Intel® Iris® Xᵉ MAX Graphics Open Source

Programmer's Reference Manual

For the 2020 Discrete GPU formerly named "DG1"

Volume 11: Media Engines

February 2021, Revision 1.0

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

Customer is responsible for safety of the overall system, including compliance with applicable safety-related requirements or standards.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described 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.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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.

Table of Contents

Media Engines

Media VDBOX

This chapter describes the VDBOX Media Engine.

AVP

The AV1 Codec Pipeline (AVP) is a fixed function hardware video codec responsible for decoding AV1 (AOMedia Video 1) video streams.

AVP Register Definitions

The Message Channel Interface is a read-only bus used to access the AVP status registers. All registers are 32 bits where reserved bits return a value of zero and subtractive-decode is used to return 0x0000 for all register holes.

Register Attributes Description

Host Register Attributes gives the defined register tags and their description.

Host Register Attributes

Tag	Name	Description
R/W	Read/Write	Bit is read and writeable.
R/SW	Read/Special Write	Bit is readable. Write is only allowed once after a reset.
RO	Read Only	Bit is only readable, but writes have no effects.
WO	Write Only	Bit is only writeable, reads return zeros.
RV	Reserved	Bit is reserved and not visible. Reads will return 0, and writes have no effect.
NA	Not Accessible	This bit is not accessible.

AVP Decoder Register Map

This documents all AVP Decoder MMIO Registers.

AVP Decoder Register Descriptions

Reserved.

AVP Command Summary

The AV1 is configured through a set of batch commands defined in the following sections. The software driver builds a frame level workload using these commands and stores these workloads in graphics memory where they are fetched by the Video Command Streamer (VCS) and presented to the AVP for processing. The commands are processed by the Workload Parser within the AVP and the hardware is

configured by the Workload Parser prior to each frame level encode or decode. A workload is defined as a set of commands necessary to encode or decode one frame.

The software driver is required to read the AVP disable fuse to determine if the AVP is enabled. If it is disabled, then the software driver must not enable AVP batch commands to be sent to the AVP or a hang event may occur. Only when the AVP is enabled through the fuse, should the batch commands be sent to the AVP.

AVP Workload Command Model

DWord0 of each command is defined in AVP DWord0 Command Definition. The AVP is selected with the **Media Instruction Opcode "8h**" for all AVP Commands.

HCP DWord0 Command Definition

DWord	Bits	Description
0	31:29	**Command Type** = PARALLEL_VIDEO_PIPE = 3h
	28:27	**Pipeline Type** = 2h
	26:23	**Media Instruction Opcode = Codec/Engine Name** = AVP = 8h
	22:16	**Media Instruction Command** = <*see* HCP Media Instruction Commands (Opcode=7h)>
	15:12	**Reserved: MBZ**
	11:0	**Dword Length** (Excludes Dwords 0, 1) = <*command length*>

Each AVP command has assigned a media instruction command as defined in AVP Media Instruction Commands (Opcode=8h).

AVP Media Instruction Commands (Opcode=8h)

Media Instruction Command	Command DWord0 [22:16]	Mode	Scope
AVP_PIPE_MODE_SELECT	0h	Dec	Picture
AVP_SURFACE_STATE	1h	Dec	Picture
AVP_PIPE_BUF_ADDR_STATE	2h	Dec	Picture
AVP_IND_OBJ_BASE_ADDR_STATE	3h	Dec	Picture
Reserved	4h-5h		
Reserved	8h-9h		

Media Instruction Command	Command DWord0 [22:16]	Mode	Scope
VD_CONTROL_STATE	Ah	Dec	Picture
Reserved	Bh-Fh		
AVP_PIC_STATE	10h	Dec	Picture
Reserved	11h		
AVP_REF_IDX_STATE	12h	Dec	Tile
Reserved	13h-14h		
AVP_TILE_CODING	15h	Dec	Tile
Reserved	16h-1Fh		
AVP_BSD_OBJECT_STATE	20h	Dec	Tile
Reserved	21h-31h		
Reserved	33h-7Fh		

AVP Command Sequence

The AV1 is configured for encoding or decoding through a set of batch commands defined in the following sections. The software driver builds a frame level workload using these commands and stores these workloads in graphics memory where they are fetched by the Video Command Streamer (VCS) and presented to the AVP for processing. The commands are processed by the Workload Parser within the AVP and the hardware is configured by the Workload Parser prior to each frame level encode or decode. A workload is defined as a set of commands necessary to encode or decode one frame.

The software driver is required to read the AVP disable fuse to determine if the AVP is enabled. If it is disabled, then the software driver must not enable AVP batch commands to be sent to the AVP or a hang event may occur. Only when the AVP is enabled through the fuse, should the batch commands be sent to the AVP.

AVP Command Sequence Examples for Decoder

AV1 workload is based upon a single tile decode. There are no states saved between tile decodes in the AVP.

The following programming sequence will be used by single pipe decode.

<<Start Workload>>	
|	

VD_CONTROL (AVP_Pipe_Initialization)	AVP PIpe Reset
|	
AVP_PIPE_MODE_SELECT	AVP Pipe Setup
|	

AVP_SURFACE_STATE (Multiple)	Frame Level Commands
AVP_PIPE_BUF_ADDR_STATE	Frame Level Commands
AVP_IND_OBJ_BASE_ADDR_STATE	Frame Level Commands
AVP_PIC_STATE	Frame Level Commands
AVP_SEGMENT_STATE	Frame Level Commands
AVP_INLOOP_FILTER_STATE	Frame Level Commands
if (FILM_GRAIN ON) {	
AVP_FILM_GRAIN_STATE	Frame Level Commands
}	
|	
AVP_TILE_CODING	Tile Level Commands
|	

AVP_BSD_OBJECT	Decode Start
|	
<< Tile Done >>	Decode Finish
|	
VD_CONTROL (AVP_Memory_Implicit_Fush)	HW Data Flush to Memory
|	
MI_FLUSH_DW (Protection Disable)	SW Flush and Protection OFF
|	
<<End Workload>>	

Each tile should be programmed independently

AVP Buffer Size Requirements

This documents all the Memory Buffer Size Requirement and Media Internal Storage Programming.

The following tables indicate AV1 rowstore size requirement.

The number below indicates is number of cacheline per SB (SB can be 64x64 or 128x128).

The rowstore data can be stored in Internal Media Storage.

To allocate the following: **Total CLs = (#CLs_per_SB * num_of_SB_per_tile_width)**

Surface	8 bits		10 bits		Comments
	SB64	SB128	SB64	SB128	
Bitstream Decoder/~~Encode~~ Line Rowstore (BTDL)	2	4	2	4	**Decoder Only**
Spatial Motion Vector Line Rowstore (SMVL)	4	8	4	8	
Intra Prediction Line Rowstore (IPDL)	2	4	4	8	
Deblocker Filter Line Y Buffer (DFLY)	9	17	11	21	
Deblocker Filter Line U Buffe (DFLU)	3	4	3	5	
Deblocker Filter Line V Buffe (DFLV)	3	4	3	5	

The following rowstore requires extra CL allocation in addition to CL per SB.

To allocate the following: **Total CLs = (#CLs_per_SB * num_of_SB_per_tile_width) + #CLs_extra_per_surface**

Surface	#CLs per SB				#CLs extra per surface			
	8 bit		10 bit		8 bit		10 bit	
	SB64	SB128	SB64	SB128	SB64	SB128	SB64	SB128
CDEF Filter Line Buffer (CDEF)	8	16	10	20	1	1	2	2

The following tables indicate AV1 tile storage. These will NOT be stored in Internal Media Storage.

To allocate the following, use the following equations based on Tile Line Rowstore or Tile Column Rowstore:

Total Tile Line CLs = (#CLs_per_SB * num_of_SB_per_FRAME_width)

Total Tile Column CLs = (#CLs_per_SB * num_of_SB_per_FRAME_height)

Note: In scalable mode, separate buffers must be allocated per pipes run concurrently.

[Programming suggestion: The largest tile width is 4096 in pixels. It is recommended to allocate the buffer based on 4096 tile width

and it can be reused for all tiles within the frame]

Surface	8 bit		10 bit		Comments
	SB64	SB128	SB64	SB128	
Bitstream Decode/Encode Tile Line Rowstore	2	4	2	4	**Decode/Encode**
Spatial Motion Vector Tile Line Rowstore	4	8	4	8	
Intra Prediction Tile Line Rowstore Tile Row	2	4	4	8	
Deblocker Filter Tile Line Y Buffer	9	17	11	21	
Deblocker Filter Tile Column Y Buffer	8	16	10	20	

Deblocker Filter Tile Line U Buffer	3	4	3	5	
Deblocker Filter Tile Column U Buffer	2	4	3	5	
Deblocker Filter Tile Line V Buffer	3	4	3	5	
Deblocker Filter Tile Column V Buffer	2	4	3	5	
CDEF Filter Top-Left Corner Buffer	**1 * num_Tile_Horz * num_Tile_Vert**				1 CL per tiles
CDEF Filter Meta Tile Line Buffer	**1 * num_Tile_Horz**				1 CL per horz tile
Loop Restoration Tile Line Y Buffer	**7 * num_Tile_Horz**				7 CL per horz tile
Loop Restoration Tile Line U Buffer	**5 * num_Tile_Horz**				5 CL per horz tile
Loop Restoration Tile Line V Buffer	**5 * num_Tile_Horz**				5 CL per horz tile

The following buffer requires extra CLs at the end of frame width/height. These will NOT be stored in Internal Media Storage.

To allocate the following, use the following equations based on Tile Line Rowstore or Tile Column Rowstore:

Total Tile Line CLs = (#CLs_per_SB * num_of_SB_per_FRAME_width) + #CLs_extra_per_surface

Total Tile Column CLs = (#CLs_per_SB * num_of_SB_per_FRAME_height) + #CLs_extra_per_surface

Note: In scalable mode, separate buffers must be allocated per pipes run concurrently.

[Programming suggestion: The largest tile width is 4096 in pixels. It is recommended to allocate the buffer based on 4096 tile width

and it can be reused for all tiles within the frame]

Surface	**#CLs per SB**				**#CLs extra per surface**			
	8 bit		**10 bit**		**8 bit**		**10 bit**	
	SB64	**SB128**	**SB64**	**SB128**	**SB64**	**SB128**	**SB64**	**SB128**
CDEF Filter Tile Line Buffer	8	16	10	20	1	1	2	2
CDEF Filter Tile Column Buffer	8	16	10	20	1	1	2	2
CDEF Filter Meta Tile Column Buffer	1	1	1	1	0	0	0	0
Super-Res Tile Column Y Buffer	22	44	29	58	22	44	29	58
Super-Res Tile Column U Buffer	8	16	10	20	8	16	10	20
Super-Res Tile Column V Buffer	8	16	10	20	8	16	10	20
Loop Restoration Filter Tile Column Y Buffer	9	17	11	22	2	2	2	2
Loop Restoration Filter Tile Column U Buffer	5	9	6	12	1	1	1	1
Loop Restoration Filter Tile Column V Buffer	5	9	6	12	1	1	1	1
Loop Restoration Meta Tile Column Buffer	1	1	1	1	1	1	1	1

The following table indicates AV1 frame buffer (except pixel buffer).

Surface	SB64	SB128	Comments
CDF Tables Initialization Buffer	**242** CLs		Intra frame uses 183 CLs; inter frame uses 242 CLs
CDF Tables Backward Adaptation Buffer	**242** CLs		
AV1 Segment ID Read Buffer	**2** * Total_Num_SB64_in_Frame	**8** * Total_Num_SB128_in_Frame	
AV1 Segment ID Write Buffer	**2** * Total_Num_SB64_in_Frame	**8** * Total_Num_SB128_in_Frame	
Collocated Motion Vector Temporal Buffer	**4** * Total_Num_SB64_in_Frame	**16** * Total_Num_SB128_in_Frame	
Current Frame Motion Vector Write Buffer	**4** * Total_Num_SB64_in_Frame	**16** * Total_Num_SB128_in_Frame	

This section documents the Internal Media Storage Programming for AV1 decoder.

The following table is created for a maximum of 4k tile width.

Since AV1 has a restriction of maximum 4k tile width and each tile is programmed

per tile independently, only 4k tile programming is needed (unlike other codec)

			Address Programming (N/A means disable)							
Format	Bitdepth	TileSize	BTDL	SMVL	IPDL	DFLY	DFLU	DFLV	CDEF	
420	8/10 bit	<= 4k	0	128	384	640	1344	1536	1728	N/A

AVP Common Commands

This documents commands only for AVP codec decoder

Commands
AVP_REF_IDX_STATE
AVP_SEGMENT_STATE
AVP_BSD_OBJECT

AVP Pipe Common Commands

The AVP Pipe Common Commands specify the AVP Decoder pipeline level configuration.

Shared Commands
VD_CONTROL_STATE

AVP Commands
AVP_PIPE_MODE_SELECT
AVP_SURFACE_STATE

AVP Commands
AVP_PIC_STATE
AVP_PIPE_BUF_ADDR_STATE
AVP_TILE_CODING
AVP_IND_OBJ_BASE_ADDR_STATE

Video Command Streamer (VCS)

The VCS (Video Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the MFD Engine. It is responsible for fetching, decoding, and dispatching of data packets (Media Commands with the header DWord removed) to the front-end interface module of MFX Engine.

Its logic functions include:

- MMIO register programming interface
- DMA action for fetching of execlists and ring data from memory
- Management of the Head pointer for the Ring Buffer
- Decode of ring data and sending it to the appropriate destination: AVC, VC1, or MPEG2 engine
- Handling of user interrupts
- Handling of ring context switch interrupt
- Flushing the MFX Engine
- Handle NOP

The register programming (RM) bus is a DWord interface bus that is driven by the Gx Command Streamer. The VCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x4000 to 0x4FFFF. The Gx and MFX Engines use semaphore to synchronize their operations.

VCS operates completely independent of the Gx CS.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside VCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (16 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

The parser starts executing once the DMA FIFO has valid commands. All the commands have a header DWord packet. Based on the encoding in the header packet, the command may be targeted towards AVC/VC1/MPEG2 engine or the command parser. After execution of every command, the actual head pointer is updated. The ring is considered empty when the head pointer becomes equal to the tail pointer.

Context Management

Video Engine Power Context

This section lists the power context image of Video Engine across generations.

Video Engine Power Context

Table below captures the data from VCS power context save/restored by PM. Address offset in the below table is relative to the starting location of VCS in the overall power context image managed by PM.

Address offsets in this table are relative to the starting location of VCS in the power context image managed by each engine. MMIO offset mentioned for the registers in the below table are offset from the units "MMIO Base Offset" mentioned in the table " Base Offset for all engines in the section Register Access and User Mode Privileges. For Example: VCS has MMIO Base Offset as "0x1C_0000". In the below table GFX_MODE register has 0x0029C as offset against it, actual MMIO Offset of GFX_MODE register for VCS is 0xx1C_029C and for VECS it would be 0x1C_829C.

