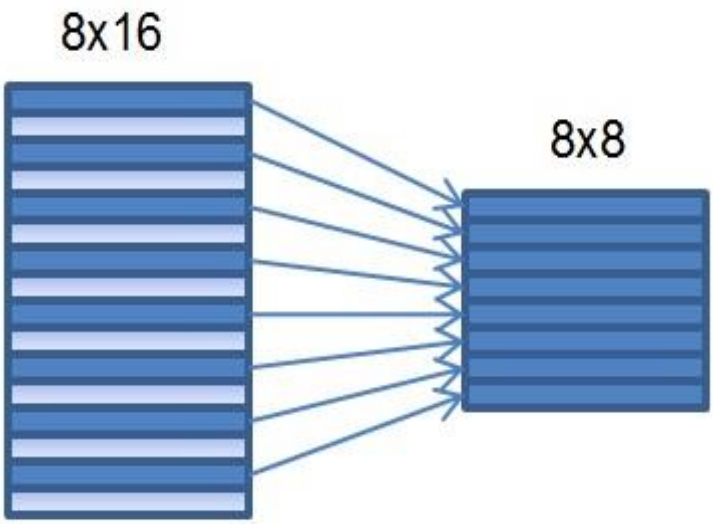

If picture width *X* in the Frame header is not a multiple of 8, the decoding process needs to extend the number of columns to complete the right-most sample blocks. If the component is to be interleaved, the decoding process needs to extend the number of samples by one or more additional blocks so that the number of blocks is an integer multiple of *Hi*. In other words, "The number of blocks in width" in the table should be an integer multiple of (8x*H1*). Similarly, if picture height *Y* in the Frame header is not a multiple of 8, the decoding process needs to extend the number of lines to complete bottom-most block-row. If the component is to be interleaved, the decoding process also needs to extend the number of lines by one or more additional block-rows so that the number of block-row is an integer multiple of (8x*V1*). For example, if non-interleaved YUV411 with *X*=270, then "The number of blocks in width" shall be (270 + 7) / 8 = 34, where "/" is integer division. Therefore, **FrameWidthInBlksMinus1** is set to 33. However, for interleaved data, "The number of blocks in width" shall be ((270 + 31) / 32) x 4 = 36. Therefore, **FrameWidthInBlksMinus1** is set to 35.

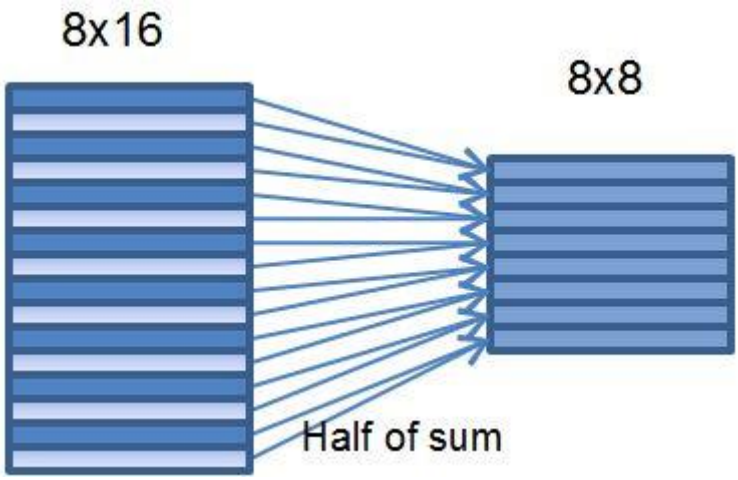
**The following description applies only to [BDW]:**

VertUpSamplingEnb is used to convert an input chroma420 to an output chroma422 in the surface format of YUY2 or UYVY. To enable this flag, the input should be interleaved Scan, InputFormatYUV should be set to YUV420, and OutputFormatYUV should be set to YUY2 or UYVY. Vertical 2:1 up-sampling is only applied to chroma blocks where each line of 8x8 block pixels is replicated to make 8x16 U/V blocks. For example:




VertDownSamplingEnb is used to convert an input chroma422 to an output chroma420 in the surface format NV12. To enable this flag, the input should be interleaved Scan, InputFormatYUV should be set to YUV422H_2Y or YUV422H_4Y, and OutputFormatYUV should be set to NV12. Combined with AvgDownSampling flag, the following table and figures show the down-sampling methods.

| VertDownSamplingEnb | AvgDownSampling | Down-Sampling Methods |
|---|---|---|
| 0 | 0 or 1 | No down-sampling. |
| 1 | 0 | Drop every other line:<br>  <br> |

| VertDownSamplingEnb | AvgDownSampling | Down-Sampling Methods |
|---|---|---|
| 1 | 1 | Average vertically neighboring two pixels:   |

The recent history forJPEG Decoder Commands are described in the following:

- InputFormat on HSW ULT/C0+ is the same as IVB, and should be programmed the same way as IVB.
- If the InputFormat is YUV400 or YUV444 or YUV411, then output cannot be NV12, YUY2 or UYVY, it has to be planar (like legacy IVB). But for 420 and 422 InputFormat, there's a choice of having Planar, NV12, YUY2 or UYVY OutputFormat (new addition from HSW ULT/C0+). And the surface state should be programmed accordingly.
- Refer "Output Format YUV" field for more details.

## MFX_JPEG_HUFF_TABLE_STATE

## JPEG Encoder Commands

JPEG Encoder Command Sequence:

**MFX_PIPE_MODE_SELECT**

**MFX_SURFACE_STATE**

**MFX_PIPE_BUF_ADDR_STATE**

**MFX_JPEG_PIC_STATE**

**MFX_FQM_STATE [BDW]** (One each for Luma, CB and CR)

**MFX_PAK_INSERT_OBJECT** (Multiple commands can be given based on the need)

Following are JPEG Encoder Commands:

**MFX_JPEG_PIC_STATE**

**Programming Note:** For completion of partial MCUs in JPEG encoding, it is important to note the following:

If the image's dimensions are not an exact multiple of the MCU size, the encoded data should include padding to round up to the next complete MCU, which is called completion of partial MCU. If the number of lines is not aligned with MCU structure (not a multiple of MCU size, i.e. 8, 16, 32), the encoding process needs to extend the number of lines to complete the bottom-most MCU-row. Similarly, if the number of samples per line is not aligned with MCU structure, the encoding process needs to extend the number of columns to complete the right-most sample MCUs. JPEG standard recommends that any incomplete MCUs be completed by replication of the right-most column and the bottom line of each component Y, U, and V.

The following equations are used to set the command for encoding partial MCUs.

**FrameWidthInBlksMinus1** = (((*X* + (*H*$_1$ *8 -1)) / (*H*$_1$ *8)) * *H*$_1$) – 1

**FrameHeightInBlksMinus1** = (((*Y* + (*V*$_1$*8 -1)) / (*V*$_1$*8)) * *V*$_1$) – 1

```
For YUV400,
    PixelsInHoriLastMCU = X % 8
    PixelsInVertLastMCU = Y % 8

For YUV420,
    PixelsInHoriLastMCU = X % 16 if X % 2 = 0, ((X % 16) + 1) % 16 if X % 2 = 1
    PixelsInVertLastMCU = Y % 16 if Y % 2 = 0, ((Y % 16) + 1) % 16 if Y % 2 = 1

For YUV422H_2Y,
    PixelsInHoriLastMCU = X % 16 if X % 2 = 0, ((X % 16) + 1) % 16 if X % 2 = 1
    PixelsInVertLastMCU = Y % 8

    X: the number of samples per line in Y-image

    Y: the number of lines in Y-image

    H1: horizontal sampling factor of Y-image in the Frame header

    V1:  vertical sampling factor of Y-image in the Frame header
```

Note that PixelsInHoriLastMCU=0 does not mean the num of pixels in the right-most MCUs = 0, but does mean that the right-most MCUs are fully filled with pixels, i.e., not a partial MCU.