VCS Power Context Image

Description	MMIO Offset	Unit	# of DW	Address Offset (PWR)	CSFE/CSBE
CSFE Power context without Display		VCS	192	**0**	**CSFE**
NOOP		VCS	1	**00C0**	**CSBE**
Load_Register_Immediate header	0x1100_1005	VCS	1	**00C1**	**CSBE**
GAC MODE REGISTER	0x000a0	VCS	2	**00C6**	**CSBE**
HUC_PMCR_REGISTER	0x00400	HUC	1	**00C8**	**CSBE**
VCS_WAKERATE_HCP	0x006E0	VCS	2	**00CA**	**CSBE**
VCS_WAKERATE_MFX	0x006E4	VCS	2	**00CC**	**CSBE**
VCS_SUBWELL	0x006E8	VCS	2	**00CE**	**CSBE**
VCS_BUSYNESS_VCS	0x006EC	VCS	2	**00D0**	**CSBE**
VCS_BUSYNESS_HCP	0x006F0	VCS	2	**00D2**	**CSBE**
VCS_BUSYNESS_MFX	0x006F4	VCS	2	**00D4**	**CSBE**
NOOP		VCS	9	**00D9**	**CSBE**
MI_BATCH_BUFFER_END		VCS	1	**00DF**	**CSBE**

VDBOX - Engine Register State and Context

This section discusses the following topics for the BSD Logical Render Context Address (LRCA):

- Ring Context
- Register State Context

Register State Context

EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)
ENGINE CONTEXT

Description	Description When Suspended Context	Unit	Dword Count	Address Offset (Dword)
CSFE Execlist Context		VCSFE	192	**0**
MI_BATCH_BUFFER_END		CSEND	1	**00C0**
NOOP		CSEND	127	**00C1**
			DW	320
			K Bytes	1.25

Video Command Formats

MFX Commands

The MFX (MFD for decode and MFC for encode) commands are used to program the multi-format codec engine attached to the Video Codec Command Parser. See the *MFD* and *MFC* chapters for a description of these commands.

MFX state commands support direct state model and indirect state model. Recommended usage of indirect state model is provided here (as a software usage guideline).

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
MFX Common (State)							
2h	0h	0h	0h	MFX_PIPE_MODE_SELECT	MFX	IMAGE	N/A
2h	0h	0h	1h	MFX_SURFACE_STATE	MFX	IMAGE	N/A
2h	0h	0h	2h	MFX_PIPE_BUF_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	3h	MFX_IND_OBJ_BASE_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	4h	MFX_BSP_BUF_BASE_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	6h	MFX_ STATE_POINTER	MFX	IMAGE	N/A
2h	0h	0h	7-8h	Reserved	N/A	N/A	N/A
MFX Common (Object)							
2h	0h	1h	9h	MFD_ IT_OBJECT	MFX	N/A	Yes
2h	0h	0h	4-1Fh	Reserved	N/A	N/A	N/A
AVC Common (State)							
2h	1h	0h	0h	MFX_AVC_IMG_STATE	MFX	IMAGE	N/A
2h	1h	0h	1h	MFX_AVC_QM_STATE	MFX	IMAGE	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
2h	1h	0h	6-1Fh	Reserved	N/A	N/A	N/A
AVC Dec							
2h	1h	1h	0-7h	Reserved	N/A	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 Enc							
2h	1h	2h	0-1h	Reserved	N/A	N/A	N/A
2h	1h	2h	2h	MFC_AVC_FQM_STATE	MFX	IMAGE	N/A
2h	1h	2h	3-7h	Reserved	N/A	N/A	N/A
2h	1h	2h	8h	MFC_AVC_PAK_INSERT_OBJECT	MFX	N/A	N/A
2h	1h	2h	9h	MFC_AVC_PAK_OBJECT	MFX	N/A	Yes
2h	1h	2h	A-1Fh	Reserved	N/A	N/A	N/A
2h	1h	2h	0-1Fh	Reserved	N/A	N/A	N/A
VC1 Common							
2h	2h	0h	0h	MFX_VC1_PIC_STATE	MFX	IMAGE	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	2-1Fh	Reserved	N/A	N/A	N/A
VC1 Dec							
2h	2h	1h	0-7h	Reserved	N/A	N/A	N/A

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
2h	2h	1h	8h	MFD_VC1_BSD_OBJECT	MFX	N/A	Yes
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							
2h	3h	0h	0h	MFX_MPEG2_PIC_STATE	MFX	IMAGE	N/A
2h	3h	0h	1h	MFX_MPEG2_QM_STATE	MFX	IMAGE	N/A
2h	3h	0h	2-1Fh	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	Yes
2h	3h	1h	9-1Fh	Reserved	N/A	N/A	N/A
MPEG2 Enc							
2h	3h	2h	0-1Fh	Reserved	N/A	N/A	N/A
The Rest							
2h	4-5h, 7h	x	x	Reserved	N/A	N/A	N/A

Video Command Header Format

Video Command Header Format

Type	Bits				
	31:29	28:24	23	22	21:0
Memory Interface (MI)	000	Opcode 00h - NOP 0Xh - Single DWord Commands 1Xh - Reserved 2Xh - Store Data Commands 3Xh - Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 - DWord Count 5:0 - DWord Count 5:0 - DWord Count	

Type	Bits				
	31:29	28:27	26:24	23:16	15:0
Reserved	011	00	XXX	XX	
MFX Single DW	011	01	000	Opcode: 0h	0
Reserved	011	01	1XX		
Reserved	011	10	0XX		
AVC State	011	10	100	Opcode: 0h - 4h	DWord Count
AVC Object	011	10	100	Opcode: 8h	DWord Count
VC1 State	011	10	101	Opcode: 0h - 4h	DWord Count
VC1 Object	011	10	101	Opcode: 8h	DWord Count
Reserved	011	10	11X		
Reserved	011	11	XXX		

Type	Bits					
	31:29	28:27	26:24	23:21	20:16	15:0
MFX Common	011	10	000	000	subopcode	DWord Count
Reserved	011	10	000	001-111	subopcode	DWord Count
AVC Common	011	10	001	000	subopcode	DWord Count
AVC Dec	011	10	001	001	subopcode	DWord Count
AVC Enc	011	10	001	010	subopcode	DWord Count
Reserved	011	10	001	011-111	subopcode	DWord Count
Reserved (for VC1 Common)	011	10	010	000	subopcode	DWord Count
VC1 Dec	011	10	010	001	subopcode	DWord Count
Reserved (for VC1 Enc)	011	10	010	010	subopcode	DWord Count
Reserved	011	10	010	011-111	subopcode	DWord Count
Reserved (MPEG2 Common)	011	10	011	000	subopcode	DWord Count
MPEG2Dec	011	10	011	001	subopcode	DWord Count
Reserved (for MPEG2 Enc)	011	10	011	010	subopcode	DWord Count
Reserved	011	10	011	011-111	subopcode	DWord Count
Reserved	011	10	100-111	XXX		

Watchdog Timer Registers

The following registers are defined as Watchdog Timer registers:

Register
PR_CTR_CTL - Watchdog Counter Control
PR_CTR_THRSH - Watchdog Counter Threshold

Logical Context Support

This section contains the registers for Logical Context Support.

Register
BB_STATE - Batch Buffer State Register
CXT_EL_OFFSET - Exec-List Context Offset
BB_START_ADDR - Batch Buffer Start Head Pointer Register
BB_START_ADDR_UDW - Batch Buffer Start Upper Head Pointer Register
BB_ADDR_DIFF - Batch Address Difference Register
BB_ADDR - Batch Buffer Head Pointer Register
SBB_ADDR - Second Level Batch Buffer Head Pointer Register
SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register
WAIT_FOR_RC6_EXIT - Control Register for Power Management
SBB_STATE - Second Level Batch Buffer State Register
BB_OFFSET - Batch Offset Register
RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG
BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register
BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register
SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register
SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register
MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1
MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2
DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register
FORCE_TO_NONPRIV - FORCE_TO_NONPRIV
INDIRECT_CTX - Indirect Context Pointer
INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer
BB_PER_CTX_PTR - Batch Buffer Per Context Pointer
SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register
SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1
SYNC_FLIP_STATUS_2 - Wait For Event and Display Flip Flags Register 2

Mode Registers

The following are Mode Registers:

Mode Register
MI_MODE - Mode Register for Software Interface
INSTPM - Instruction Parser Mode Register
NOPID - NOP Identification Register
IDLEDLY - Idle Switch Delay
RESET_CTRL - Reset Control Register
PREEMPTION_HINT - Preemption Hint
PREEMPTION_HINT_UDW - Preemption Hint Upper DWord
SEMA_WAIT_POLL - Semaphore Polling Interval on Wait

Misc Register
HWS_PGA - Hardware Status Page Address Register
Hardware Status Page Layout

Registers in Media Engine

This topic describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. The functions performed by some of these registers are discussed in more detail in the Memory Interface Functions, Memory Interface Instructions, and Programming Environment chapters.

The registers detailed in this chapter are used across multiple projects and are extentions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

Register
TIMESTAMP - Reported Timestamp Count
CTX_TIMESTAMP - Context Timestamp Count

Memory Interface Commands for Video Codec Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the Video Codec Engine.

The commands detailed in this chapter are used across product families. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for details.

MI Commands
MI_BATCH_BUFFER_END
MI_CONDITIONAL_BATCH_BUFFER_END
MI_BATCH_BUFFER_START
MI_FLUSH_DW
MI_COPY_MEM_MEM
MI_LOAD_REGISTER_REG
MI_MATH
MI_NOOP
MI_REPORT_HEAD
MI_SEMAPHORE_SIGNAL
MI_SEMAPHORE_WAIT
MI_STORE_REGISTER_MEM
MI_STORE_DATA_IMM
MI_SUSPEND_FLUSH
MI_USER_INTERRUPT
MI_LOAD_REGISTER_MEM
MI_ATOMIC
MI_FORCE_WAKEUP

HCP

HCP HW Codec Pipeline Introduction

The HEVC/VP9 Codec Pipeline (HCP) is a fixed function hardware video codec responsible for decoding and encoding HEVC/VP9 (High Efficiency Video Coding) video streams.

Scope

The primary scope of the HCP document is to provide a description of the HCP commands processed by the Video Command Streamer (VCS). The secondary scope is to provide a description of the status registers on the Message Channel Interface to support encoding and decoding of the HEVC and VP9 video formats.

The sections include:

- Summary of Features
- Architecture Overview
- Commands
- Register Definitions

Acronyms and Applicable Standards

Summary of Features

The following sections define the general features of the HCP HW Decoder and Encoder pipeline, and the features specific to HEVC and VP9 decoding and encoding, respectively.

VP9 Decoder Features

• Support full-featured VP9 Profile 1 and part of Profile 2 (444 only), up to 8K.

• All headers (uncompressed and compressed header) are parsed and decoded outside the HCP HW pipeline. They are then fed to the HW through a set of HCP (with VP9 specific) state commands.

• Supports inner-loop decode part of the VP9 encoder implementation.

VP9 Encoder Features

- Supports ENC-PAK architecture
- Supports multiple pass BRC rate control operation flow
- Supports the VP9 Main Profile standard
- VP9 PAK Only mode is not supported.

HCP Hardware Pipeline Features

- Supports both decoder and encoder functions, setup on a per picture basis:
  - Hardware acceleration provides Ctb/CU level decode and encode.
  - No context switch is supported within a frame process.
- Supports Video Command Streamer (VCS):
  - Shared with MFX HW pipeline, and at any one time, only one pipeline (MFX or HCP) and one operation (decoding or encoding) can be active.
- Supports Message Channel Interface:

Feature
Supports Tile-YS and Tile-YF.
Supports Tile-Y Legacy.

- Supports NV12 video buffer plane:
  - Supports 4:2:0, 8-bit per pixel component (Y, Cb and Cr) video.
- Supports 8Kx8K frame size.

HEVC Decoder Features

- Supports full-featured HEVC Main Profile standard, up to Level 6.2.
- Supports the long format HW decoding interface:
  - All headers (SPS, PPS, Slice Header) are parsed and decoded outside the HCP HW pipeline. They are then fed to the HW through a set of HCP state commands.

- Supports inner-loop decode with hardware entry points for Encoder.
- Error detection/resiliency down to the Ctb/CU level.

All 41 HEVC profiles:

- Yellow colored profiles have explicit GUID assigned
- Pink colored profiles are subset of Yellow colored profiles, and do not have their own GUID.
- Grey colored profiles are not supported by Intel HW at all.

No.	Version	HEVC Profiles	Spec. Description	Intel HW Decoder	Intel HW VDENC	Intel HW VME
1	Base	Main	8b 4:2:0 only			
2	Base	Main 10	8/9/10b 4:2:0 only			
3	Base	Main Still Picture	8b 4:2:0, 1 frame only			
7	RExt	Main 12	8 to 12b 400/420	-Mono		
8	RExt	Main 4:2:2 10	8/9/10b 400/420/422	-Mono		
9	RExt	Main 4:2:2 12	8-12b 400/420/422	G -Mono		
10	RExt	Main 4:4:4	8b 400/420/422/444	-Mono		
11	RExt	Main 4:4:4 10	8/9/10b 400/420/422/444	-Mono		
12	RExt	Main 4:4:4 12	8-12b 400/420/422/444	-Mono		
15	RExt	Main 12 Intra		-Mono		
16	RExt	Main 4:2:2 10 Intra		-Mono		
17	RExt	Main 4:2:2 12 Intra		-Mono		
18	RExt	Main 4:4:4 Intra		-Mono		
19	RExt	Main 4:4:4 10 Intra		-Mono		

No.	Version	HEVC Profiles	Spec. Description	Intel HW Decoder	Intel HW VDENC	Intel HW VME
20	RExt	Main 4:4:4 12 Intra		-Mono		
22	RExt	Main 4:4:4 Still Pic	8b 400/420/422/444	-Mono		
28	V3	Screen-Extended Main	8b 400/420	-Mono		
29	V3	Screen-Extended Main 10	8/9/10b 400/420	-Mono		
30	V3	Screen-Extended Main 4:4:4	8b 400/420/444 + Main 4:4:4	-Mono		
31	V3	Screen-Extended Main 4:4:4 10	8/9/10b 400/420/444 + Main 4:4:4 10	-Mono		

HEVC Encoder Features

- Supports ENC-PAK architecture
- Supports multiple pass BRC rate control operation flow
- Supports the HEVC Main Profile standard, with certain restrictions on the feature set and coding parameters, listed in the following table:

HEVC Encoder Features and Restrictions

Note that there is a difference between what PAK supported and what ENC supported. A feature/function that is supported in the PAK, does not necessary being supported by ENC and MediaSDK and the like.

Coding Tool	Support	Restriction	Comments
LCU Size	Yes (spec)		Support all 3 sizes: 16x16, 32x32, 64x64.
CU Size	Yes (spec)		Support 8x8, 16x16, 32x32, 64x64. Max 64 CU per LCU; min. CU size intel supported is 8x8 for all LCU size.
PU Partition	Yes (spec)		Support all inter symmetric (square) and asymmetric (non-square) PU partitioning, according to HEVC spec. PU Size for inter : Smallest allowed is 4x8 and 8x4, and they cannot be bidirectional. Inter 4x4 PU is not allowed in Main Profile.
TU QuadTree		Partial (intel) max depth is set to 3	Max depth is 3 (64x64 CU with 4x4 TU). Decoder supports this depth, but probably no need to search this for encode. Better to just split CU. HM common conditions set the max to 2 for both inter and intra. Intel Encoder only supports 2 levels of quad-tree. That is, max_transform_hierarchy_depth_inter/intra <= 2. Max num of TUs per CU is 16.
AMP	Yes (spec)		Asymmetric Motion Partition (rectangular PU partitioning - 2NxnU, 2NxnD, nLx2N or nRx2N). Available only for 64x64 to 16x16 CU.
AMVP	Yes (spec)		Adaptive/Advanced Motion Vector Prediction: spatial and PU-based temporal co-located MV candidates with scaling. Logic available from decoder. HW PAK is supporting temporal MV candidates.
Merge	Yes (spec)		Merge Skip and Regular Merge. Max. 5 MV Merge candidates (4 spatial + 1 temporal co-located) with scaling. Logic available from decoder. [merge_flag, merge_index, skip_flag]
Parallel Motion Merge		No (intel)	Tool for parallel decode of MVs. Since this isn't constrained by Main Profile, the decoder has to meet performance targets in the worst case anyway.
MC Interpolation Filter	Yes (spec)		1/4-pel Luma MV precision, 1/8-pel Chroma MV precision. 8-tap Luma filtering for both 1/2-pel and 1/4-pel locations (1-pass). 4-tap Chroma filtering. Use separable (first horizontal then vertical 1-D filtering) filter coefficients. Not all filter kernels are symmetrical and can map into simple arithmetic. It is a DCT-IF based filter. All operations are within 16-bit data.
Weighted Prediction	Yes (spec)		Free for PAK since decoder already has it. Cross fade and fade-in/out detection required by VME. VME must supply weights, implicit weighted prediction is not supported in HEVC. Only explicit weight is supported. They are both unidirectional and bidirectional.
Combined Reference Frame List		No (intel)	Combine List0 and List1 into a single list to remove uni-prediction signaling overhead.