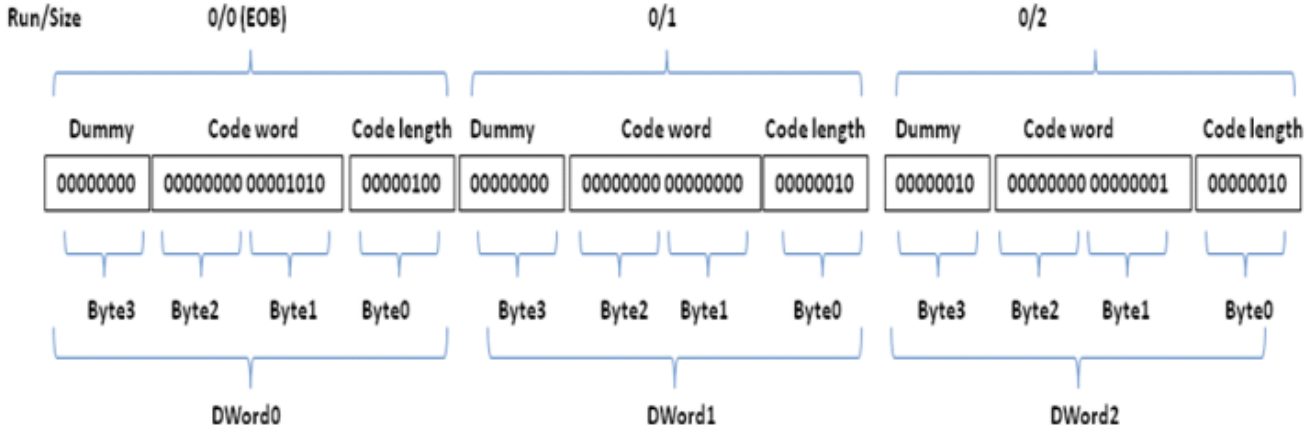
For example, for input image dimension 17x26 pixels and an interleaved Scan, the following equations and the table show how to set the command for each OutputMcuStructure.

| | YUV400 | YUV420 | YUV422H_2Y |
|---|---|---|---|
| MCU size of Y | 8x8 | 16x16 | 16x8 |
| MCU size of U and V | 8x8 | 8x8 | 8x8 |
| $H_1$ and $V_1$ | 1, 1 | 2, 2 | 2, 1 |
| **FrameWidthInBlksMinus1** | 2 | 3 | 3 |
| **FrameHeightInBlksMinus1** | 3 | 3 | 3 |
| PixelsInHoriLastMCU | 1 | 2 | 2 |
| PixelsInVertLastMCU | 2 | 10 | 2 |

The JPEG standard Table K.5 shows the real table of code length and code word as follows:

| Run/Size | Code length | Code word |
|---|---|---|
| 0/0 (EOB) | 4 | 1010 |
| 0/1 | 2 | 00 |
| 0/2 | 2 | 01 |
| 0/3 | 3 | 100 |
| 0/4 | 4 | 1011 |
| 0/5 | 5 | 11010 |
| 0/6 | 7 | 1111000 |
| 0/7 | 8 | 11111000 |
| 0/8 | 10 | 1111110110 |
| 0/9 | 16 | 1111111110000010 |
| 0/A | 16 | 1111111110000011 |

It is not necessary to send Run/size in the command as driver will send the increasing order of run/size. Each symbol aligns to a DWord with the following byte structure. Each DWord (a group of 4 bytes) contains Byte0 for Code length, Byte1 and Byte2 for Code word, and Byte3 for dummy.




Driver will program to always send 12 pairs of Code length and Code Word in DC coefficient table and 162 pairs in AC coefficient table. When a Huffman table contains valid full entries of Run/Size, all the Code word and Code length will not be zero. If a Huffman table is customized or optimized, the table can contain smaller set of Code length and Code Word, i.e., the number of entries of the real Huffman table will be less than 12 for DC, or less than 162 for AC. For the customized Huffman table, driver will set the missing entry (Run/Size) to Code length = 0 and Code word = 0.

**MFX_PAK_INSERT_OBJECT**

# More Decoder and Encoder

## MFD IT Mode Decode Commands

These are decoder-only commands to support the IT-mode specified in DXVA interface.

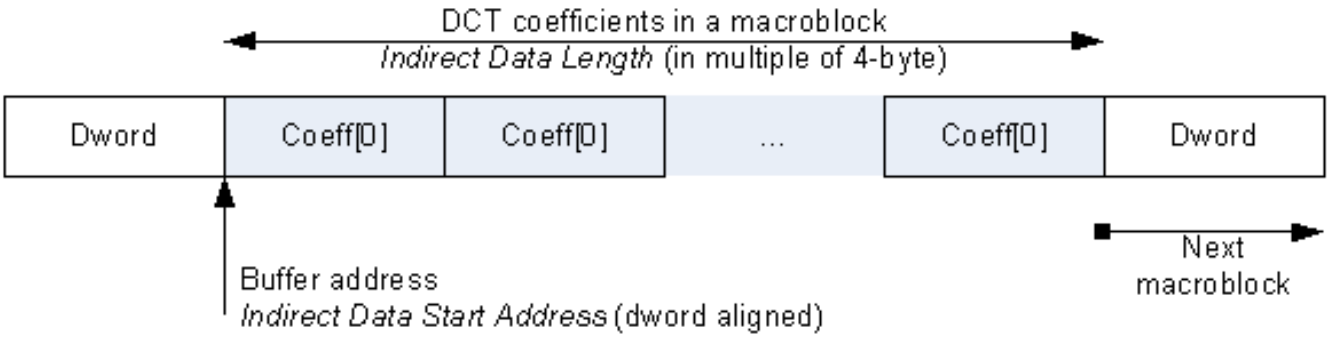
**MFD_IT_OBJECT**

## Common Indirect IT-COEFF Data Structure

Transform-domain residual data block in AVC-IT, VC1-IT and MPEG2-IT mode follows the same data structure.

The indirect IT-COEFF data start address in MFD_IT_OBJECT command specifies the doubleword aligned address of the first non-zero DCT coefficient of the first block of the macroblock. Only the non-zero coefficients are present in the data buffer and they are packed in the 8x8 block sequence of Y0, Y1, Y2, Y3, Cb4 and Cr5, as shown in *Common Indirect IT-COEFF Data Structure*. When an 8x8 block is further subdivided into 4x4 subblocks, the coefficients, if present, are organized in the subblock order. The smallest subblock division is referred to as a **transform block**. The indirect IT-COEFF data length in the command includes all the non-zero coefficients for the macroblock. It must be doubleword aligned.

## Structure of the IDCT Compressed Data Buffer




Each non-zero coefficient in the indirect data buffer is contained in a doubleword-size data structure consisting of the coefficient index, end of block (EOB) flag and the fixed-point coefficient value in 2's compliment form. As shown in *Common Indirect IT-COEFF Data Structure*, *index* is the row major 'raster' index of the coefficient **within a transform block** (*please note that it is not converted to 8x8 block basis*). A coefficient is a 16-bit value in 2's complement.

## Structure of a transform-domain residue unit

| DWord | Bit | Description |
|---|---|---|
| 0 | 31:16 | **Transform-Domain Residual (coefficient) Value.** This field contains the value of the non-zero transform-domain residual data in 2's compliment. |
| | 15:7 | Reserved: MBZ |
| | 6:1 | **Index.** This field specifies the raster-scan address (raw address) of the coefficient within the transform block. For a coefficient at Cartesian location (row, column) = (y, x) in a transform block of width W, Index is equal to (y * W + x). For example, coefficient at location (row, column) = (0, 0) in a 4x4 transform block has an index of 0; that at (2, 3) has an index of 2*4 + 3 = 11.<br>The valid range of this field depends on the size of the transform block.<br>Format = U6<br>Range = [0, 63] |
| | 0 | **EOB (End of Block).** This field indicates whether the transform-domain residue is the last one of the current transform block. |

## Allowed transform block dimensions per coding standard

| Transform Block Dimension | AVC | VC1 | MPEG2 |
|---|---|---|---|
| 8x8 | Yes | Yes | Yes |
| 8x4 | No | Yes | No |
| 4x8 | No | Yes | No |
| 4x4 | Yes | Yes | No |

For AVC, there is intra16x16 mode, in which the DC Luma coefficients of all 4x4 sub-blocks within the current MB are sent separately in its own 4x4 Luma block. As such, only 15 coefficients remains in each of the 16 4x4 Luma blocks.

## Inline Data Description in AVC-IT Mode

The Inline Data includes all the required MB decoding states, extracted primarily from the Slice Data, MB Header and their derivatives. It provides information for the following operations:

1. Inverse Quantization
2. Inverse Transform
3. Intra and inter-Prediction decoding operations
4. Internal error handling

IT Mode supports only packed MV data as specified in the DXVA spec.