Coding Tool	Support	Restriction	Comments
Intra modes	Yes (spec)		33 directions and DC/Planar modes, with adaptive pre-filtering on reference pixels and boundary smoothing. Intel ENC-VME Mode search reduction for large CUs (64x64 and 32x32) for performance speed up with minimal loss.
IPCM (intra)		No (intel)	Can be disabled completely by SPS, or only allowed at certain CU sizes. No bit maximum on CUs in any profile/level (yet), so as of today there's no mandate to support this for encode.
Constrained Intra		No (intel)	Allow only intra neighboring blocks for current block intra-prediction. Enabling this is a coding loss and does not result in a performance improvement in HW designs.
2D DCT Transform	Yes (spec)		Square shape only; 32x32, 16x16, 8x8 and 4x4.
Transform Skip Evaluation		No (intel) ENC will estimate the use of transform skip	Significant coding gains for screen content (PowerPoint etc.). This tool is disabled in the common conditions but isn't explicitly disallowed by Main Profile. FQ is not bypass.
Sign bit hiding		No (intel)	Coding gain by removing one bypass bin per TU. Requires some smarts in the PAK.
Trellis		No (intel)	Trellis Quantization
SAO		No (intel)	Difficult to implement in single pass, performance impact in 2-pass or with previous frame search. Needs investigation. Decoder will support it.
Loop Filter across tiles/slices boundary		No (intel)	Can be disabled for tiles and or slices in SPS, so that filter across all tiles and slices boundaries. Main profile doesn't constrain.
Scaling List	Yes (spec)		This uses the default (or custom) qp adjustment on a per-frequency basis within a TU. Good coding improvement over flat scaling.
dQP		Partial (intel) Yes for LCU; No for CU	Being able to change QP per LCU or even up to once per 8x8 CU can lead to significant coding gains. Not sure how easy it is for VME to decide qp values though.
Chroma QP offset		No (intel)	No ROI.
Dependent slices		No (intel)	It is now part of Main Profile, but Intel will not support it. SW can perform the slice repackaging without re-encoding.
Tiles		No (intel)	Although in Main Profile, it results in coding loss and doesn't improve performance on HW. SW parallel processing (multithreaded) tool
Wavefront (aka WPP)		No (intel)	Latest Main Profile spec has included Wavefront. We got a feedback that this feature is highly desirable to support high performance multithreading HEVC decoder.
Lossless coding		No (intel)	Note: this is not the same as IPCM. Also details of this are in flux.

Coding Tool	Support	Restriction	Comments
Interlaced Video		No (intel)	Only progressive video encoding is supported.
LM mode	No (spec)		Chroma-from-luma intra prediction (Linear Mode) is not allowed in Main Profile (yet).
NSQT	No (spec)		Non square transform is not allowed in the Main Profile
ALF	No (spec)		Expensive and not in Main Profile currently. If decoder is going to support it, may consider for encoder as well.
Entropy slices	No (spec)		Not allowed in Main Profile, it is a SW parallel processing (multithreaded) tool.
Slice granularity != 0	No (spec)		Not allowed by Main Profile, and highly likely to be removed from the standard completely.

Architecture Overview

HCP HW pipeline is designed to support two codec standards: HEVC and VP9. It implements the complete decoder process, but does not handle header (sequence header, frame/picture header, slice header and tile header) parsing which is to be done by application/driver at software level. It also implements the bitstream coding, residual generation and frame reconstruction part of the encoding process (namely PAK), whereas the bit rate control, motion estimation and the block coding decision are done either in software and/or in a separate HW modules.

For decoder, both HEVC and VP9 are fully compliant to the standards, while for PAK, only a subset of coding tools are implemented.

The HCP can be programmed to function as either VP9 or HEVC at frame level at a time. The command sequence for each codec is frame based.

HEVC/VP9 Encoder

The HEVC/VP9 encoder architecture consists of 2 major HW components: VDENC and PAK. In addition, the HEVC architecture also supports a 3 HW components mode: Media ENC (EUs/Kernels+VME), and PAK. Media EUs/Kernels implement the ENC portion of the encoding process. It communicates with the VME to determine the best inter and intra coding modes for each block based on a set of cost functions and algorithms. It also responsible for setting up multiple encoding passes to meet the target coding efficiency. For both modes, the PAK is used to generate the final compressed bitstream on a per LCU basis with coding parameters received from the ENC. It also provides feedback information for BRC rate control purpose. As part of the PAK operation, it invokes the decoder in the reconstruction process.

HCP Command Summary

The HCP is configured for encoding or decoding through a set of batch commands defined in the following sections. The software driver builds a frame level workload using these commands and stores

these workloads in graphics memory where they are fetched by the Video Command Streamer (VCS) and presented to the HCP for processing. The commands are processed by the Workload Parser within the HCP and the hardware is configured by the Workload Parser prior to each frame level encode or decode. A workload is defined as a set of commands necessary to encode or decode one frame.

The software driver is required to read the HCP disable fuse to determine if the HCP is enabled. If it is disabled, then the software driver must not enable HCP batch commands to be sent to the HCP or a hang event may occur. Only when the HCP is enabled through the fuse, should the batch commands be sent to the HCP.

Workload Command Model

DWord0 of each command is defined in HCP DWord0 Command Definition. The HCP is selected with the **Media Instruction Opcode "7h**" for all HCP Commands.

HCP DWord0 Command Definition

DWord	Bits	Description
0	31:29	**Command Type** = PARALLEL_VIDEO_PIPE = 3h
	28:27	**Pipeline Type** = 2h
	26:23	**Media Instruction Opcode = Codec/Engine Name** = HCP = 7h
	22:16	**Media Instruction Command** = <*see* HCP Media Instruction Commands (Opcode=7h)>
	15:12	**Reserved: MBZ**
	11:0	**Dword Length** (Excludes Dwords 0, 1) = <*command length*>

Each HCP command has assigned a media instruction command as defined in HCP Media Instruction Commands (Opcode=7h).

HCP Media Instruction Commands (Opcode=7h)

Media Instruction Command	Command DWord0 [22:16]	Mode	Scope
HCP_PIPE_MODE_SELECT	0h	Enc/Dec	Picture
HCP_SURFACE_STATE	1h	Enc/Dec	Picture
HCP_PIPE_BUF_ADDR_STATE	2h	Enc/Dec	Picture
HCP_IND_OBJ_BASE_ADDR_STATE	3h	Enc/Dec	Picture
HCP_QM_STATE	4h	Enc/Dec	Picture
HCP_FQM_STATE (encoder only)	5h	Enc	Picture
Reserved	8h-Fh		
HCP_PIC_STATE	10h	Enc/Dec	Picture
HCP_TILE_STATE	11h	Dec	Picture
HCP_REF_IDX_STATE	12h	Enc/Dec	Slice
HCP_WEIGHTOFFSET_STATE	13h	Enc/Dec	Slice
HCP_SLICE_STATE	14h	Enc/Dec	Slice
HCP_TILE_CODING	15h	Enc/Dec	Tile
Reserved	16h-1Fh		
HCP_BSD_OBJECT_STATE (decoder only)	20h	Dec	Slice
HCP_PAK_OBJECT (encoder only)	21h	Enc	LCU
HCP_INSERT_PAK_OBJECT (encoder only)	22h	Enc	Bitstream
Reserved	23h-2Fh		
HCP_VP9_PIC_STATE	30h	Dec	Picture
HCP_VP9_SEGMENT_STATE	32h	Dec	Picture
HCP_VP9_PAK_STATE	35h	Enc	LCU
HCP_VP9_RDOQ_STATE	3Ch	Enc	LCU
Reserved	3Dh-7Fh		

HCP Command Sequence Examples

VP9 Encoder Command Sequence

For a single frame encoding process (w/o encryption) the command sequence is listed below. There are no states saved between frame encoded in the HCP. There should be no other commands or context switch within a group of PAK OBJECT Commands. HCP and MFX share the same VCS, but there is no common encoding and decoding command that can be executed in both pipes, except the encryption commands, mi_flush and MMIO commands.

------ Per Frame Level Commands

HCP_PIPE_MODE_SELECT

HCP_SURFACE_STATE

HCP_PIPE_BUF_ADDR_STATE

HCP_IND_OBJ_BASE_ADDR_STATE

HCP_VP9_PIC_STATE

HCP_VP9_SEGMENT_STATE

HCP_VP9_QUANT_LOOKUP_TABLES

HCP_PAK_INSERT_OBJECT - if header present at the beginning of frame

    ------- A group of LCUs

    HCP_PAK_OBJECT

    ...

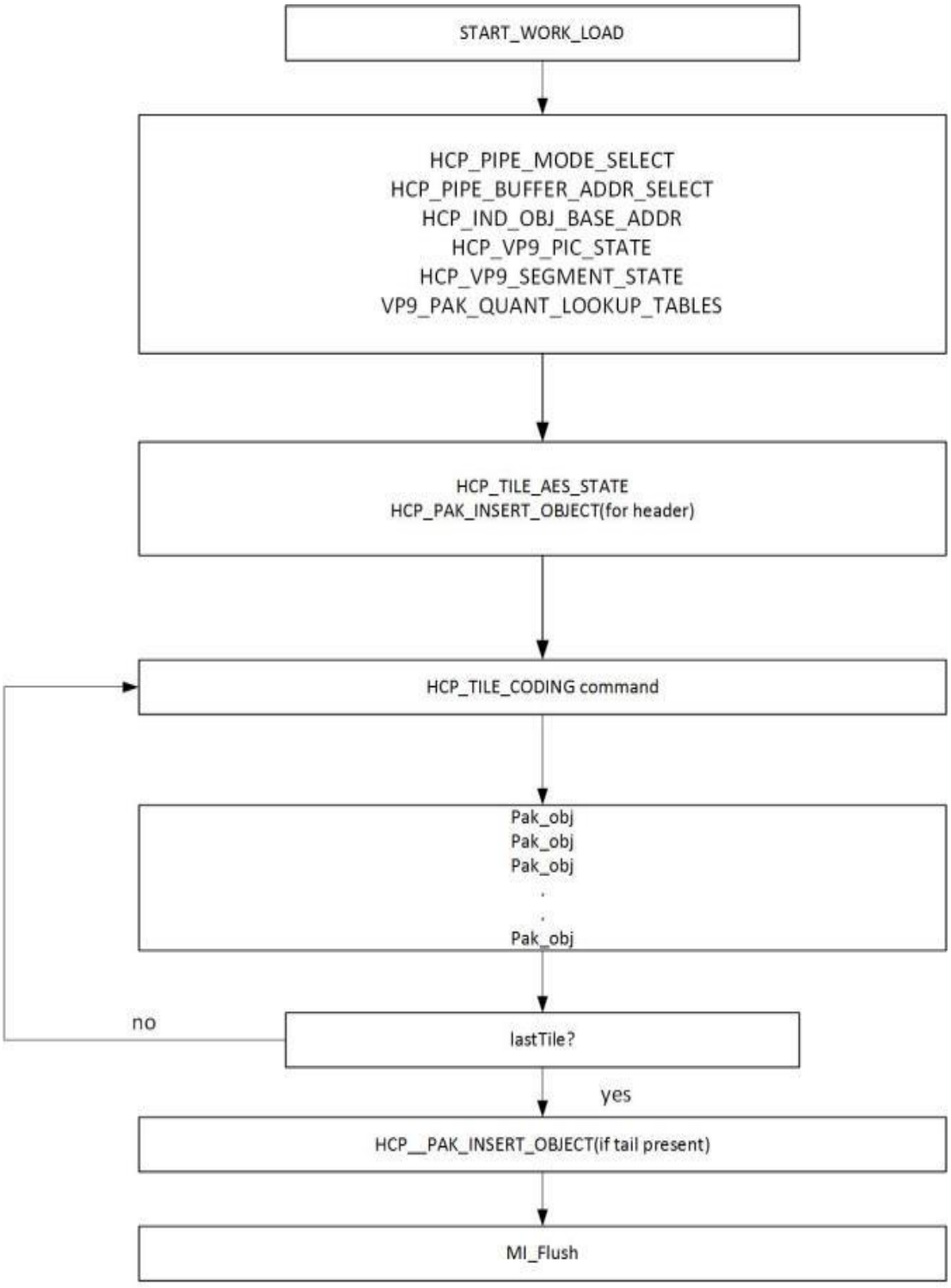
HCP_PAK_INSERT_OBJECT - if tail present at frame end

MI_FLUSH - when the frame is done

Command Sequence in Single Pipe Mode

Command Sequences with Tile Support

Single Pipe Mode- Following flow chart shows the command sequence when encoding frame using a single pipe(VDbox).



VP9 encode command sequence for Single Pipe mode

Multiple Pipe Mode- When encoding a frame using multiple pipes, each pipe gets a single Tile Column or multiple Tile columns depending upon Number of tile columns to encode. Following flow chart shows commands sequence in a pipe (VDbox) when multiple pipes are used for encoding.

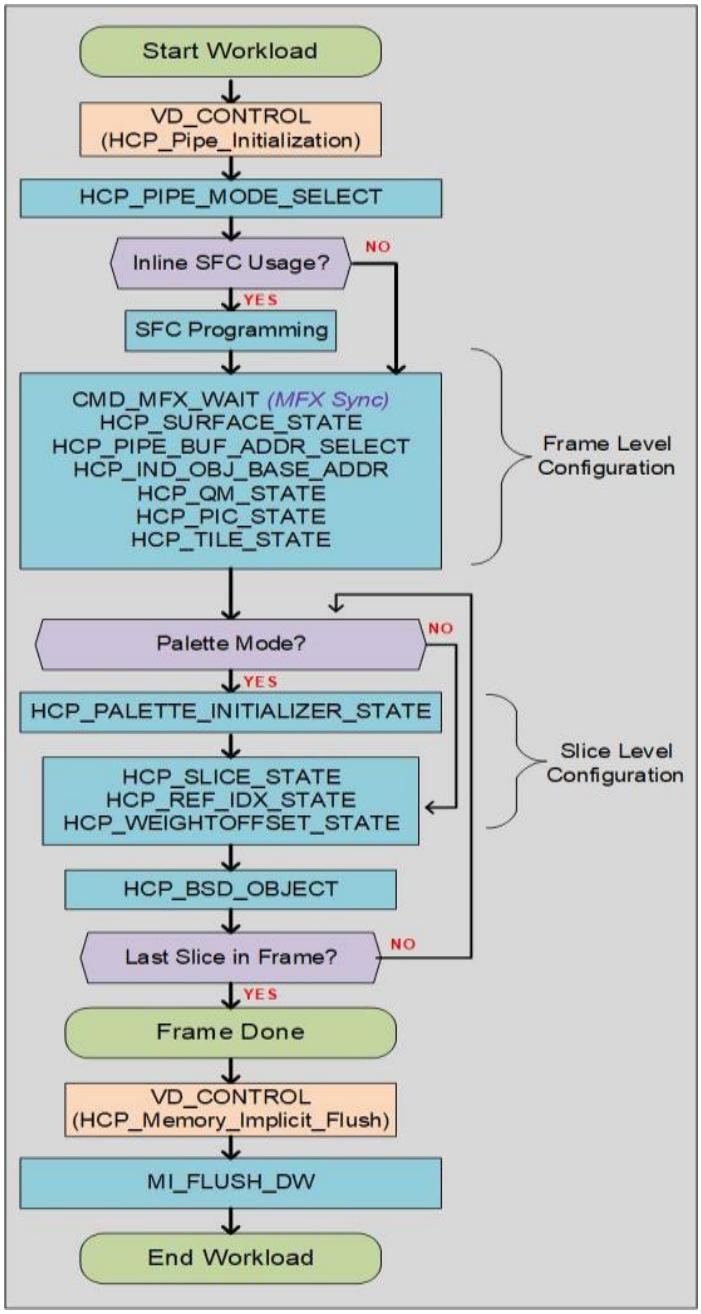
VP9 encode command sequence in Multiple Pipes mode

HCP Decoder Command Sequence

The long format workload for the HCP is based upon a single frame decode. There are no states saved between frame decodes in the HCP. Once the bit stream DMA is configured with the HCP_BSD_OBJECT command, and the bit stream is presented to the HCP, the frame decode will begin.

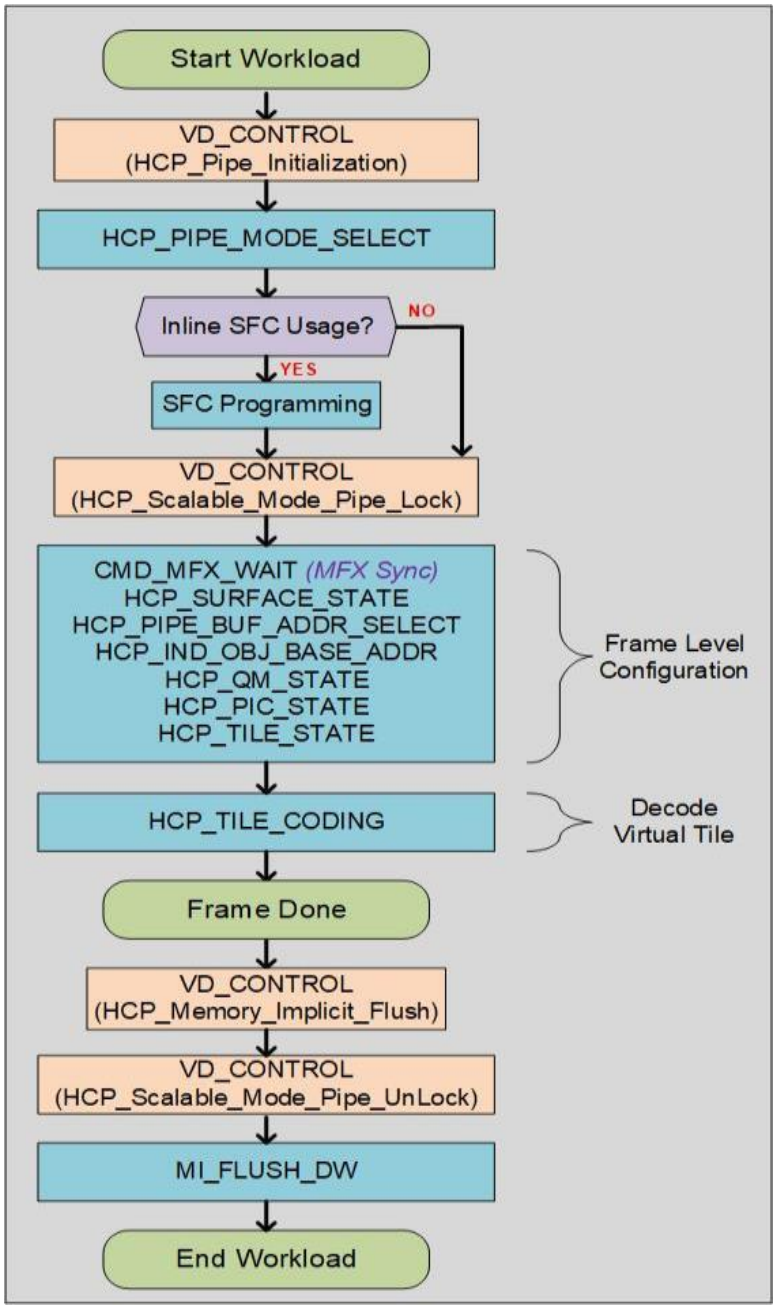
HCP Long Format Decode Workload Chart

The following programming sequence will be used by single pipe decode (CABAC+BE reconstruction) or scalable CABAC only decode mode.



HCP Scalabe Backend Only Workload Chart

The scalable decoder workload for the HCP backend pipe is based upon a single frame decode using multiple backend pipes. The frame is split into multple "virtual" vertical (column) tiles and they are processed by multple linked backend pipes. [NOTE: the above command sequence is still used for HCP CABAC decode and the decoded syntax elements are streamed to memory.]

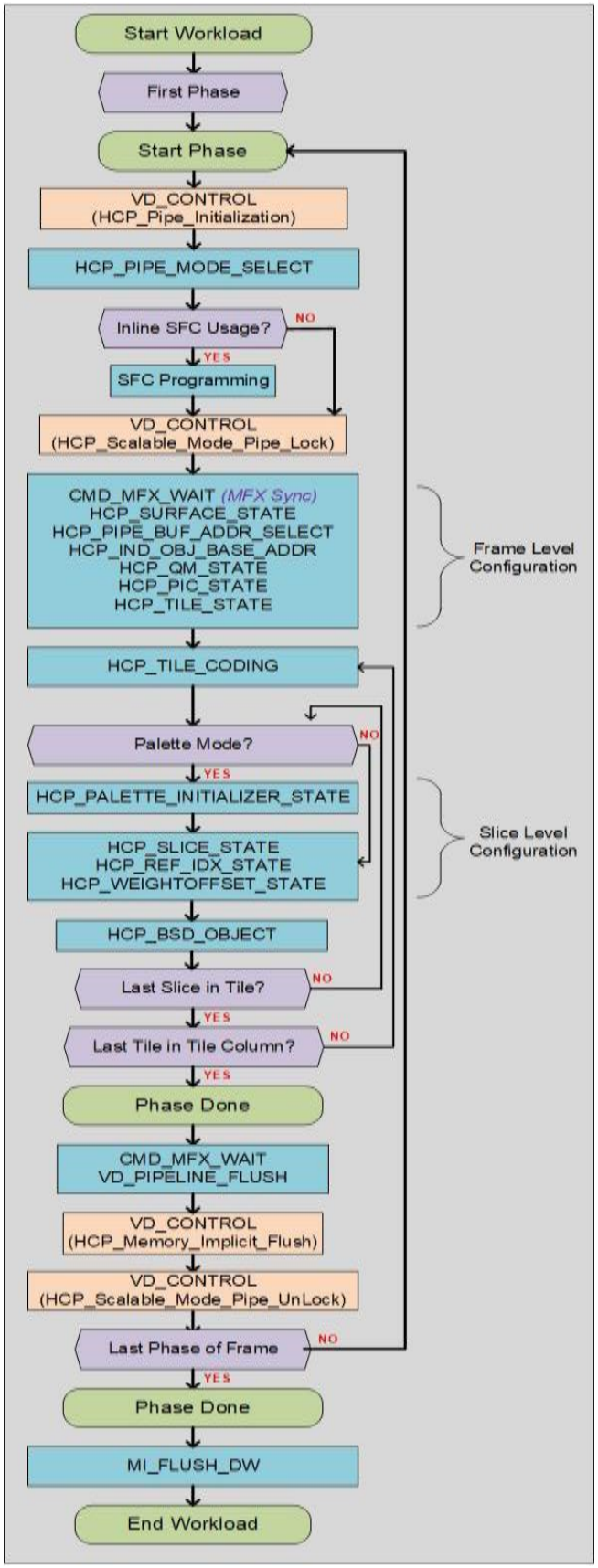


The scalable decoder with CABAC in real tiles allows frame with tiles to be decodes by multiple pipes. Each pipes will decode separate tile columns.

In this mode, the CABAC and BE will link and decode together. The following is the programming sequence.

Also, the number of tiles may be greater than the number of pipes used to decode the frame. In this case, multiple phases will be introduced in the programming. For example, if N number of pipes are used to decode the frame. The first phases will decode 0 to (N-1) tile column. The next phase will decode N to (2N -1). This will continue till all the tile columns are processed.

[NOTE: The last phases may have fewer than N tile columns. In this case, only the needed pipes will be programmed and used.


Start Workload
First Phase
Start Phase
VD_CONTROL
(HCP_Pipe_Initialization)
HCP_PIPE_MODE_SELECT
Inline SFC Usage?
NO
YES
SFC Programming
VD_CONTROL
(HCP_Scalable_Mode_Pipe_Lock)
CMD_MFX_WAIT (MFX Sync)
HCP_SURFACE_STATE
HCP_PIPE_BUF_ADDR_SELECT
HCP_IND_OBJ_BASE_ADDR
HCP_QM_STATE
HCP_PIC_STATE
HCP_TILE_STATE
Frame Level
Configuration
HCP_TILE_CODING
Palette Mode?
NO
YES
HCP_PALETTE_INITIALIZER_STATE
HCP_SLICE_STATE
HCP_REF_IDX_STATE
HCP_WEIGHTOFFSET_STATE
Slice Level
Configuration
HCP_BSD_OBJECT
Last Slice in Tile?
NO
YES
Last Tile in Tile Column?
NO
YES
Phase Done
CMD_MFX_WAIT
VD_PIPELINE_FLUSH
VD_CONTROL
(HCP_Memory_Implicit_Flush)
VD_CONTROL
(HCP_Scalable_Mode_Pipe_UnLock)
Last Phase of Frame
NO
YES
Phase Done
MI_FLUSH_DW
End Workload

HCP Encoder Command Sequence

For a single frame encoding process (w/o multiple slices per frame), the command sequence is listed below. There are no states saved between frame encoded in the HCP. There should be no other commands or context switch within a group of PAK OBJECT Commands, representing a complete slice. HCP and MFX share the same VCS, but there is no common encoding and decoding command that can be executed in both pipes.

------ Per Frame Level Commands

HCP_PIPE_MODE_SELECT

HCP_SURFACE_STATE

HCP_PIPE_BUF_ADDR_STATE

HCP_IND_OBJ_BASE_ADDR_STATE

HCP_FQM_STATE - issue n number of times

HCP_QM_STATE - issue n number of times

HCP_PIC_STATE

------- Per Slice Level Commands (2 cases)

------- A Frame with only 1 Slice:

HCP_REF_IDX_STATE - set to provide L0 list for a P or B-Slice

HCP_REF_IDX_STATE - set to provide L1 list for a B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L0 of a P or B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L1 of a B-Slice

HCP_SLICE_STATE

HCP_PAK_INSERT_OBJECT - if header present at 1st slice start

    ------- A group of LCUs Per Slice

    HCP_PAK_OBJECT

    ...

HCP_PAK_INSERT_OBJECT - if tail present at frame end

MI_FLUSH - when the frame is done

------ A Frame with Multiple Slices:

HCP_REF_IDX_STATE - set to provide L0 list for a P or B-Slice

HCP_REF_IDX_STATE - set to provide L1 list for a B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L0 of a P or B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L1 of a B-Slice

HCP_SLICE_STATE

HCP_PAK_INSERT_OBJECT - if header present at 1st slice start of a frame

HCP_PAK_OBJECT - a group of LCUs for a slice or a frame

...

HCP_PAK_INSERT_OBJECT - if tail present at slice or frame end

HCP_REF_IDX_STATE - set to provide L0 list for a P or B-Slice

HCP_REF_IDX_STATE - set to provide L1 list for a B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L0 of a P or B-Slice

HCP_WEIGHTOFFSET_STATE Command - set to provide for L1 of a B-Slice

HCP_SLICE_STATE

HCP_PAK_INSERT_OBJECT - if header present at slice start

HCP_PAK_OBJECT - a group of LCUs for a slice or a frame

...

HCP_PAK_INSERT_OBJECT - if tail present at last slice end (frame end)

MI_FLUSH - when the frame is done

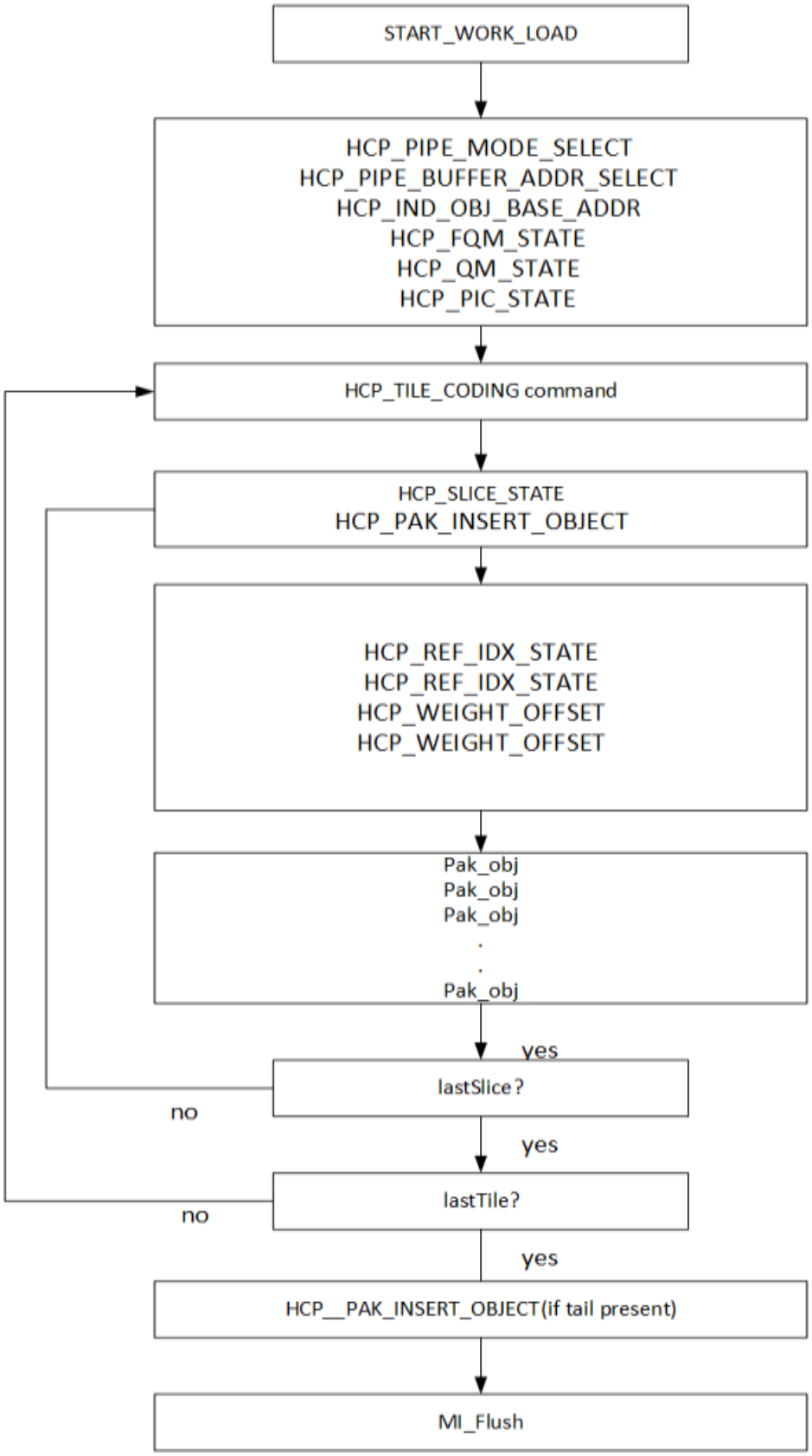
MFX_STITCH_OBJECT - a generic bitstream stitching command from MFX pipe