These state/parameter values may subject to change on a per-MB basis, and must be provided in each MFD_IT_OBJECT command. The values set for these variables are retained internally, until they are reset by hardware Asynchronous Reset or changed by the next MFC_AVC_PAK_OBJECT command.

The inline data has been designed to match the DXVA 2.0, with the exception of the starting byte (DW0:0-7) and the ending dword (DW7:0-31).

The Deblocker Filter Control flags (FilterInternalEdgesFlag, FilterTopMbEdgeFlag and FilterLeftMbEdgesFlag) are generated by H/W, which are depending on MbaffFrameFlag, CurrMbAddr, PicWidthInMbs and disable_deblocking_filter_idc states.

Current MB [x,y] address is not sent, it is assumed that the H/W will keep track of the MB count and current MB position internally.

| DWord | Bit | Description |
|---|---|---|
| 0 | 31:24 | **MvQuantity**<br>Specify the number of MVs (in unit of motion vector, 4 bytes each) to be fetched for motion compensation operation.<br>Decoder IT mode only supports packed MV format (DXVA). This field specifies the exact number of MVs present for the current MB.<br>For a P-Skip MB, there is still 1 MV being sent (Skip MV is sent explicitly); for a B-Direct/Skip MB, there are 2 MVs being sent.<br>For an Intra-MB, MvQuantity is set to 0.<br>MvQuantity = 0, signifies there is no MV indirect data for the current MB.<br>This field must be set in consistent with **Indirect MV Data Length**, so as not to exceed its bound<br>Unsigned. |
| | 23:20 | Reserved MBZ (DXVA) |

| DWord | Bit | Description |
|---|---|---|
| | 19 | **DcBlockCodedYFlag**<br>1 – the 4x4 DC-only Luma sub-block of the Intra16x16 coded MB is present; it is still possible that all DC coefficients are zero.<br>0 – no 4x4 DC-only Luma sub-block is present; either not in Intra16x16 MB mode or all DC coefficients are zero. |
| | 18 | **DcBlockCodedCbFlag**<br>For 4:2:0 case :<br>1 – the 2x2 DC-only Chroma Cb sub-block of all coded MB (any type) is present; it is still possible that all DC coefficients are zero.<br>0 – no 2x2 DC-only Chroma Cb sub-block is present; all DC coefficients are zero. |
| | 17 | **DcBlockCodedCrFlag**<br>For 4:2:0 case :<br>1 – the 2x2 DC-only Chroma Cr sub-block of all coded MB (any type) is present; it is still possible that all DC coefficients are zero.<br>0 – no 2x2 DC-only Chroma Cr sub-block is present; all DC coefficients are zero. |
| | 16 | Reserved MBZ (DXVA) |
| | 15 | **Transform8x8Flag**<br>0: indicates the current MB is coded with 4x4 transform and therefore the luma residuals are presented in 4x4 blocks.<br>1: indicates the current MB is coded with 8x8 transform and therefore the luma residuals are presented in 8x8 blocks.<br>Same as the transform_szie_8x8_flag syntax element in AVC spec. |
| | 14 | **MbFieldFlag**<br>This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode.<br>1 = Field macroblock<br>0 = Frame macroblock<br>This field is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode.<br>Same as the mb_field_decoding_flag syntax element in AVC spec. |
| | 13 | **IntraMbFlag**<br>This field specifies whether the current macroblock is an Intra (I) macroblock. |

| DWord | Bit | Description |
|---|---|---|
| | | 0 – not an intra MB<br>1 – is an intra MB<br>I_PCM is considered as Intra MB.<br>For I-picture MB (IntraPicFlag =1), this field must set to 1.<br>This flag must be set in consistent with the interpretation of MbType (inter or intra modes). |
| | 12:8 | **MbType**<br>This field carries the Macroblock Type. The meaning depends on IntraMbFlag.<br>If IntraMbFlag is 1, this field is the intra macroblock type as defined in MbType definition for Intra Macroblock .<br>If IntraMbFlag is 0, this field is the inter macroblock type as defined in the first two columns of MbType definition for Inter Macroblock (and MbSkipflag = 0). All macroblock types in a P Slice are mapped into the corresponding types in a B Slice. Skip and Direct modes are converted into its corresponding processing modes.<br>Programming note: It is exactly matched with that of DXVA 2.0. |
| | 7 | **FieldMbPolarityFlag**<br>This field indicates the field polarity of the current macroblock.<br>Within a MbAff frame picture, this field may be different per macroblock and is set to 1 only for the second macroblock in a MbAff pair if FieldMbFlag is set. Otherwise, it is set to 0.<br>Within a field picture, this field is set to 1 if the current picture is the bottom field picture. Otherwise, it is set to 0. It is a constant for the whole field picture.<br>This field is only valid for MBAFF frame picture. It is reserved and set to 0 for a progressive frame picture or a field picture.<br>0 = Current macroblock is a field macroblock from the **top** field (first in a MBAFF pair)<br>1 = Current macroblock is a field macroblock from the **bottom** field (second in a MBAFF pair) |
| | 6 | **IsLastMB**<br>1 – the current MB is the last MB in the current Slice<br>0 – the current MB is not the last MB in the current Slice |
| | 5-4 | Reserved MBZ (Intel encoder) |
| | 3:0 | Reserved MBZ (DXVA Decoder) |

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>1</td><td>31:16</td><td>
CbpY[bit 15:0] (Coded Block Pattern Y)<br><br>
For 4x4 sub-block (when Transform8x8flag = 0 or in intra16x16) :<br><br>
16-bit cbp, one bit for each 4x4 Luma sub-block (not including the DC 4x4 Lumablock in intra16x16) in a MB. The 4x4 Luma sub-blocks are numbered as
<table>
<tr><td>blk0</td><td>1</td><td>4</td><td>5</td><td>bit15</td><td>14</td><td>11</td><td>10</td></tr>
<tr><td>blk2</td><td>3</td><td>6</td><td>7</td><td>bit13</td><td>12</td><td>9</td><td>8</td></tr>
<tr><td>blk8</td><td>9</td><td>12</td><td>13</td><td>bit7</td><td>6</td><td>3</td><td>2</td></tr>
<tr><td>blk10</td><td>11</td><td>14</td><td>15</td><td>bit 5</td><td>4</td><td>1</td><td>0</td></tr>
</table>
The cbpY bit assignment is cbpY bit [15 – X] for sub-block_num X.<br><br>
For 8x8 block (when Transform8x8flag = 1)<br><br>
Only the lower 4 bits [3:0] are valid; the remaining upper bits [15:4] are ignored.The 8x8 Luma blocks are numbered as
<table>
<tr><td>blk0</td><td>1</td><td>bit3</td><td>2</td></tr>
<tr><td>blk2</td><td>3</td><td>bit1</td><td>0</td></tr>
</table>
The cbpY bit assignment is cbpY bit [3 – X] for block_num X.<br><br>
0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero)<br><br>
1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).
</td></tr>
<tr><td></td><td>15:8</td><td>
VertOrigin (Vertical Origin). This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks.<br><br>
For field macroblock pair in MBAFF frame, the vertical origins for both macroblocks should be set as if they were located in corresponding field pictures. For example, for field macroblock pair originated at (16, 64) pixel location in an MBAFF frame picture, the Vertical Origin for both macroblocks should be set as 2 (macroblocks). Whether the current macroblock is the first/second (top/bottom) in a MBAFF pair is specified by FieldMbPolarityFlag.<br><br>
The macroblocks with (VertOrigin, HorzOrigin) must be delivered in the strict order as coded in the bitstream (raster order for progressive frame or field pictures and MBAFF pair order for MBAFF pictures). No gap is allowed. Otherwise, hardware behavior is undefined.<br><br>
Format = U8 in unit of macroblock.
</td></tr>
<tr><td></td><td>7:0</td><td>
HorzOrigin (Horizontal Origin). This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks.<br><br>
Format = U8 in unit of macroblock.
</td></tr>
</table>

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>2</td><td>31:16</td><td>CbpCr (Coded Block Pattern Cr 4:2:0-only)<br>Only the lower 4 bits [3:0] are valid; the remaining upper bits [15:4] are ignored (only valid for 4:2:2 and 4:4:4). The 4x4 Chroma Cr sub-blocks are numbered as<br><table><tr><td>blk0</td><td>1</td><td>bit3</td><td>2</td></tr><tr><td>blk2</td><td>3</td><td>bit1</td><td>0</td></tr></table>The cbpCr bit assignment is cbpCr bit [3 – X] for sub-block_num X.<br>0 in a bit – indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero)<br>1 in a bit – indicates the corresponding 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).<br>For monochrome, this field is ignored.</td></tr>
<tr><td></td><td>15-0</td><td>CbpCb (Coded Block Pattern Cb 4:2:0-only)<br>Only the lower 4 bits [3:0] are valid; the remaining upper bits [15:4] are ignored (only valid for 4:2:2 and 4:4:4). The 4x4 Chroma Cb sub-blocks are numbered as<br><table><tr><td>blk0</td><td>1</td><td>bit3</td><td>2</td></tr><tr><td>blk2</td><td>3</td><td>bit1</td><td>0</td></tr></table>The cbpCb bit assignment is cbpCb bit [3 – X] for sub-block_num X.<br>0 in a bit – indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero)<br>1 in a bit – indicates the corresponding 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).<br>For monochrome, this field is ignored.</td></tr>
<tr><td>3</td><td>31:24</td><td>Reserved MBz</td></tr>
<tr><td></td><td>23:16</td><td>QpPrimeCr<br>Driver is responsible for deriving the QpPrimeCr from QpPrimeY.<br>For 8-bit pixel data, QpCr is the same as QpPrimeCr, and it takes on a value in the range of 0 to 51, positive integer.</td></tr>
<tr><td></td><td>15:8</td><td>QpPrimeCb<br>Driver is responsible for deriving the QpPrimeCb from QpPrimeY.<br>For 8-bit pixel data, QpCb is the same as QpPrimeCb, and it takes on a value in the range of 0 to 51, positive integer.</td></tr>
</table>

| DWord | Bit | Description |
|---|---|---|
| | 7:0 | **QpPrimeY**<br>This is the per-MB QP value specified for the current MB.<br>For 8-bit pixel data, QpY is the same as QpPrimeY, and it takes on a value in the range of 0 to 51, positive integer. |
| 4 to 6 | 31:0<br>Each | For intra macroblocks, definition of these fields are specified in Inline data subfields for an Intra Macroblock<br>For inter macroblocks, definition of these fields are specified in Inline data subfields for an Inter Macroblock |

## Indirect Data Format in AVC-IT Mode

Indirect data in AVC-IT mode consist of Motion Vectors, Transform-domain Residue (Coefficient) and ILDB control data. All three data records have variable size. Size of each Motion Vector record is determined by the MvQuantity value as shown in *Indirect Data Format in AVC-IT Mode*. ILDB control record is fixed at the same size for all MBs in a picture. Coefficient data record is variable size per MB, since it may only consist of non-zero coefficients.

Each MV is represented in 4 bytes, in the form of

- Lower 2 bytes : horizontal MVx component in q-pel units
- Upper 2 bytes : vertical MVy component in q-pel units
- Integer distance is measured in unit of samples in the frame or field grid position.
- Chroma MVs are not sent and are derived in the H/W.

## Indirect MV record size in AVC-IT mode

| Macroblock Type | MVQuant |
|---|---|
| *BP_L0_16x16* | 1 |
| B_L1_16x16 | 1 |
| B_Bi_16x16 | 2 |
| *BP_L0_L0_16x8* | 2 |
| *BP_L0_L0_8x16* | 2 |
| B_L1_L1_16x8 | 2 |
| B_L1_L1_8x16 | 2 |
| B_L0_L1_16x8 | 2 |
| B_L0_L1_8x16 | 2 |
| B_L1_L0_16x8 | 2 |
| B_L1_L0_8x16 | 2 |
| B_L0_Bi_16x8 | 3 |
| B_L0_Bi_8x16 | 3 |
| B_L1_Bi_16x8 | 3 |
| B_L1_Bi_8x16 | 3 |
| B_Bi_L0_16x8 | 3 |
| B_Bi_L0_8x16 | 3 |
| B_Bi_L1_16x8 | 3 |
| B_Bi_L1_8x16 | 3 |
| B_Bi_Bi_16x8 | 4 |
| B_Bi_Bi_8x16 | 4 |
| **BP_8x8** | Sum |

For macroblock type of BP_8x8, MvQuant takes the sum of value MvQ[i] of the four individual 8x8 sub macroblocks.

| SubMbShape[i] | SubMbPredMode[i] | Description | MvQ[i] |
|---|---|---|---|
| 0 | 0 | BP_L0_8x8 | 1 |
| 0 | 1 | B_L1_8x8 | 1 |
| 0 | 2 | B_BI_8x8 | 2 |
| 1 | 0 | BP_L0_8x4 | 2 |
| 1 | 1 | B_L1_8x4 | 2 |
| 1 | 2 | B_BI_8x4 | 4 |
| 2 | 0 | BP_L0_4x8 | 2 |
| 2 | 1 | B_L1_4x8 | 2 |
| 2 | 2 | B_BI_4x8 | 4 |
| 3 | 0 | BP_L0_4x4 | 4 |
| 3 | 1 | B_L1_4x4 | 4 |
| 3 | 2 | B_BI_4x4 | 8 |

### Indirect data Deblocking Filter Control block in AVC-IT mode:

AVC Deblocker Control Data record has a fixed size for each MB in a picture and is 48 bytes or 12 Dwords in size.

| DWord | Bit | Description |
|---|---|---|
| 0 | 31:24 | Reserved : MBZ (DXVA Decoder) |
| | 23 | **FilterTopMbEdgeFlag** |
| | 22 | **FilterLeftMbEdgeFlag** |
| | 21 | **FilterInternal4x4EdgesFlag** |
| | 20 | **FilterInternal8x8EdgesFlag** |
| | 19 | **FieldModeAboveMbFlag** |
| | 18 | **FieldModeLeftMbFlag** |
| | 17 | **FieldModeCurrentMbFlag** |
| | 16 | **MbaffFrameFlag** (DXVA Decoder reserved bit) |
| | 15:8 | **VertOrigin** Current MB y position (address) |
| | 7:0 | **HorzOrigin** Current MB x position (address) |

| DWord | Bit | Description |
|---|---|---|
| 1 | 31:30 | **bS_h13** 2-bit boundary strength for internal top horiz 4-pixel edge 3 |
| | 29:28 | **bS_h12** 2-bit boundary strength for internal top horiz 4-pixel edge 2 |
| | 27:26 | **bS_h11** 2-bit boundary strength for internal top horiz 4-pixel edge 1 |
| | 25:24 | **bS_h10** 2-bit boundary strength for internal top horiz 4-pixel edge 0 |
| | 23:22 | **bS_v33** 2-bit boundary strength for internal right vert 4-pixel edge 3 |
| | 21:20 | **bS_v23** 2-bit boundary strength for internal right vert 4-pixel edge 2 |
| | 19:18 | **bS_v13** 2-bit boundary strength for internal right vert 4-pixel edge 1 |
| | 17:16 | **bS_v03** 2-bit boundary strength for internal right vert 4-pixel edge 0 |
| | 15:14 | **bS_v32** 2-bit boundary strength for internal mid vert 4-pixel edge 3 |
| | 13:12 | **bS_v22** 2-bit boundary strength for internal mid vert 4-pixel edge 2 |
| | 11:10 | **bS_v12** 2-bit boundary strength for internal mid vert 4-pixel edge 1 |
| | 9:8 | **bS_v02** 2-bit boundary strength for internal mid vert 4-pixel edge 0 |
| | 7:6 | **bS_v31** 2-bit boundary strength for internal left vert 4-pixel edge 3 |
| | 5:4 | **bS_v21** 2-bit boundary strength for internal left vert 4-pixel edge 2 |
| | 3:2 | **bS_v11** 2-bit boundary strength for internal left vert 4-pixel edge 1 |
| | 1:0 | **bS_v01** 2-bit boundary strength for internal left vert 4-pixel edge 0 |
| 2 | 31:28 | **bS_v30_0** 4-bit boundary strength for Left0 4-pixel edge 3 (MSbit is wasted) |
| | 17:24 | **bS_v20_0** 4-bit boundary strength for Left0 4-pixel edge 2 (MSbit is wasted) |
| | 23:20 | **bS_v10_0** 4-bit boundary strength for Left0 4-pixel edge 1 (MSbit is wasted) |
| | 19:16 | **bS_v00_0** 4-bit boundary strength for Left0 4-pixel edge 0 (MSbit is wasted) |
| | 15:14 | **bS_h33** 2-bit boundary strength for internal bot horiz 4-pixel edge 3 |