MI_FLUSH

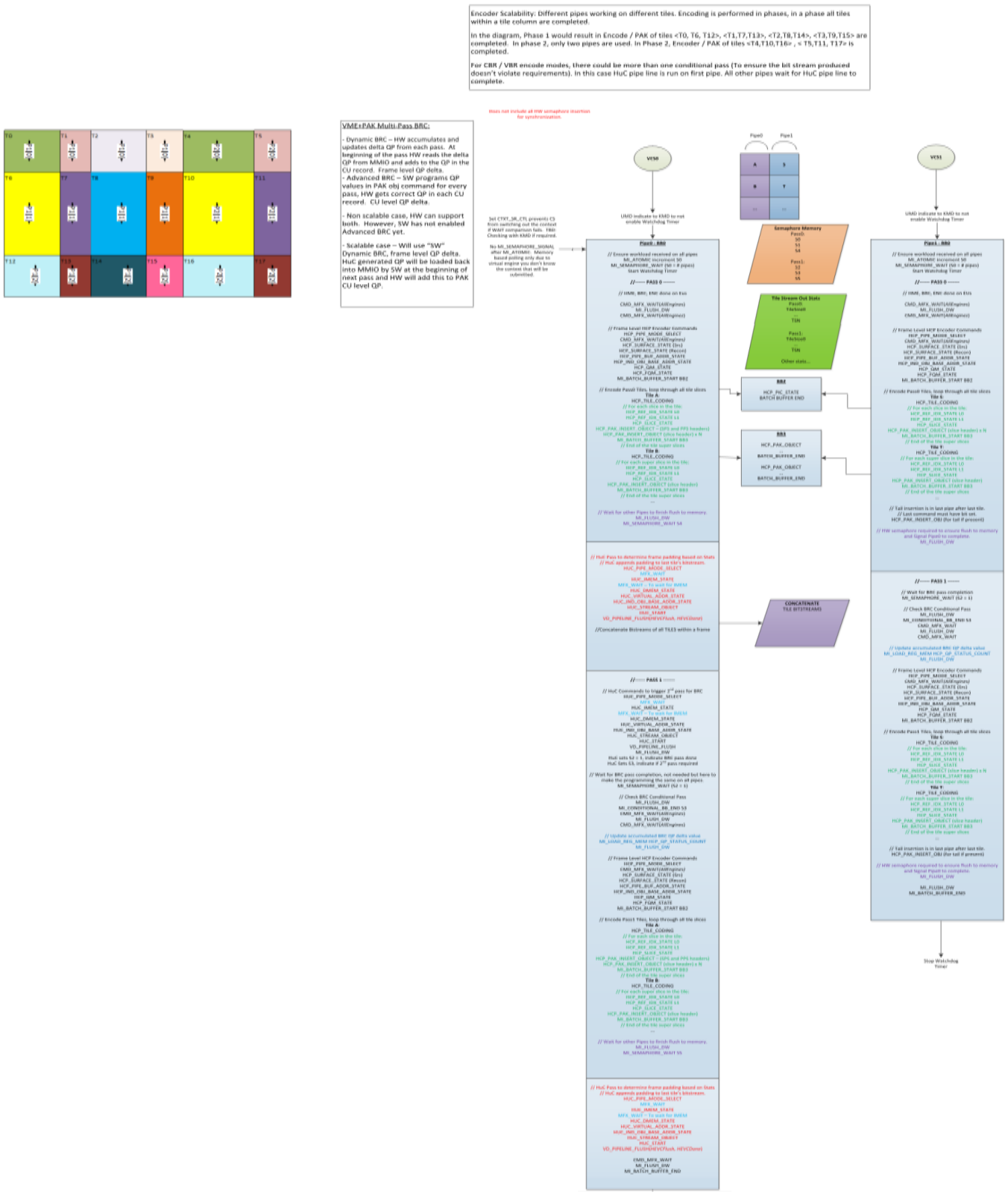
MI_FLUSH is not allowed between Slices. HEVC CABAC has simplified its operation from AVC. There is no longer a BSP_BUF_BASE_ADDR_STATE Command, as only a small local internal buffer is needed for BSP/BSE row store. THE HCP PAK_INSERT_OBJECT has been designed to support both inline and indirectly payload. Nevertheless, the MFX_STITCH_OBJECT command can still be used to stitch HEVC bitstreams together, and is run in the MFX pipe. No HEVC specific STITCH command is implemented. The SURFACE_STATE command for HEVC is redesigned and much simplified from that of MFX pipe.

Command Sequences with Tile Support

Single Pipe Mode- Following flow chart shows the command sequence when encoding frame using a single pipe(VDbox).


START_WORK_LOAD
HCP_PIPE_MODE_SELECT
HCP_PIPE_BUFFER_ADDR_SELECT
HCP_IND_OBJ_BASE_ADDR
HCP_FQM_STATE
HCP_QM_STATE
HCP_PIC_STATE
HCP_TILE_CODING command
HCP_SLICE_STATE
HCP_PAK_INSERT_OBJECT
HCP_REF_IDX_STATE
HCP_REF_IDX_STATE
HCP_WEIGHT_OFFSET
HCP_WEIGHT_OFFSET
Pak_obj
Pak_obj
Pak_obj
.
.
Pak_obj
yes
lastSlice?
no
yes
lastTile?
no
yes
HCP__PAK_INSERT_OBJECT(if tail present)
MI_Flush

Multiple Pipe Mode- When encoding a frame using multiple pipes, each pipe gets a single Tile Column or multiple Tile columns depending upon Number of tile columns to encode. Following flow chart shows commands sequence in a pipe (VDbox) when multiple pipes are used for encoding.



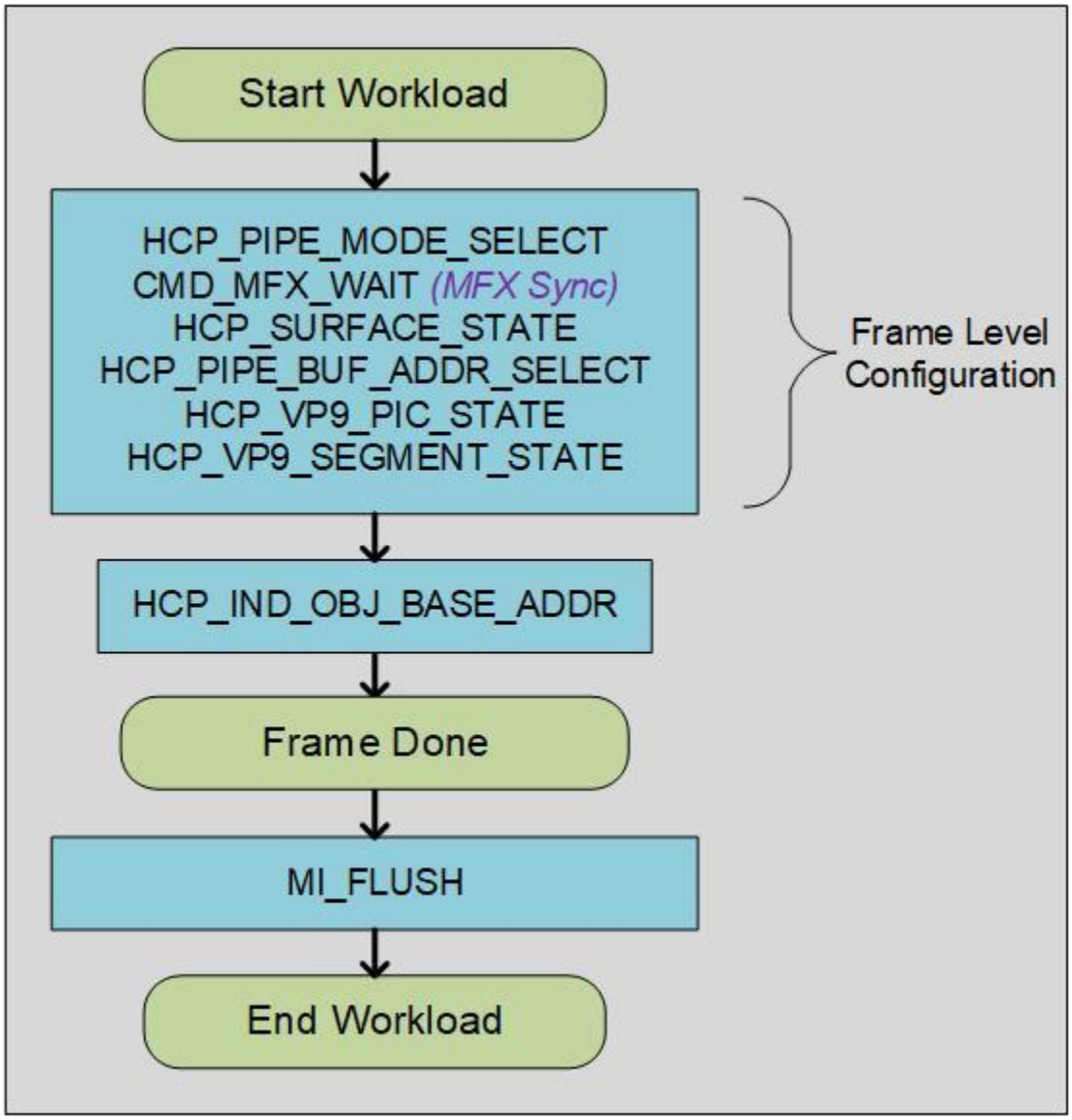
VP9 Decoder Command Sequence

VP9 decode programming is based upon a single frame decode. There are no states saved between frame decodes in the HCP. Once the bit stream DMA is configured with the HCP_IND_OBJ_BSD_OBJECT command, and the bit stream is presented to the HCP, the frame decode will begin.

VP9 Long Format Workflow Chart

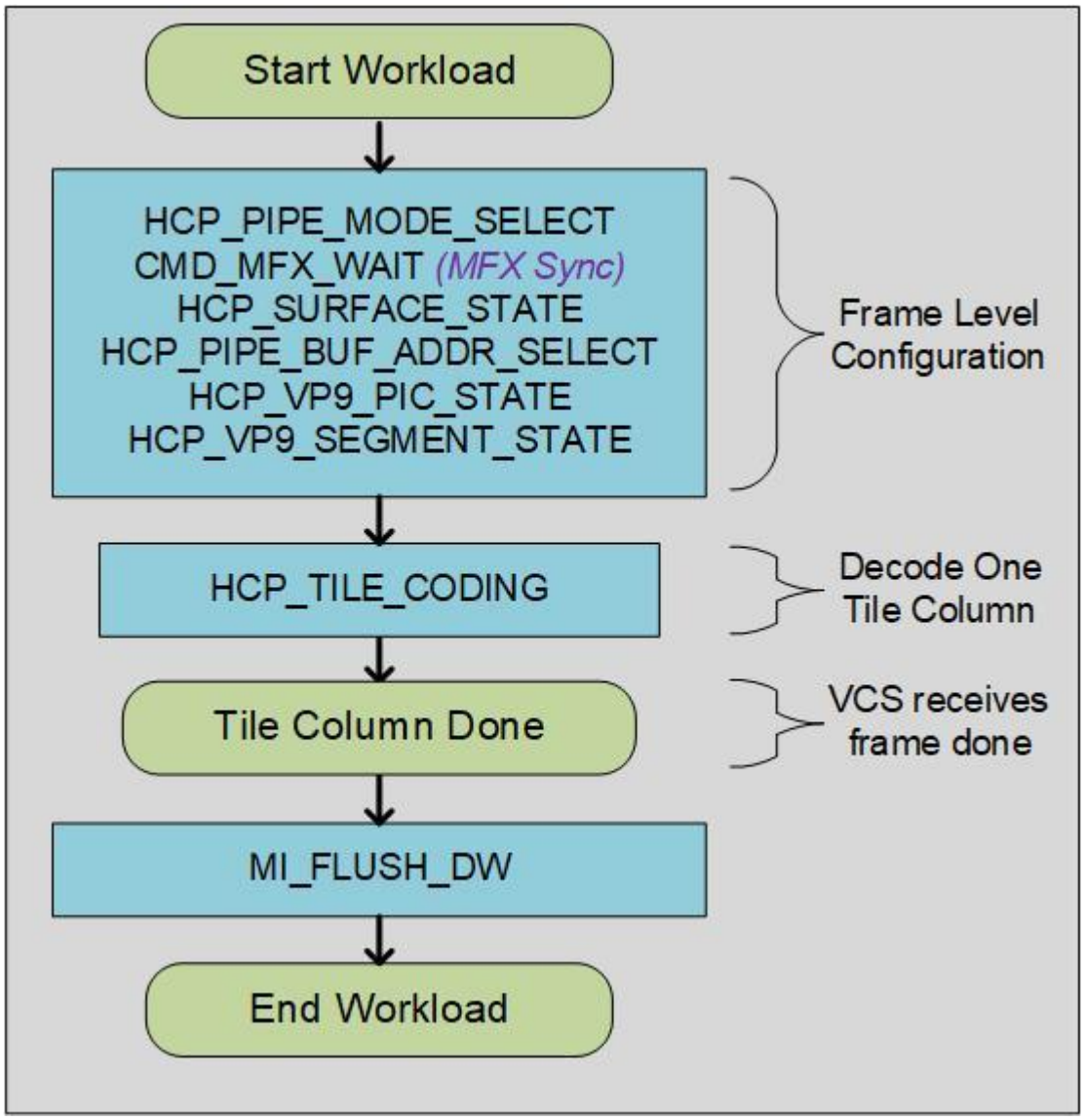
The following is the programming for single pipe decode.

The following programming also for scalable mode CABAC FE decode pass. There will be only one bitstream programming even if there are multiple tiles in the frame.



VP9 Scalable BE Only Workflow Chart

The following is the programming for scalable mode BE reconstruction pass (with multiple pipes). The following will be programmed on all the pipes.



Memory Address Attributes

This section defines the memory address attributes for the third DWord of the HCP command buffer address.

NOTE: The first DWord defines the lower address range and the second Dword defines the upper address range in the HCP command buffer address.

MemoryAddressAttributes

HCP Pipe Common Commands

The HCP Pipe Common Commands specify the HEVC Decoder pipeline level configuration.