| DWord | Bit | Description |
|---|---|---|
| | 13:12 | **bS_h32** 2-bit boundary strength for internal bot horiz 4-pixel edge 2 |
| | 11:10 | **bS_h31** 2-bit boundary strength for internal bot horiz 4-pixel edge 1 |
| | 9:8 | **bS_h30** 2-bit boundary strength for internal bot horiz 4-pixel edge 0 |
| | 7:6 | **bS_h23** 2-bit boundary strength for internal mid horiz 4-pixel edge 3 |
| | 5:4 | **bS_h22** 2-bit boundary strength for internal mid horiz 4-pixel edge 2 |
| | 3:2 | **bS_h21** 2-bit boundary strength for internal mid horiz 4-pixel edge 1 |
| | 1:0 | **bS_h20** 2-bit boundary strength for internal mid horiz 4-pixel edge 0 |
| 3 | 31:28 | **bS_h03_0** 4-bit boundary strength for Top0 4-pixel edge 3 (MSbit is wasted) |
| | 27:24 | **bS_h02_0** 4-bit boundary strength for Top0 4-pixel edge 2 (MSbit is wasted) |
| | 23:20 | **bS_h01_0** 4-bit boundary strength for Top0 4-pixel edge 1 (MSbit is wasted) |
| | 19:16 | **bS_h00_0** 4-bit boundary strength for Top0 4-pixel edge 0 (MSbit is wasted) |
| | 15:12 | **bS_v03** 4-bit boundary strength for Left1 4-pixel edge 3 (MSbit is wasted) |
| | 11:8 | **bS_v02** 4-bit boundary strength for Left1 4-pixel edge 2 (MSbit is wasted) |
| | 7:4 | **bS_v01** 4-bit boundary strength for Left1 4-pixel edge 1 (MSbit is wasted) |
| | 3:0 | **bS_v00** 4-bit boundary strength for Left1 4-pixel edge 0 (MSbit is wasted) |
| 4 | 31:24 | **bIndexBinternal_Y** Internal index B for Y |
| | 23:16 | **bIndexAinternal_Y** Internal index A for Y |
| | 15:12 | **bS_h03_1** 4-bit boundary strength for Top1 4-pixel edge 3 (MSbit is wasted) |
| | 11:8 | **bS_h02_1** 4-bit boundary strength for Top1 4-pixel edge 2 (MSbit is wasted) |
| | 7:4 | **bS_h01_1** 4-bit boundary strength for Top1 4-pixel edge 1 (MSbit is wasted) |
| | 3:0 | **bS_h00_1** 4-bit boundary strength for Top1 4-pixel edge 0 (MSbit is wasted) |

| DWord | Bit | Description |
|---|---|---|
| 5 | 31:24 | **bIndexBleft1_Y** |
| | 23:16 | **bIndexAleft1_Y** |
| | 15:8 | **bIndexBleft0_Y** |
| | 7:0 | **bIndexAleft0_Y** |
| 6 | 31:24 | **bIndexBtop1_Y** |
| | 23:16 | **bIndexAtop1_Y** |
| | 15:8 | **bIndexBtop0_Y** |
| | 7:0 | **bIndexAtop0_Y** |
| 7 | 31:24 | **bIndexBleft0_Cb** |
| | 23:16 | **bIndexAleft0_Cb** |
| | 15:8 | **bIndexBinternal_Cb** |
| | 7:0 | **bIndexAinternal_Cb** |
| 8 | 31:24 | **bIndexBtop0_Cb** |
| | 23:16 | **bIndexAtop0_Cb** |
| | 15:8 | **bIndexBleft1_Cb** |
| | 7:0 | **bIndexAleft1_Cb** |
| 9 | 31:24 | **bIndexBinternal_Cr** |
| | 23:16 | **bIndexAinternal_Cr** |
| | 15:8 | **bIndexBtop1_Cb** |
| | 7:0 | **bIndexAtop1_Cb** |
| 10 | 31:24 | **bIndexBleft1_Cr** |

| DWord | Bit | Description |
|---|---|---|
| | 23:16 | **bIndexAleft1_Cr** |
| | 15:8 | **bIndexBleft0_Cr** |
| | 7:0 | **bIndexAleft0_Cr** |
| 11 | 31:24 | **bIndexBtop1_Cr** |
| | 23:16 | **bIndexAtop1_Cr** |
| | 15:8 | **bIndexBtop0_Cr** |
| | 7:0 | **bIndexAtop0_Cr** |

## Inline Data Description in VC1-IT Mode

| DWord | Bits | Description |
|---|---|---|
| +0 | 31:28 | **MvFieldSelect.** A bit-wise representation indicating which field in the reference frame is used as the reference field for current field. It's only used in decoding interlaced pictures.<br>This field is valid for non-intra macroblock only.<br><br>**Bit** / **Description**<br>28 – Forward predict of current frame/field or TOP field of interlace frame, or block 0 in 4MV mode.<br>29 – Backward predict of current frame/field or TOP field of interlace frame, or forward predict for block 1 in 4MV mode.<br>30 – Forward predict of BOTTOM field of interlace frame, or block 2 in 4MV mode.<br>31 – Backward predict of BOTTOM field of interlace frame, or forward predict for block 3 in 4MV mode.<br><br>Each corresponding bit has the following indication.<br>0 = The prediction is taken from the top reference field.<br>1 = The prediction is taken from the bottom reference field. |
| | 27 | **Reserved.** MBZ |
| | 26 | **MvFieldSelectChroma .** This field specifies the polarity of reference field for chroma blocks when their motion vector is derived in **Motion4MV** mode for interlaced (field) picture.<br>Non-intra macroblock only. This field is derived from MvFieldSelect. |

| DWord | Bits | Description |
|---|---|---|
| | | 0 = The prediction is taken from the top reference field.<br>1 = The prediction is taken from the bottom reference field. |
| | 25:24 | **MotionType – Motion Type**<br>For frame picture, a macroblock may only be either 00 or 10.<br>For interlace picture, a macroblock may be of any motion types. It can be 01 if and only if DctType is 1.<br>This field is 00 if and only if IntraMacroblock is 1.<br>00 = Intra<br>01 = Field Motion.<br>10 = Frame Motion or no motion.<br>Others = Reserved. |
| | 23 | **Reserved.** MBZ |
| | 22 | **MvSwitch.** This field specifies whether the prediction needs to be switched from forward to backward or vice versa for single directional prediction for top and bottom fields of interlace frame B macroblocks.<br>0 = No directional prediction switch from top field to bottom field<br>1 = Switch directional prediction from top field to bottom field |
| | 21 | **DctType.** This field specifies whether the residual data is coded as field residual or frame residual for interlaced picture. This field can be 1 only if MotionType is 00 (intra) or 01 (field motion).<br>For progressive picture, this field must be set to '0', i.e. all macrobalcoks are frame macroblock.<br>0 = Frame residual type.<br>1 = Field residual type. |
| | 20 | **OverlapTransform.** This field indicates whether overlap smoothing filter should be performed on I-block boundaries.<br>0 = No overlap smoothing filter.<br>1 = Overlap smoothing filter performed. |
| | 19 | **Motion4MV.** This field indicates whether current macroblock a progressive P picture uses 4 motion vectors, one for each luminance block.<br>It's only valid for progressive P-picture decoding. Otherwise, it is reserved and MBZ. For example, with MotionForward is 0, this field must also be set to 0.<br>0 = 1MV-mode. |