Commands
HCP_PIPE_MODE_SELECT_VideoCS
HCP_SURFACE_STATE_VideoCS
HCP_PIPE_BUF_ADDR_STATE_VideoCS

Commands
HCP_IND_OBJ_BASE_ADDR_STATE_VideoCS
HCP_QM_STATE_VideoCS
HCP_FQM_STATE_VideoCS

Column Title1
HCP_TILE_CODING
VD_CONTROL_STATE

Buffer Size Requirements

HEVC Buffer Requirement

HEVC Buffer Size Requirements

The following table indicates the buffer size in CLs per LCU. For memory allocation, the size will be the CLs per LCU * the number of LCU horizontally (if it is line) or vertically (if is column)

Mode	HEVC					
Lcu_y_size in pixels (64 for 64x64)						
Lcu_size is in number of 4x4 in the LCU per column (16 for 64x64)	8 bit			>8 bit		
	4:2:0	4:2:2	4:4:4	4:2:0	4:2:2	4:4:4
Surface						
Deblock Line (per LCU)	[(2*lcu_size*128) + 511]/512	[(2*lcu_size*128) + 511]/512	[(2*lcu_size*1.5*128) + 511]/512	[(2*lcu_size*256) + 511]/512	[(2*lcu_size*256) + 511]/512	[(2*lcu_size*1.5*256) + 511]/512
Deblock Tile Line (per LCU)	**(** [(2*lcu_size*128) + 511]/512**) * 2**	**(** [(2*lcu_size*128) + 511]/512**) * 2**	**(** [(2*lcu_size*1.5*128) + 511]/512**) * 2**	**(** [(2*lcu_size*256) + 511]/512**) * 2**	**(** [(2*lcu_size*256) + 511]/512**) * 2**	**(** [(2*lcu_size*1.5*256) + 511]/512**) * 2**
Deblock Tile Column (per	**(** [(2*lcu_size*128	**(** [(2*lcu_size*1.5*128	**(** [(2*lcu_size*1.5*128	**(** [(2*lcu_size*256	**(** [(2*lcu_size*1.5*256	**(** [(2*lcu_size*1.5*256

LCU)	8 + 3*128) + 511]/512**) * 2**	8 + 3*128) + 511]/512**) * 2**	8 + 3*128) + 511]/512**) * 2**	6 + 3*256) + 511]/512**) * 2**	6 + 3*256) + 511]/512**) * 2**	6 + 3*256) + 511]/512**) * 2**
Top Right Motion Vector Tile Column (per LCU)	1	1	1	1	1	1
Motion Vector Line (per LCU)	LCU16/32 : 1 LCU64:2	LCU16/32 : 1 LCU64:2	LCU16/32 : 1 LCU64:2	LCU16/32 : 1 LCU64:2	LCU16/32 : 1 LCU64:2	LCU16/32 : 1 LCU64:2
Motion Vector Tile Line (per LCU)	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4
Right Motion Vector Tile Column(per LCU)	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4	LCU16/32 : 2 LCU64 : 4
Top Right Neighbor(per LCU)	1 CL	1 CL	LCU16 : 1 LCU32/64 : 2	LCU16 : 1 LCU32/64 : 2	LCU16 : 1 LCU32/64 : 2	LCU16 : 2 LCU32/64 : 3
HPR Left Recon Column(per LCU)	LCU16/32 : 1 LCU64 : 2	LCU16:1 LCU32:2 LCU64:3	LCU16:1 LCU32:2 LCU64:3	LCU16:1 LCU32:2 LCU64:4	LCU16:2 LCU32:3 LCU64:6	LCU16:2 LCU32:3 LCU64:6
HSF						
	8/10 bit			12 bit		
	4:2:0	4:2:2	4:4:4	4:2:0	4:2:2	4:4:4
Surface						
SAO Line (per LCU)	LCU16 : 2 LCU32 : 3 LCU64: 5	LCU16 : 2 LCU32 : 3 LCU64: 5	LCU16 : 3 LCU32 : 4 LCU64: 7	LCU16 : 2 LCU32 : 4 LCU64: 6	LCU16 : 2 LCU32 : 4 LCU64: 6	[LCU16 : 3 LCU32 : 5 LCU64: 8
SAO Tile Line (per LCU)	LCU16 : **4** LCU32 : **6** LCU64 : **10**	LCU16 : **4** LCU32 : **6** LCU64 : **10**	LCU16 : **6** LCU32 : **8** LCU64 : **14**	LCU16 : **4** LCU32 : **8** LCU64 : **12**	LCU16 : **4** LCU32 : **8** LCU64 : **12**	[LCU16 : **6** LCU32 : **10** LCU64 : **16**
SAO Tile Column (per LCU)	LCU16 : **8** LCU32 : **10** LCU64 : **18**	LCU16 : **10** LCU32 : **14** LCU64 : **24**	LCU16 : **10** LCU32 : **14** LCU64 : **24**	[LCU16 : **8** LCU32 : **10** LCU64 : **18**	LCU16 : **10** LCU32 : **14** LCU64 : **24**	LCU16 : **10** LCU32 : **14** LCU64 : **24**

The following table indicates the buffer size in CLs for the each row of frame or tile column.

Mode	HEVC					
	8 bit			>8 bit		
	4:2:0	4:2:2	4:4:4	4:2:0	4:2:2	4:4:4
Surface						

HSSE(per row of frame or tile colum n)	16*(frame/tile_ column width_in_lcu + 3)	16*(frame/tile_ column width_in_lcu + 3)	16*(frame/tile_column width_in_lcu + 3)	16*(frame/tile_ column width_in_lcu + 3)	16*(frame/tile_ column width_in_lcu + 3)	16*(frame/tile_ column width_in_lcu + 3)
HSAO(per row of frame or tile colum n)	(frame/tile_col umn width_in_lcu + 3)/4	(frame/tile column width_in_lcu + 3)/4	(frame(tile_column)_wi dth_in_lcu + 3)/4	(frame/tile_col umn width_in_lcu + 3)/4	(frame/tile_col umn width_in_lcu + 3)/4	(frame/tile_col umn width_in_lcu + 3)/4

VP9 Buffer Size Requirements

The following table indicates the buffer size in CLs per LCU. For memory allocation, the size will be the CLs per LCU * the number of LCU horizontally (if it is line) or vertically (if is column)

Mode	VP9					
	8 bits			> 8 bits		
	4:2:0	4:2:2	4:4:4	4:2:0	4:2:2	4:4:4
Deblock Line (per SB64)	18	18	27	36	36	54
Deblock Tile Line (per SB64)	18	18	27	36	36	54
Deblock Tile Column (per SB64)	17	25	25	34	50	50
Top Right Motion Vector Tile Column (per LCU)	NA	NA	NA	NA	NA	NA
Right Motion Vector Line	5	5	5	5	5	5
Right Motion Vector Tile Line	5	5	5	5	5	5
Right Motion Vector Tile Column	NA	NA	NA	NA	NA	NA
HPR Left Recon Column(per LCU)	2	3	3	4	6	6
Top Right Neighbor	1	1	1	1	1	1
HSAO	NA	NA	NA	NA	NA	NA
VP9 HVD Line Rowstore (per SB64)	1	1	1	1	1	1
VP9 HVD Tile Rowstore (per SB64)	1	1	1	1	1	1
VP9 Probability buffer (per frame)	32	32	32	32	32	32

The following table indicates the buffer size in CLs for each row of frame or tile column.

Mode	VP9					
	8 bit			> 8 bits		
	4:2:0	4:2:2	4:4:4	4:2:0	4:2:2	4:4:4
HSSE	32*(frame_width_in_sb64 + 3)	32*(frame_width_in_sb64 + 3)	32*(frame_width_in_sb64 + 3)	32*(frame_width_in_sb64 + 3)	32*(frame_width_in_sb64 + 3)	32*(frame_width_in_sb64 + 3)

The following table indicates the buffer size of each buffer for the whole frame. These data will be used across frames.

Mode	VP9					
	8 bit			> 8 bits		
	4:2:0	4:2:2	4:4:4	4:2:0	4:2:2	4:4:4
Current Motion Vector Temporal Buffer	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9
Collocated Motion Vector Temporal Buffer	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9	(num_width_in_SB * num_height_in_SB) * 9
VP9 Probability Buffer	32	32	32	32	32	32
VP9 Segment ID buffer (per frame)	(frame_width_in_sb64 * frame_height_in_sb64	(frame_width_in_sb64 * frame_height_in_sb64	(frame_width_in_sb64 * frame_height_in_sb64	(frame_width_in_sb64 * frame_height_in_sb64	(frame_width_in_sb64 * frame_height_in_sb64	(frame_width_in_sb64 * frame_height_in_sb64

Internal Media Rowstore table - If the internal Media Rowstore exists, driver should use the storage as the following table indicates.

This is for HEVC VMM setting. FrameWidth means frame width in picture for single pipe mode or Tile width for scalable mode.

HEVC				Enable Setting					Addr Setting				
ArrayType	Bitdepth	LCU Size	FrameWidth	Meta/MV	Deblock	SAO	HSAO	VDEnc	DAT	DF	SAO	HSAO	VDEnc
420/422	8/10/12	16	<= 4096	Y	Y	Y	Y	N	0	256	1280	2048	N/A
		32/64	<= 4096	Y	Y	Y	Y	Y	0	256	1280	1792	1824
	8/10/12	16	4097 - 8192	Y	Y	N	N	N	0	512	N/A	N/A	N/A
		32/64	4097 - 8192	Y	Y	N	N	Y	0	256	N/A	N/A	2304
444	8	16	<= 4096	Y	Y	Y	Y	Y	0	256	1024	1792	N/A
			4097 - 8192	Y	Y	N	Y	N	0	512	N/A	2048	N/A
	10		<= 4096	Y	Y	Y	N	N	0	256	1792	N/A	N/A
			4097 - 8192	Y	N	Y	Y	N	0	N/A	512	2048	N/A
	12		<= 4096	Y	Y	Y	N	N	0	256	1792	N/A	N/A
			4097 - 8192	Y	N	Y	Y	N	0	N/A	256	1792	N/A
	8	32/64	<= 4096	Y	Y	Y	Y	Y	0	256	1024	1536	1568
			4097 - 8192	Y	Y	N	Y	Y	0	512	N/A	2048	2112
	10		<= 4096	Y	Y	Y	Y	Y	0	256	1792	2304	2336
			4097 - 8192	Y	N	Y	Y	Y	0	N/A	512	1536	1600
	12		<= 4096	Y	Y	Y	Y	Y	0	128	1664	2304	2336
			4097 - 8192	Y	N	Y	Y	Y	0	N/A	256	1536	1600

The following table is for VP9 VMM setting. FrameWidth means frame width in picture for Single Pipe mode or Tile Width for Scalable Mode

VP9				Enable Setting				Addr Setting			
ArrayType	Bitdepth	LCU Size	FrameWidth	HVD	Meta/MV	Deblock	VDENC	HVD	Meta/MV	Deblock	VDEnc
420	8	64	<= 4096	Y	Y	Y	Y	0	64	384	1536
		64	4097 - 8192	N	N	Y	Y	N/A	N/A	0	2304
	10/12	64	<= 4096	Y	N	Y	Y	0	N/A	64	2368
		64	4097 - 8192	Y	Y	N	Y	0	128	N/A	768
422	8	64	<= 4096	Y	Y	Y	Y	0	64	384	1536
		64	4097 - 8192	N	N	Y	Y	N/A	N/A	0	2304
	10/12	64	<= 4096	N	N	Y	N	N/A	N/A	0	N/A
		64	4097 - 8192	Y	Y	N	Y	0	128	N/A	768
444	8	64	<= 4096	Y	Y	Y	Y	0	64	384	2112
		64	4097 - 8192	Y	Y	N	Y	0	128	N/A	768
	10/12	64	<= 2048	Y	Y	Y	Y	0	32	192	1920
		64	2049 - 4096	Y	Y	N	Y	0	128	N/A	768
		64	4097 - 8192	Y	Y	N	Y	0	128	N/A	768

VP9 Common Commands

Commands
HCP_PIPE_MODE_SELECT
HCP_SURFACE_STATE
HCP_PIPE_BUF_ADDR_STATE
HCP_IND_OBJ_BASE_ADDR_STATE
HCP_VP9_SEGMENT_STATE
HCP_VP9_PIC_STATE

HCP Common Commands

HCP Common Commands
HCP_PIC_STATE
HCP_TILE_STATE
HCP_REF_IDX_STATE
HCP_WEIGHTOFFSET_STATE
HCP_SLICE_STATE
HEVC_VP9_RDOQ_STATE
HCP_BSD_OBJECT
HCP_PAK_OBJECT
HCP_PAK_INSERT_OBJECT
HCP_PALETTE_INITIALIZER_STATE

HCP and VP9 Commands

HCP_BSD_OBJECT (triggers HW start)

HCP_VP9_PAK_OBJECT

Tile Size and CU Stream-out Records

TileSize Record for both HEVC and VP9 Codecs

Fields	Bits	Notes
Address	DW0-1[63:0]	Bitstream start address; baseAddr + Tile offset. Used for stitching purpose in scalability mode. MBZ in single pipe mode.
Length	DW2[31:0]	Bitstream length per tile; Includes header in first tile and tail in last tile. Used for stitching purpose in scalability mode. MBZ in single pipe mode.
Tile Size	DW3[31:0]	Tile Size(no header) used by HW for back annotation. Also HuC uses for BRC purpose in scalability mode for both hevc/vp9. MBZ in HEVC single pipe mode.
AddressOffset	DW4[31:0]	Cacheline Address to be Modified. Used by HW for back

Fields	Bits	Notes
		annotation. MBZ in HEVC mode.
Offset	DW5[5:0]	Byte offset to be Modified. Used by HW for back annotation. MBZ in HEVC mode.
Reserved	DW5[31:6]	MBZ
HCP_BITSTREAM_SE_BITCOUNT_FRAME	DW6[31:0]	Bitstream size for Syntax element per Tile (see the MMIO register for details). valid only for HEVC in scalability mode and MBZ for VP9
HCP_CABAC_BIN_COUNT_FRAME	DW7[31:0]	Bitstream size for Bin count per Tile (see the MMIO register for details). valid only in scalability mode
Reserved	DW8[31:0]	MBZ
HCP_IMAGE_STATUS_CONTROL	DW9[31:0]	Image Status Mask Control(see the MMIO register for details). Only valid fields are Total-NumPass[11:8](hevc/vp9) and LCUbitCountViolate[0](hevc only).The rest of the fields are MBZ. valid only in scalability mode
HCP_QP_STATUS_COUNT	DW10[31:0]	QP Status count (see the MMIO register for details). valid only for HEVC in scalability mode and MBZ for VP9
HCP_SLICE_COUNT	DW11[31:0]	Slice count (see the MMIO register for details). valid only for HEVC in scalability mode and MBZ for VP9
Reserved	DW12-15	MBZ

HEVC: Streamout0 cacheline is composed of 4 quarter cachelines, each containing information on CU skip flag, coding block flag for the TUs in a PU, residual/coefficient bit count for a PU, total bit count for CU, SB exceed limit flag. A typical streamout0 cacheline, therefore, has information on statistics for 4 PUs and Super Block exceed limit flag.

Pak pipeline streamout enable bit, set by HCP_PIPE_MODE_SELECT command, enables or disables the streamout.

Programming Note				
Context:	CU level statistics			
Level	**Field**	**Width**	**Cacheline**	**Comment**
PU	PU Skip Flag	1	qcacheline[0]	Packed in Quarter Cacheline in PU format
SB	SB exceed limit	1	qcacheline[1]	Packed in Quarter Cacheline in PU format (valid on last PU of SB)
	Reserved	14	qcacheline[15:2	Reserved
PU	TU CBF Y/U/V	48	qcacheline[63:16]	Packed in Quarter Cacheline in PU format
PU	PU Coefficient Bit Count (Only residual)	18	qcacheline[81:64]	Packed in Quarter Cacheline in PU format
PU	PU Bit Count (all PU Syntax)	18	qcacheline[113:96]	Packed in Quarter Cacheline in PU format
	Reserved	14	qcacheline[127:114]	Reserved

Programming Note	
Context:	.

HEVC Streamout 1: Per Tile Quarter Cacheline

Level	Field	Width	Cacheline	Comment
Tile	Tile Bit Count (header + data + tail)	32	cacheline[31:0]	
	Reserved(MBZ)	32	cacheline[63:32]	
	TilePositionX[15:0]	16	cacheline[79:64]	
	TilePositionY[15:0]	16	cacheline[95:80]	
	Reserved(MBZ)	32	cacheline[127:96]	

VP9: CU statistics record (individual PUs per record down to 8x8 only)

Fields	Bits	
Skip	3:0	Indicates Skip flag Group 4 4x4s -> 4 bits
InterMode	11:4	InterMode: 0 NEARESTMV, 1 NEARMV, 2 ZEROMV, 3NEWMV Group 4 4x4s total 8 bits
Reserved	15:12	
NZ coeff count	28:16	Number of non-zero coeffs; sum of YUV, 13bits
Reserved	31:29	
NumBitsforCoeffs	47:32	Number of Bits for coefficients per block, 16bits
NumBitsforBlock	63:48	Number of Bits in block

VP9 Stream-in/Stream-out Probability Table

In Encoder mode, one table will be streamed out. SW can use it as the current frame probability update and use as the probability table for future frame. This stream-out table will become the stream-in probability table for current frame or future frame. The stream-out and stream-in probability tables have exactly the same format.

Alignment	New Offset	# Bytes	Description	Keyframe defaults	Inter frame defaults					Capture At DV_CNT	State counter EBB Address	Coefficient counter EBB Address							
												4x4 (KF)	4x4 (INTER)	8x8 (KF)	8x8 (INTER)	16x16 (KF)	16x16 (INTER)	32x32 (KF)	32x32 (INTER)
CL aligned	0	1	tx_probs_8x8 [0] [0..0]	100	100	0	0	0	0	MODE COUNTERS (counts tx)	0-17								
	1	1	tx_probs_8x8 [1] [0..0]	66	66	0	1	1											

Stream-in formats for creating compressed header

The following memory surfaces are input to PAK for Compressed Header coding

i. Prob Diff Surface

In Probability Diff Surface, there are 1805 8-bit Probability Diffs. Each of them corresponding to a Probability Diff in Compressed Header syntax. Although, for a given compressed header, not all the Probability Diff would be coded (depends on update flag), Probability Diff Surface is fully populated with 1805 entries (1805*8 / 512 = 29 cachelines). The 1805 8-bit Probability Diffs are expected to follow Compressed Header syntax order and fully packed.

ii. Compressed Header Syntax Surface

Each of the Compressed header Coding element (described in (2)) is represented by a 4-bit field. These 4-bit fields follows Compressed Header Syntax. Each of the field has a valid, Bin_probDiff_select, Prob_select, Bin as described in the table below.

	Description
Valid	Set to 1 if this is a valid Bin OR ProbabilityDiff field to code; Set to 0 to skip coding this field
Bin_ProbDiff_select	Set to 1 if Current field is a Bin (corresponding Prob, Bin are indicated by next 2 bits); Set to 0 if Current field is Probability Diff (probability diff to be coded is located in probability surface - ReMap)
Prob_Select	If current field is Bin, set to 1 if prob is 252; set to 0 if prob is 128
Bin	if current field is Bin, this is Bin value to be encoded

Compressed Header Syntax Surface is a fixed length surface. For syntax that should not be coded, valid bit should be set to 0. Total length of Compressed Header syntax Surface has 4033 Coding elements (16132 bits in 32 cachelines):

1805 Prob Diff and Prob Update flag

4 is_coeff_updated flag (per 4x4, 8x8, 16x16, 32x32)

5 control fields (MIN (tx_mode, ALLOW_32x32), tx_mode == TX_MODE_SELECT, use_compound_pred, use_hybrid_pred)

VP9 PAK Quant Lookup Tables

	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
Qindex	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
0	4	4	4	4	4	4	1	1	1	1	1	1	32768	32768	32768	32768	32768	32768
1	8	8	9	9	12	13	2	2	2	2	2	2	32768	32768	29127	29127	21845	20164
2	8	9	10	11	18	19	2	2	2	2	3	3	32768	29127	26214	23831	29127	27594
3	9	10	13	13	25	27	2	2	2	2	3	3	29127	26214	20164	20164	20971	19418
4	10	11	15	16	33	35	2	2	2	3	4	4	26214	23831	17476	32768	31775	29959
5	11	12	17	18	41	44	2	2	3	3	4	4	23831	21845	30840	29127	25575	23831
6	12	13	20	21	50	54	2	2	3	3	4	4	21845	20164	26214	24966	20971	19418
7	12	14	22	24	60	64	2	2	3	3	4	5	21845	18724	23831	21845	17476	32768
8	13	15	25	27	70	75	2	2	3	3	5	5	20164	17476	20971	19418	29959	27962
9	14	16	28	30	80	87	2	3	3	3	5	5	18724	32768	18724	17476	26214	24105
10	15	17	31	33	91	99	2	3	3	4	5	5	17476	30840	16912	31775	23045	21183
11	16	18	34	37	103	112	3	3	4	4	5	5	32768	29127	30840	28339	20360	18724
12	17	19	37	40	115	126	3	3	4	4	5	5	30840	27594	28339	26214	18236	16644
13	18	20	40	44	127	139	3	3	4	4	5	6	29127	26214	26214	23831	16513	30174
14	19	21	43	48	140	154	3	3	4	4	6	6	27594	24966	24385	21845	29959	27235
15	19	22	47	51	153	168	3	3	4	4	6	6	27594	23831	22310	20560	27413	24966
16	20	23	50	55	166	183	3	3	4	4	6	6	26214	22795	20971	19065	25266	22919
17	21	24	53	59	180	199	3	3	4	4	6	6	24966	21845	19784	17772	23301	21076
18	22	25	57	63	194	214	3	3	4	4	6	6	23831	20971	18396	16644	21620	19599
19	23	26	60	67	208	230	3	3	4	5	6	6	22795	20164	17476	31300	20164	18236
20	24	27	64	71	222	247	3	3	5	5	6	6	21845	19418	32768	29537	18893	16980
21	25	28	68	75	237	263	3	3	5	5	6	7	20971	18724	30840	27962	17697	31895
22	26	29	71	79	251	280	3	3	5	5	6	7	20164	18078	29537	26546	16710	29959
23	26	30	75	83	266	297	3	3	5	5	7	7	20164	17476	27962	25266	31536	28244
24	27	31	78	88	281	314	3	3	5	5	7	7	19418	16912	26886	23831	29852	26715
25	28	32	82	92	296	331	3	4	5	5	7	7	18724	32768	25575	22795	28339	25343
26	29	33	86	96	312	349	3	4	5	5	7	7	18078	31775	24385	21845	26886	24036
27	30	34	90	100	327	366	3	4	5	5	7	7	17476	30840	23301	20971	25653	22919

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
28	31	35	93	105	343	384	3	4	5	5	7	7	16912	29959	22550	19972	24456	21845
29	32	36	97	109	358	402	4	4	5	5	7	7	32768	29127	21620	19239	23431	20867
30	32	37	101	114	374	420	4	4	5	5	7	7	32768	28339	20763	18396	22429	19972
31	33	38	105	118	390	438	4	4	5	5	7	7	31775	27594	19972	17772	21509	19152
32	34	39	109	122	405	456	4	4	5	5	7	7	30840	26886	19239	17189	20712	18396
33	35	40	113	127	421	475	4	4	5	5	7	7	29959	26214	18558	16513	19925	17660
34	36	41	116	131	437	493	4	4	5	6	7	7	29127	25575	18078	32017	19195	17015
35	37	42	120	136	453	511	4	4	5	6	7	7	28339	24966	17476	30840	18517	16416
36	38	43	124	140	469	530	4	4	5	6	7	8	27594	24385	16912	29959	17886	31655
37	38	44	128	145	484	548	4	4	6	6	7	8	27594	23831	32768	28926	17331	30615
38	39	45	132	149	500	567	4	4	6	6	7	8	26886	23301	31775	28149	16777	29589
39	40	46	136	154	516	586	4	4	6	6	8	8	26214	22795	30840	27235	32513	28630
40	41	47	140	158	532	604	4	4	6	6	8	8	25575	22310	29959	26546	31536	27776
41	42	48	143	163	548	623	4	4	6	6	8	8	24966	21845	29330	25731	30615	26929
42	43	49	147	168	564	642	4	4	6	6	8	8	24385	21399	28532	24966	29746	26132
43	43	50	151	172	580	660	4	4	6	6	8	8	24385	20971	27776	24385	28926	25420
44	44	51	155	177	596	679	4	4	6	6	8	8	23831	20560	27060	23696	28149	24708
45	45	52	159	181	611	698	4	4	6	6	8	8	23301	20164	26379	23172	27458	24036
46	46	53	163	186	627	716	4	4	6	6	8	8	22795	19784	25731	22550	26757	23431
47	47	54	166	190	643	735	4	4	6	6	8	8	22310	19418	25266	22075	26092	22826
48	48	55	170	195	659	753	4	4	6	6	8	8	21845	19065	24672	21509	25458	22280
49	48	56	174	199	674	772	4	4	6	6	8	8	21845	18724	24105	21076	24892	21732
50	49	57	178	204	690	791	4	4	6	6	8	8	21399	18396	23563	20560	24314	21210
51	50	58	182	208	706	809	4	4	6	6	8	8	20971	18078	23045	20164	23763	20738
52	51	59	185	213	721	828	4	4	6	6	8	8	20560	17772	22671	19691	23269	20262
53	52	60	189	217	737	846	4	4	6	6	8	8	20164	17476	22192	19328	22764	19831
54	53	61	193	222	752	865	4	4	6	6	8	8	19784	17189	21732	18893	22310	19395
55	53	62	197	226	768	884	4	4	6	6	8	8	19784	16912	21290	18558	21845	18978
56	54	63	200	231	783	902	4	4	6	6	8	8	19418	16644	20971	18157	21426	18600
57	55	64	204	235	798	920	4	5	6	6	8	8	19065	32768	20560	17848	21024	18236
58	56	65	208	240	814	939	4	5	6	6	8	8	18724	32263	20164	17476	20610	17867
59	57	66	212	244	829	957	4	5	6	6	8	8	18396	31775	19784	17189	20237	17531
60	57	67	215	249	844	976	4	5	6	6	8	8	18396	31300	19508	16844	19878	17189
61	58	68	219	253	859	994	4	5	6	6	8	8	18078	30840	19152	16578	19531	16878
62	59	69	223	258	874	1012	4	5	6	7	8	8	17772	30393	18808	32513	19195	16578
63	60	70	226	262	889	1030	4	5	6	7	8	9	17476	29959	18558	32017	18872	32577
64	61	71	230	267	904	1049	4	5	6	7	8	9	17189	29537	18236	31418	18558	31987
65	62	72	233	271	919	1067	4	5	6	7	8	9	16912	29127	18001	30954	18255	31447

	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
Qindex	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
66	62	73	237	275	934	1085	4	5	6	7	8	9	16912	28728	17697	30504	17962	30925
67	63	74	241	280	949	1103	4	5	6	7	8	9	16644	28339	17403	29959	17678	30421
68	64	75	244	284	964	1121	5	5	6	7	8	9	32768	27962	17189	29537	17403	29932
69	65	76	248	289	978	1139	5	5	6	7	8	9	32263	27594	16912	29026	17154	29459
70	66	77	251	293	993	1157	5	5	6	7	8	9	31775	27235	16710	28630	16895	29001
71	66	78	255	297	1008	1175	5	5	6	7	8	9	31775	26886	16448	28244	16644	28556
72	67	79	259	302	1022	1193	5	5	7	7	8	9	31300	26546	32388	27776	16416	28126
73	68	80	262	306	1037	1211	5	5	7	7	9	9	30840	26214	32017	27413	32357	27708
74	69	81	266	311	1051	1229	5	5	7	7	9	9	30393	25890	31536	26973	31926	27302
75	70	82	269	315	1065	1246	5	5	7	7	9	9	29959	25575	31184	26630	31506	26929
76	70	83	273	319	1080	1264	5	5	7	7	9	9	29959	25266	30727	26296	31068	26546
77	71	84	276	324	1094	1282	5	5	7	7	9	9	29537	24966	30393	25890	30671	26173
78	72	85	280	328	1108	1299	5	5	7	7	9	9	29127	24672	29959	25575	30283	25830
79	73	86	283	332	1122	1317	5	5	7	7	9	9	28728	24385	29641	25266	29905	25477
80	74	87	287	337	1136	1335	5	5	7	7	9	9	28339	24105	29228	24892	29537	25134
81	74	88	290	341	1151	1352	5	5	7	7	9	9	28339	23831	28926	24600	29152	24818
82	75	89	293	345	1165	1370	5	5	7	7	9	9	27962	23563	28630	24314	28802	24492
83	76	90	297	349	1179	1387	5	5	7	7	9	9	27594	23301	28244	24036	28460	24192
84	77	91	300	354	1192	1405	5	5	7	7	9	9	27235	23045	27962	23696	28149	23882
85	78	92	304	358	1206	1422	5	5	7	7	9	9	26886	22795	27594	23431	27822	23596
86	78	93	307	362	1220	1440	5	5	7	7	9	9	26886	22550	27324	23172	27503	23301
87	79	94	310	367	1234	1457	5	5	7	7	9	9	26546	22310	27060	22857	27191	23029
88	80	95	314	371	1248	1474	5	5	7	7	9	9	26214	22075	26715	22610	26886	22764
89	81	96	317	375	1261	1491	5	5	7	7	9	9	25890	21845	26462	22369	26609	22504
90	81	97	321	379	1275	1509	5	5	7	7	9	9	25890	21620	26132	22133	26317	22236
91	82	98	324	384	1288	1526	5	5	7	7	9	9	25575	21399	25890	21845	26051	21988
92	83	99	327	388	1302	1543	5	5	7	7	9	9	25266	21183	25653	21620	25771	21746
93	84	100	331	392	1315	1560	5	5	7	7	9	9	24966	20971	25343	21399	25516	21509
94	85	101	334	396	1329	1577	5	5	7	7	9	9	24672	20763	25115	21183	25247	21277
95	85	102	337	401	1342	1595	5	5	7	7	9	9	24672	20560	24892	20919	25003	21037
96	87	104	343	409	1368	1627	5	5	7	7	9	9	24105	20164	24456	20510	24528	20623
97	88	106	350	417	1393	1660	5	5	7	7	9	9	23831	19784	23967	20116	24087	20213
98	90	108	356	425	1419	1693	5	5	7	7	9	9	23301	19418	23563	19737	23646	19819
99	92	110	362	433	1444	1725	5	5	7	7	9	9	22795	19065	23172	19373	23237	19451
100	93	112	369	441	1469	1758	5	5	7	7	9	9	22550	18724	22733	19021	22841	19086
101	95	114	375	449	1494	1791	5	5	7	7	9	9	22075	18396	22369	18682	22459	18735
102	96	116	381	458	1519	1824	5	5	7	7	9	9	21845	18078	22017	18315	22089	18396
103	98	118	387	466	1544	1856	5	5	7	7	9	9	21399	17772	21675	18001	21732	18078