| DWord | Bits | Description |
|---|---|---|
| | | 1 = 4MV-mode. |
| | 18 | **MotionBackward.** This field specifies whether the backward motion vector is active for B-picture. This field must be 0 if Motion4MV is 1 (no backward motion vector in 4MV-mode).<br>0 = No backward motion vector.<br>1 = Use backward motion vector(s). |
| | 17 | **MotionForward.** This field specifies whether the forward motion vector is active for P and B pictures.<br>0 = No forward motion vector.<br>1 = Use forward motion vector(s). |
| | 16 | **IntraMacroblock.** This field specifies if the current macroblock is intra-coded. When set, Coded Block Pattern is ignored and no prediction is performed (i.e., no motion vectors are used).<br>For field motion, this field indicates whether the top field of the macroblock is coded as intra.<br>0 = Non-intra macroblock.<br>1 = Intra macroblock. |
| | 15:12 | **LumaIntra8x8Flag – Luma Intra 8x8 Flag**<br>This field specifies whether each of the four 8x8 luminance blocks are intra or inter coded when Motion4MV is set to 4MV-Mode.<br>Each bit corresponds to one block. “0” indicates the block is inter coded and ‘1’ indicates the block is intra coded.<br>When Motion4MV is not 4MV-Mode, this field is reserved and MBZ.<br>Bit 15: Y0<br>Bit 14: Y1<br>Bit 13: Y2<br>Bit 12: Y3 |
| | 11:6 | **CBP - Coded Block Pattern**<br>This field specifies whether the 8x8 residue blocks in the macroblock are present or not.<br>Each bit corresponds to one block. “0” indicates residue block isn’t present, “1” indicates residue block is present.<br>Note: For each block in an intra-coded macroblock or an intra-coded block in a P macroblock in 4MV-Mode, the corresponding CBP must be 1. Subsequently, there must be at least one coefficient (this coefficient might be zero) in the indirect data buffer associated with the bock (i.e. residue block must be present). |

| DWord | Bits | Description |
|---|---|---|
| | | Bit 11: Y0<br>Bit 10: Y1<br>Bit 9: Y2<br>Bit 8: Y3<br>Bit 7: Cb4<br>Bit 6: Cr5 |
| | 5 | **ChromaIntraFlag - Derived Chroma Intra Flag**<br>This field specifies whether the chroma blocks should be treated as intra blocks based on motion vector derivation process in 4MV mode.<br>0 = Chroma blocks are not coded as intra.<br>1 = Chroma blocks are coded as intra |
| | 4 | **LastRowFlag – Last Row Flag**<br>This field indicates that the current macroblock belongs to the last row of the picture.<br>This field may be used by the kernel to manage pixel output when overlap transform is on.<br>0 = Not in the last row<br>1 = In the last row |
| | 3 | **LastMBInRow – This field indicates the last MB in row flag.** |
| | 2:0 | **Reserved.** MBZ |
| +1 | 32:26 | **Reserved.** MBZ |
| | 25:24 | **OSEdgeMaskChroma**<br>This field contains the overscan edge mask for the Chroma blocks.<br>The left edge masks are hardware and the top edge masks are used by the kernel software.<br>Bit 24: Top edge of block Cb/Cr<br>Bit 25: Left edge of block Cb/Cr |
| | 23:16 | **OSEdgeMaskLuma**<br>This field contains the overscan edge mask for the Luma blocks.<br>The left edge masks are hardware and the top edge masks are used by the kernel software.<br>Bit 16: Top edge of block Y0<br>Bit 17: Top edge of block Y1 |

| DWord | Bits | Description |
|---|---|---|
| | | Bit 18: Top edge of block Y2<br>Bit 19: Top edge of block Y3<br>Bit 20: Left edge of block Y0<br>Bit 21: Left edge of block Y1<br>Bit 22: Left edge of block Y2<br>Bit 23: Left edge of block Y3<br>*Programming Note: In order to create 8 predication bits from each edge mask bit, software may first create a 0, 1 vector by using a shr instruction with a step shift vector like 0, 1, 2, 3 (e.g. using immediate of type :v. Then each 0 or 1 of the LSB can be repeated by an and instruction to create 8 bits to the flag register. Alternatively, this can be achieved with one and instruction using a CURBE constant map of bit 0 and bit 1 mask.* |
| | 15:8 | **VertOrigin - Vertical Origin**<br>In unit of macroblocks relative to the current picture (frame or field). |
| | 7:0 | **HorzOrigin - Horizontal Origin**<br>In unit of macroblocks. |
| +2 | 31:16 | **MotionVector[0].Vert** |
| | 15:0 | **MotionVector[0].Horz** |
| +3 | 31:0 | **MotionVector[1]** |
| +4 | 31:0 | **MotionVector[2]** |
| +5 | 31:0 | **MotionVector[3]** |
| +6 | 31:0 | **MotionVectorChroma**<br>This field is not valid for a field motion in an interlaced frame picture where 4 MVs for chroma blocks.<br>Notes: This field is derived from MotionVector[3:0] as described in the following section. |
| +7 | 31:24 | **Subblock Code for Y3**<br>The following subblock coding definition applies to all 6 subblock coding bytes. Bits 7:6 are reserved. |

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th colspan="2">Subblock Partitioning<br>(Bits [1:0])<br>Specify Transform uses for an 8x8 block</th><th colspan="4">Subblock Present<br>(0 means not present, 1 means present)</th></tr>
<tr><th>Bits [1:0]</th><th>Meaning</th><th>Bit 2</th><th>Bit 3</th><th>Bit 4</th><th>Bit 5</th></tr>
<tr><td>00</td><td>Single 8x8 block (sb0)</td><td>Sb0</td><td>Don’t care</td><td>Don’t care</td><td>Don’t care</td></tr>
<tr><td>01</td><td>Two 8x4 subblocks (sb0-1)</td><td>Sb1 (bot)</td><td>Sb0 (top)</td><td>Don’t care</td><td>Don’t care</td></tr>
<tr><td>10</td><td>Two 4x8 subblocks (sb0-1)</td><td>Sb1 (right)</td><td>Sb0 (left)</td><td>Don’t care</td><td>Don’t care</td></tr>
<tr><td>11</td><td>Four 4x4 subblocks (sb0-3)</td><td>Sb3 (lower right)</td><td>Sb2 (lower left)</td><td>Sb1 (upper right)</td><td>Sb0 (upper left)</td></tr>
</table>
</td></tr>
<tr><td></td><td>23:16</td><td>Subblock Code for Y2</td></tr>
<tr><td></td><td>15:8</td><td>Subblock Code for Y1</td></tr>
<tr><td></td><td>7:0</td><td>Subblock Code for Y0</td></tr>
<tr><td>+8</td><td>31:16</td><td>Reserved. MBZ</td></tr>
<tr><td></td><td>15:8</td><td>Subblock Code for Cr</td></tr>
<tr><td></td><td>7:0</td><td>Subblock Code for Cb</td></tr>
<tr><td>+9</td><td>31:24</td><td>ILDB control data for block Y3</td></tr>
<tr><td></td><td>23:16</td><td>ILDB control data for block Y2</td></tr>
<tr><td></td><td>15:8</td><td>ILDB control data for block Y1</td></tr>
<tr><td></td><td>7:0</td><td>ILDB control data for block Y0</td></tr>
<tr><td>+10</td><td>31:16</td><td>Reserved</td></tr>
<tr><td></td><td>15:8</td><td>ILDB control data for Cr block</td></tr>
<tr><td></td><td>7:0</td><td>ILDB control data for Cb block</td></tr>
</table>

## Indirect Data Format in VC1-IT Mode

VC1-IT mode only contains IT-COEFF indirect data which is described in *Common Indirect IT-COEFF Data Structure*.

## Inline Data Description in MPEG2-IT Mode

The content in this command is similar to that in the MEDIA_OBJECT command in IS mode described in the Media Chapter.

Each MFD_IT_OBJECT command corresponds to the processing of one macroblock. Macroblock parameters are passed in as inline data and the non-zero DCT coefficient data for the macroblock is passed in as indirect data.

*Inline Data Description in MPEG2-IT Mode* depicts the inline data format. Inline data starts at dword 7 of MFD_IT_OBJECT command. There are 7 dwords total.

### Inline data in MPEG2-IT Mode

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>+0</td><td>31:28</td><td>Motion Vertical Field Select. A bit-wise representation of a long [2][2] array as defined in §6.3.17.2 of the ISO/IEC 13818-2 (see also §7.6.4).
<table>
<tr><th>Bit</th><th>MVector[r]</th><th>MVector[s]</th><th>MotionVerticalFieldSelect Index</th></tr>
<tr><td>28</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>29</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>30</td><td>1</td><td>0</td><td>2</td></tr>
<tr><td>31</td><td>1</td><td>1</td><td>3</td></tr>
</table>
Format = MC_MotionVerticalFieldSelect.<br>0 = The prediction is taken from the top reference field.<br>1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>27</td><td>Reserved (was Second Field)</td></tr>
<tr><td></td><td>26</td><td>Reserved. (HWMC mode)</td></tr>
<tr><td></td><td>25:24</td><td>Motion Type. When combined with the destination picture type (field or frame) this Motion Type field indicates the type of motion to be applied to the macroblock. See ISO/IEC 13818-2 §6.3.17.1, Tables 6-17, 6-18. In particular, the device supports dual-prime motion prediction (11) in both frame and field picture type.<br>Format = MC_MotionType</td></tr>
</table>

| DWord | Bit | Description |
|---|---|---|
| | | <table><tr><th>Value</th><th>Destination = Frame<br>Picture_Structure = 11</th><th>Destination = Field<br>Picture_Structure != 11</th></tr><tr><td>‘00’</td><td>Reserved</td><td>Reserved</td></tr><tr><td>‘01’</td><td>Field</td><td>Field</td></tr><tr><td>‘10’</td><td>Frame</td><td>16x8</td></tr><tr><td>‘11’</td><td>Dual-Prime</td><td>Dual-Prime</td></tr></table> |
| | 23:22 | Reserved. (Scan method) |
| | 21 | **DCT Type.** This field specifies the DCT type of the current macroblock. The kernel should ignore this field when processing Cb/Cr data. See *ISO/IEC 13818-2* §6.3.17.1. This field is zero if Coded Block Pattern is also zero (no coded blocks present).<br>0 = MC_FRAME_DCT (Macroblock is frame DCT coded).<br>1 = MC_FIELD_DCT (Macroblock is field DCT coded). |
| | 20 | Reserved (was Overlap Transform - H261 Loop Filter). |
| | 19 | Reserved (was 4MV Mode - H263) |
| | 18 | **Macroblock Motion Backward.** This field specifies if the backward motion vector is active. See *ISO/IEC 13818-2* Tables B-2 through B-4.<br>0 = No backward motion vector.<br>1 = Use backward motion vector(s). |
| | 17 | **Macroblock Motion Forward.** This field specifies if the forward motion vector is active. See *ISO/IEC 13818-2* Tables B-2 through B-4.<br>0 = No forward motion vector.<br>1 = Use forward motion vector(s). |
| | 16 | **Macroblock Intra Type.** This field specifies if the current macroblock is intra-coded. When set, Coded Block Pattern is ignored and no prediction is performed (i.e., no motion vectors are used). See *ISO/IEC 13818-2* Tables B-2 through B-4.<br>0 = Non-intra macroblock.<br>1 = Intra macroblock. |
| | 15:12 | **Reserved : MBZ** |
| | 11:6 | **Coded Block Pattern.** This field specifies whether blocks are present or not.<br>Format = 6-bit mask.<br>Bit 11: Y0 |

| DWord | Bit | Description |
|---|---|---|
| | | Bit 10: Y1<br>Bit 9: Y2<br>Bit 8: Y3<br>Bit 7: Cb4<br>Bit 6: Cr5 |
| | 5:4 | Reserved. (Quantization Scale Code) |
| | 3 | LastMBInRow – This field indicates the last MB in each row. |
| | 2:0 | Reserved: MBZ |
| +1 | 31:16 | **Reserved : MBZ** |
| | 15:8 | **VertOrigin - Vertical Origin**<br>In unit of macroblocks relative to the current picture (frame or field). |
| | 7:0 | **HorzOrigin - Horizontal Origin**<br>In unit of macroblocks. |
| +2 | 31:16 | **Motion Vectors – Field 0, Forward, Vertical Component.** Each vector component is a 16-bit two's-complement value. The vector is relative to the current macroblock location. According to ISO/IEC 13818-2 Table 8, the valid range of each vector component is [-2048, +2047.5], implying a format of s11.1. However, it should be noted that motion vector values are sign extended to 16 bits. |
| | 15:0 | **Motion Vectors – Field 0, Forward, Horizontal Component** |
| +3 | 31:16 | **Motion Vectors – Field 0, Backward, Vertical Component** |
| | 15:0 | **Motion Vectors – Field 0, Backward, Horizontal Component** |
| +4 | 31:16 | **Motion Vectors – Field 1, Forward, Vertical Component** |
| | 15:0 | **Motion Vectors – Field 1, Forward, Horizontal Component** |
| +5 | 31:16 | **Motion Vectors – Field 1, Backward, Vertical Component** |
| | 15:0 | **Motion Vectors – Field 1, Backward, Horizontal Component** |

### Indirect Data Format in MPEG2-IT Mode

MPEG2-IT mode only contains IT-COEFF indirect data which is described in Section *Common Indirect IT-COEFF Data Structure*.

### MFX Deblocking Commands

Following are MFX Deblocking Commands:

**MFX_DBK_OBJECT**

## Encoder StreamOut Mode Data Structure Definition

When StreamOut is enabled, per MB (and/or per Slice, per Picture) intermediated coding data (for example, bit allocated for each MB, and so on) are sent to the memory in a fixed record format (and of fixed size) from the PAK. The per-MB records must be written in a strict raster order and with no gap (that is, every MB regardless of its mb_type and slice type, must have an entry in the StreamOut buffer). Therefore, the consumer of the StreamOut data can offset into the StreamOut Buffer (**StreamOut Data Destination Base Address**) using individual MB addresses.

Adding per macroblock stream out for PAK is for the following purposes:

- Immediate multi-pass PAK (without host or EU intervention)
  - 3200-bit conformance
  - Re-quantization
- Providing information for host for offline processing
- Providing information for updated QP's

The description for the fixed format PAK streamout record:

Streamout Pointer: Use the existing streamout pointer and enabler

Per Macroblock Information (a fixed size structure)

| DWord | Bit | Description |
|---|---|---|
| 0 | 31:24 | **MbQpY** - Actual QPY used by the macroblock. |
| | 23:16 | **[BDW] MbClock16** – MB compute clocks in 16-clock unit. |
| | 15:8 | Reserved: MBZ |
| | 7:4 | Reserved: MBZ (future conformance flags) |
| | 3 | Reserved |
| | 2 | **MbRcFlag:** MB level Rate control flag(pass through)<br>The same value as **RateControlCounterEnable** of MFX_AVC_SLICE_STATE Command |
| | 1 | **MbInterConfFlag:** MB level InterMB conformance flag to trigger mutli-pass<br>1- if total Bit Count of an inter macroblock is more than Inter Conformance Max size limit in the MFX_AVC_IMG_STATE Command |
| | 0 | **MbIntraConfFlag:** MB level IntraMB conformance flag to trigger mutli-pass<br>1- if total Bit Count of an intra macroblock is more than Intra Conformance Max size limit in the |

| DWord | Bit | Description |
|---|---|---|
| | | MFX_AVC_IMG_STATE Command |
| 1 | 31:29 | Reserved |
| | 28:16 | **MbBits:** Total Bit Count for the macroblock |
| | 15:12 | Reserved |
| | 12:0 | **MbHdrBits:** Header Bit count (bit count due to Pre-coefficient data) for the macroblock |
| 2 | 31:27 | Reserved |
| | 26:0 | **Cbp:** Coded Block Pattern of sub-blocks |
| 3 | 31:30 | Reserved |
| | 29 | **IntraMBFlag** |
| | 28:24 | **MBType5Bits** |
| | 23:17 | Reserved |
| | 16 | **ClampFlag:** Coefficient clamping flag for RC (Status)<br>1 - Indicates if clamping of any coefficient is done on the macroblock for Rate Control |
| | 15:0 | Reserved (future QRC stat output) |

# PAK Multi-Pass

**Multi-Pass PAK Usages:**

- Intra MB 3200-bit conformance
- Inter MB Re-quantization
- Frame level Re-quantization

**How to Enable Multi-Pass PAK?**

- Using the existing conditional batch buffer execution capability to skip/execute the second pass
    - How to dynamically change the condition?
        - Defined one error condition register with a mask. Do HW status page update at the end of the first pass. 0 means all OK, non-zero means there is an error condition, requiring second pass. Mask is used by the host to control what kind of multi-pass is intended.
        - For example, one error bit is 3200-bit conformance violation. Another error bit is the total bit count exceeds (too much or too little) the target range (need to define the target range in the state).
        - **The logic perfectly fits in the conditional batch buffer control logic that VCS has today in GT. There is no additional logic need to be added in VCS to support media functionality.** (**Batch Buffer Skip:** This field only takes effect if Compare Semaphore is set and the value at **Semaphore Address is NOT greater than the Semaphore Data Dword)**.
- Adding a picture level state command to enable and control the behavior of the second pass PAK
    - How to control the re-PAK? Added 3 conformance flags (error registers) in the per-MB streamout. Then the error control is based on the error register and the mask defined in picture level states. There are 8 register flags defined out of which only the 3200-bit case has usage model defined for today. The rest are left for future usage.

**Issues and Limitations:**

- There is no programmable engine in MFX for flexible control: Therefore, whatever we have defined must consider flexibility

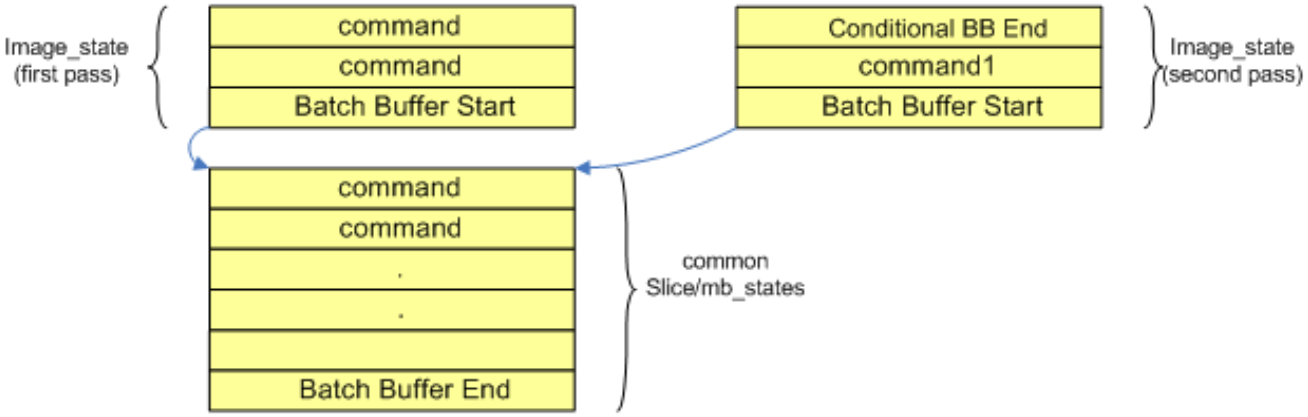
Following 2 MI packets are used inside VCS without any change to support Multipass-PAK behaviour.

- MI_Conditional_Batch_Buffer_End
- Memory Interface Registers

## Driver Usage

Driver places Image states in one batch buffer and all slice level and macroblock level states into another batch buffer and link 2 batch buffers. Also replicate Image states with multipass changes in another batch buffer link them to slice/macroblock batch buffer. In this way, only Image states are replicated but not the slice/macroblock states. The image states includes all buffers defined at image(indirectMV, original pixel buffer, etc). Following changes are needed in the Multipass Image State

- **Reset- Stream-Out Enable(disable stream out in the second pass)**
- **Set- MacroblockStatEnable (enable reading of macroblock status buffer)**
- **Reset- 3200-bit conformance (do not report 3200-bit conformance)**




Define Conditional Batch Buffer End for CS/VCSVINunit

# Programming Reference

## Monochrome Picture Processing

Monochrome picture is specified using the Surface State with Surface Format of 12. Therefore, MFX hardware, in either decode or encode mode, does not generate any read or write traffic for U/V components. The motivation for this bandwidth optimization is that monochrome video coding might be used for wireless display.

For Encoder:

1. No read in UV original components
2. Processing UV component - no
3. Reconstructed UV component reference picture - no
4. Filter UV component - no

For Decoder:

1. VLD mode: There is no color component coming out of the decoding pipeline in Monochrome mode and so no processing and not writing output.
2. IT mode: There is no color component in the coefficient buffer, and so no processing and not writing output.

## Context Switch

There is no pre-emption for the BCS pipeline; hence every command buffer is required to contain all the states setup (preamble). Specifically, CPU can not interrupt the BCS-BSD pipe, to stop the operation in the middle of decoding a bitstream data.

Switch of contexts can only be performed at picture boundary.

No state needs to be saved.

## PMSI Support

## Pipeline Flush

Implicit flush for AVC and VC1 is performed at the end of Slice : for MPEG2 is done when a new image/picture command is issued. Because MPEG2 a slice can be one MB, no point to flush. MPEG2 will snoop the next command if it is an img_state command.

Explicit flush MI (1 bit to do media pipeline vs Gx pipeline) flush and cache flush (switch reference frame) – MI flush has bit to do cache flush. MI flush is for driver synchronization.

## MMIO Interface

A set of registers are defined and accessible through MMIO interface to serve multiple purposes:

- Use for system configuration
- For accessing Performance counters

The following is the table for all the MMIO addresses for MFX.

### Decoder Registers

Following are Decoder Registers:

**MFD_ERROR_STATUS - MFD Error Status**

**AVC CAVLC**

**AVC CABAC**

**VC1**

**MPEG2**

**JPEG**

**MFD_PICTURE_PARAM - MFD Picture Parameter**

**MFX_STATUS_FLAGS - MFX Pipeline Status Flags**

**MFX_FRAME_PERFORMANCE_CT - MFX Frame Performance Count**

**MFX_SLICE_PERFORM_CT - MFX Slice Performance Count**

**MFX_MB_COUNT - MFX Frame Macroblock Count**

**MFX_SE-BIN_CT - MFX Frame BitStream SE/BIN Count**

**MFX_LAT_CT1 - MFX_Memory_Latency_Count1**

**MFX_LAT_CT2 - MFX Memory Latency Count2**

**MFX_LAT_CT3 - MFX Memory Latency Count3**

**MFX_LAT_CT4 - MFX Memory Latency Count4**

**MFX_SE-BIN_CT - MFX Frame BitStream SE/BIN Count**

**MFX_READ_CT - MFX Frame Motion Comp Read Count**

**MFX_MISS_CT - MFX Frame Motion Comp Miss Count**

## Encoder Registers

Following are the Encoder Registers:

**MFC_VIN_AVD_ERROR_CNTR - MFC_AVC Bitstream Decoding Front-End Parsing Logic Error Counter**

**MFC_BITSTREAM_BYTECOUNT_FRAME - Reported Bitstream Output Byte Count per Frame Register**

**MFC_BITSTREAM_SE_BITCOUNT_FRAME - Reported Bitstream Output Bit Count for Syntax Elements Only Register**

**MFC_AVC_CABAC_BIN_COUNT_FRAME - Reported Bitstream Output CABAC Bin Count Register**

**AVC_CABAC_INSERTION_COUNT - MFC_AVC_CABAC_INSERTION_COUNT**

**MFC_AVC_MINSIZE_PADDING_COUNT - Bitstream Output Minimal Size Padding Count Report Register**

**MFC_IMAGE_STATUS_MASK - MFC Image Status Mask**

**MFC_IMAGE_STATUS_CONTROL - MFC Image Status Control**

**MFC_QUP_CT - MFC QP Status Count**

**MFC_BITSTREAM_BYTECOUNT_SLICE - Bitstream Output Byte Count Per Slice Report Register**

**MFC_BITSTREAM_SE_BITCOUNT_SLICE - Bitstream Output Bit Count for the last Syntax Element Report Register**

**MFX_PAK_ERROR Register**

**MFX_PAK_WARNING Register**

## Row Store Sizes and Allocations

| | AVC | VC1 | MPEG2 | JPEG | IT | ENC | SEC ENC |
|---|---|---|---|---|---|---|---|
| vin_vmx_pixcoefind_ addr[31:6] | Bitstream | Bitstream | Bitstream | Bitstream | VDS COEF | Orig Pix | BSP data |
| vin_vmx_mvbsdrs_ addr[31:6] | VAD BSD | | VMD RS | | VDS MV | MPC MV | |
| vin_vmx_mpcildbmpr_ addr[31:6] | VAM MPR | | | | VDS ILDB | MPC RS | |
| vin_vmx_dmv*_ addr[31:6] | VAM DMV | VCP DMV | | | | | |
| vin_vmx_bp_addr [31:0] | | VCP BP | | | | | |

MPEG2 VLD Decoding Mode :

use BSD Row Store only, and

MPEG2 IT Decoding Mode :

MPEG2 IT mode does not need row-store

JPEG VLD Decoding Mode : no row store is needed