Qindex	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
104	99	120	394	474	1569	1889	5	5	7	7	9	9	21183	17476	21290	17697	21385	17763
105	101	122	400	482	1594	1922	5	5	7	7	9	9	20763	17189	20971	17403	21050	17458
106	102	124	406	490	1618	1954	5	5	7	7	9	9	20560	16912	20661	17119	20738	17172
107	104	126	412	498	1643	1987	5	5	7	7	9	9	20164	16644	20360	16844	20422	16886
108	105	128	418	506	1668	2020	5	6	7	7	9	9	19972	32768	20068	16578	20116	16611
109	107	130	424	514	1692	2052	5	6	7	8	9	10	19599	32263	19784	32640	19831	32704
110	108	132	430	523	1717	2085	5	6	7	8	9	10	19418	31775	19508	32078	19542	32186
111	110	134	436	531	1741	2118	5	6	7	8	9	10	19065	31300	19239	31595	19273	31685
112	111	136	442	539	1765	2150	5	6	7	8	9	10	18893	30840	18978	31126	19011	31213
113	113	138	448	547	1789	2183	5	6	7	8	9	10	18558	30393	18724	30671	18755	30741
114	114	140	454	555	1814	2216	5	6	7	8	9	10	18396	29959	18477	30229	18497	30283
115	116	142	460	563	1838	2248	5	6	7	8	9	10	18078	29537	18236	29799	18255	29852
116	117	144	466	571	1862	2281	5	6	7	8	9	10	17924	29127	18001	29382	18020	29420
117	118	146	472	579	1885	2313	5	6	7	8	9	10	17772	28728	17772	28976	17800	29013
118	120	148	478	588	1909	2346	5	6	7	8	9	10	17476	28339	17549	28532	17576	28605
119	121	150	484	596	1933	2378	5	6	7	8	9	10	17331	27962	17331	28149	17358	28220
120	123	152	490	604	1957	2411	5	6	7	8	9	10	17050	27594	17119	27776	17145	27834
121	125	155	499	616	1992	2459	5	6	7	8	9	10	16777	27060	16810	27235	16844	27291
122	127	158	507	628	2027	2508	5	6	7	8	9	10	16513	26546	16545	26715	16553	26757
123	129	161	516	640	2061	2556	6	6	8	8	10	10	32513	26051	32513	26214	32561	26255
124	131	164	525	652	2096	2605	6	6	8	8	10	10	32017	25575	31956	25731	32017	25761
125	134	167	533	664	2130	2653	6	6	8	8	10	10	31300	25115	31476	25266	31506	25295
126	136	170	542	676	2165	2701	6	6	8	8	10	10	30840	24672	30954	24818	30997	24845
127	138	173	550	688	2199	2750	6	6	8	8	10	10	30393	24244	30504	24385	30517	24403
128	140	176	559	700	2233	2798	6	6	8	8	10	10	29959	23831	30012	23967	30053	23984
129	142	179	567	713	2267	2847	6	6	8	8	10	10	29537	23431	29589	23530	29602	23571
130	144	182	576	725	2300	2895	6	6	8	8	10	10	29127	23045	29127	23140	29177	23180
131	146	185	584	737	2334	2943	6	6	8	8	10	10	28728	22671	28728	22764	28752	22802
132	148	188	592	749	2367	2992	6	6	8	8	10	10	28339	22310	28339	22399	28351	22429
133	150	191	601	761	2400	3040	6	6	8	8	10	10	27962	21959	27915	22046	27962	22075
134	152	194	609	773	2434	3088	6	6	8	8	10	10	27594	21620	27548	21704	27571	21732
135	154	197	617	785	2467	3137	6	6	8	8	10	10	27235	21290	27191	21372	27202	21392
136	156	200	625	797	2499	3185	6	6	8	8	10	10	26886	20971	26843	21050	26854	21070
137	158	203	634	809	2532	3234	6	6	8	8	10	10	26546	20661	26462	20738	26504	20751
138	161	207	644	825	2575	3298	6	6	8	8	10	10	26051	20262	26051	20336	26061	20348
139	164	211	655	841	2618	3362	6	6	8	8	10	10	25575	19878	25614	19949	25633	19960
140	166	215	666	857	2661	3426	6	6	8	8	10	10	25266	19508	25191	19576	25219	19588
141	169	219	676	873	2704	3491	6	6	8	8	10	10	24818	19152	24818	19217	24818	19223

	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
Qindex	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
142	172	223	687	889	2746	3555	6	6	8	8	10	10	24385	18808	24420	18872	24438	18877
143	174	227	698	905	2788	3619	6	6	8	8	10	10	24105	18477	24036	18538	24070	18543
144	177	231	708	922	2830	3684	6	6	8	8	10	10	23696	18157	23696	18196	23713	18216
145	180	235	718	938	2872	3748	6	6	8	8	10	10	23301	17848	23366	17886	23366	17905
146	182	239	729	954	2913	3812	6	6	8	8	10	10	23045	17549	23014	17586	23037	17604
147	185	243	739	970	2954	3876	6	6	8	8	10	10	22671	17260	22702	17296	22717	17313
148	187	247	749	986	2995	3941	6	6	8	8	10	10	22429	16980	22399	17015	22406	17028
149	190	251	759	1002	3036	4005	6	6	8	8	10	10	22075	16710	22104	16743	22104	16756
150	192	255	770	1018	3076	4069	6	6	8	8	10	10	21845	16448	21788	16480	21816	16492
151	195	260	782	1038	3127	4149	6	7	8	9	10	11	21509	32263	21454	32326	21461	32349
152	199	265	795	1058	3177	4230	6	7	8	9	10	11	21076	31655	21103	31714	21123	31729
153	202	270	807	1078	3226	4310	6	7	8	9	10	11	20763	31068	20789	31126	20802	31141
154	205	275	819	1098	3275	4390	6	7	8	9	10	11	20460	30504	20485	30559	20491	30573
155	208	280	831	1118	3324	4470	6	7	8	9	10	11	20164	29959	20189	30012	20189	30026
156	211	285	844	1138	3373	4550	6	7	8	9	10	11	19878	29433	19878	29485	19895	29498
157	214	290	856	1158	3421	4631	6	7	8	9	10	11	19599	28926	19599	28976	19616	28982
158	217	295	868	1178	3469	4711	6	7	8	9	10	11	19328	28435	19328	28484	19345	28490
159	220	300	880	1198	3517	4791	6	7	8	9	10	11	19065	27962	19065	28008	19081	28014
160	223	305	891	1218	3565	4871	6	7	8	9	10	11	18808	27503	18829	27548	18824	27554
161	226	311	906	1242	3621	4967	6	7	8	9	10	11	18558	26973	18517	27016	18533	27021
162	230	317	920	1266	3677	5064	6	7	8	9	10	11	18236	26462	18236	26504	18250	26504
163	233	323	933	1290	3733	5160	6	7	8	9	10	11	18001	25970	17982	26011	17977	26011
164	237	329	947	1314	3788	5256	6	7	8	9	10	11	17697	25497	17716	25536	17716	25536
165	240	335	961	1338	3843	5352	6	7	8	9	10	11	17476	25040	17458	25078	17462	25078
166	243	341	975	1362	3897	5448	6	7	8	9	10	11	17260	24600	17207	24636	17220	24636
167	247	347	988	1386	3951	5544	6	7	8	9	10	11	16980	24174	16980	24209	16985	24209
168	250	353	1001	1411	4005	5641	6	7	8	9	10	11	16777	23763	16760	23780	16756	23793
169	253	359	1015	1435	4058	5737	6	7	8	9	10	11	16578	23366	16529	23382	16537	23395
170	257	366	1030	1463	4119	5849	7	7	9	9	11	11	32640	22919	32577	22935	32585	22947
171	261	373	1045	1491	4181	5961	7	7	9	9	11	11	32140	22489	32109	22504	32101	22515
172	265	380	1061	1519	4241	6073	7	7	9	9	11	11	31655	22075	31625	22089	31647	22100
173	269	387	1076	1547	4301	6185	7	7	9	9	11	11	31184	21675	31184	21690	31206	21700
174	272	394	1090	1575	4361	6297	7	7	9	9	11	11	30840	21290	30783	21304	30776	21314
175	276	401	1105	1603	4420	6410	7	7	9	9	11	11	30393	20919	30366	20932	30366	20938
176	280	408	1120	1631	4479	6522	7	7	9	9	11	11	29959	20560	29959	20572	29966	20579
177	284	416	1137	1663	4546	6650	7	7	9	9	11	11	29537	20164	29511	20177	29524	20183
178	288	424	1153	1695	4612	6778	7	7	9	9	11	11	29127	19784	29101	19796	29101	19801
179	292	432	1170	1727	4677	6906	7	7	9	9	11	11	28728	19418	28679	19429	28697	19434

	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
Qindex	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
180	296	440	1186	1759	4742	7034	7	7	9	9	11	11	28339	19065	28292	19075	28304	19081
181	300	448	1202	1791	4807	7162	7	7	9	9	11	11	27962	18724	27915	18735	27921	18740
182	304	456	1218	1823	4871	7290	7	7	9	9	11	11	27594	18396	27548	18406	27554	18411
183	309	465	1236	1859	4942	7435	7	7	9	9	11	11	27147	18040	27147	18049	27158	18052
184	313	474	1253	1895	5013	7579	7	7	9	9	11	11	26800	17697	26779	17706	26773	17709
185	317	483	1271	1931	5083	7723	7	7	9	9	11	11	26462	17367	26400	17376	26405	17378
186	322	492	1288	1967	5153	7867	7	7	9	9	11	11	26051	17050	26051	17058	26046	17060
187	326	501	1306	2003	5222	8011	7	7	9	9	11	11	25731	16743	25692	16752	25702	16754
188	330	510	1323	2039	5291	8155	7	7	9	9	11	11	25420	16448	25362	16456	25367	16458
189	335	520	1342	2079	5367	8315	7	8	9	10	11	12	25040	32263	25003	32279	25007	32283
190	340	530	1361	2119	5442	8475	7	8	9	10	11	12	24672	31655	24654	31670	24663	31673
191	344	540	1379	2159	5517	8635	7	8	9	10	11	12	24385	31068	24332	31083	24328	31086
192	349	550	1398	2199	5591	8795	7	8	9	10	11	12	24036	30504	24001	30517	24006	30521
193	354	560	1416	2239	5665	8956	7	8	9	10	11	12	23696	29959	23696	29972	23692	29972
194	359	571	1436	2283	5745	9132	7	8	9	10	11	12	23366	29382	23366	29395	23362	29395
195	364	582	1456	2327	5825	9308	7	8	9	10	11	12	23045	28826	23045	28839	23041	28839
196	369	593	1476	2371	5905	9484	7	8	9	10	11	12	22733	28292	22733	28304	22729	28304
197	374	604	1496	2415	5984	9660	7	8	9	10	11	12	22429	27776	22429	27788	22429	27788
198	379	615	1516	2459	6063	9836	7	8	9	10	11	12	22133	27280	22133	27291	22137	27291
199	384	627	1537	2507	6149	10028	7	8	9	10	11	12	21845	26757	21831	26768	21827	26768
200	389	639	1559	2555	6234	10220	7	8	9	10	11	12	21564	26255	21523	26265	21529	26265
201	395	651	1580	2603	6319	10412	7	8	9	10	11	12	21236	25771	21236	25781	21240	25781
202	400	663	1601	2651	6404	10604	7	8	9	10	11	12	20971	25305	20958	25314	20958	25314
203	406	676	1624	2703	6495	10812	7	8	9	10	11	12	20661	24818	20661	24827	20664	24827
204	411	689	1647	2755	6587	11020	7	8	9	10	11	12	20410	24350	20373	24358	20376	24358
205	417	702	1670	2807	6678	11228	7	8	9	10	11	12	20116	23899	20092	23907	20098	23907
206	423	715	1692	2859	6769	11437	7	8	9	10	11	12	19831	23464	19831	23472	19828	23470
207	429	729	1717	2915	6867	11661	7	8	9	10	11	12	19553	23014	19542	23021	19545	23019
208	435	743	1741	2971	6966	11885	7	8	9	10	11	12	19284	22580	19273	22587	19267	22586
209	441	757	1766	3027	7064	12109	7	8	9	10	11	12	19021	22162	19000	22170	19000	22168
210	447	771	1791	3083	7163	12333	7	8	9	10	11	12	18766	21760	18735	21767	18737	21765
211	454	786	1817	3143	7269	12573	7	8	9	10	11	12	18477	21345	18466	21351	18464	21350
212	461	801	1844	3203	7376	12813	7	8	9	10	11	12	18196	20945	18196	20951	18196	20950
213	467	816	1871	3263	7483	13053	7	8	9	10	11	12	17962	20560	17933	20566	17936	20565
214	475	832	1900	3327	7599	13309	7	8	9	10	11	12	17660	20164	17660	20170	17662	20169
215	482	848	1929	3391	7715	13565	7	8	9	10	11	12	17403	19784	17394	19790	17396	19788
216	489	864	1958	3455	7832	13821	7	8	9	10	11	12	17154	19418	17137	19423	17137	19422
217	497	881	1990	3523	7958	14093	7	8	9	10	11	12	16878	19043	16861	19048	16865	19047

	IQ_Scale						FQ_Shift						FQ_Quant					
	8b		10b		12b		8b		10b		12b		8b		10b		12b	
Qindex	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC	DC	AC
218	505	898	2021	3591	8085	14365	7	8	9	10	11	12	16611	18682	16602	18688	16600	18686
219	513	915	2054	3659	8214	14637	8	8	10	10	12	12	32704	18335	32672	18340	32680	18339
220	522	933	2088	3731	8352	14925	8	8	10	10	12	12	32140	17982	32140	17986	32140	17985
221	530	951	2123	3803	8492	15213	8	8	10	10	12	12	31655	17641	31610	17646	31610	17645
222	539	969	2159	3876	8635	15502	8	8	10	10	12	12	31126	17313	31083	17313	31086	17316
223	549	988	2197	3952	8788	15806	8	8	10	10	12	12	30559	16980	30545	16980	30545	16983
224	559	1007	2236	4028	8945	16110	8	8	10	10	12	12	30012	16660	30012	16660	30009	16662
225	569	1026	2276	4104	9104	16414	8	9	10	11	12	13	29485	32704	29485	32704	29485	32708
226	579	1046	2319	4184	9275	16734	8	9	10	11	12	13	28976	32078	28938	32078	28941	32082
227	590	1066	2363	4264	9450	17054	8	9	10	11	12	13	28435	31476	28399	31476	28405	31480
228	602	1087	2410	4348	9639	17390	8	9	10	11	12	13	27869	30868	27846	30868	27848	30872
229	614	1108	2458	4432	9832	17726	8	9	10	11	12	13	27324	30283	27302	30283	27302	30287
230	626	1129	2508	4516	10031	18062	8	9	10	11	12	13	26800	29720	26757	29720	26760	29723
231	640	1151	2561	4604	10245	18414	8	9	10	11	12	13	26214	29152	26204	29152	26201	29155
232	654	1173	2616	4692	10465	18766	8	9	10	11	12	13	25653	28605	25653	28605	25650	28608
233	668	1196	2675	4784	10702	19134	8	9	10	11	12	13	25115	28055	25087	28055	25082	28058
234	684	1219	2737	4876	10946	19502	8	9	10	11	12	13	24528	27526	24519	27526	24523	27529
235	700	1243	2802	4972	11210	19886	8	9	10	11	12	13	23967	26994	23950	26994	23946	26997
236	717	1267	2871	5068	11482	20270	8	9	10	11	12	13	23399	26483	23374	26483	23378	26485
237	736	1292	2944	5168	11776	20670	8	9	10	11	12	13	22795	25970	22795	25970	22795	25973
238	755	1317	3020	5268	12081	21070	8	9	10	11	12	13	22221	25477	22221	25477	22219	25480
239	775	1343	3102	5372	12409	21486	8	9	10	11	12	13	21648	24984	21634	24984	21632	24987
240	796	1369	3188	5476	12750	21902	8	9	10	11	12	13	21076	24510	21050	24510	21053	24512
241	819	1396	3280	5584	13118	22334	8	9	10	11	12	13	20485	24036	20460	24036	20463	24038
242	843	1423	3375	5692	13501	22766	8	9	10	11	12	13	19901	23580	19884	23580	19882	23582
243	869	1451	3478	5804	13913	23214	8	9	10	11	12	13	19306	23125	19295	23125	19293	23127
244	896	1479	3586	5916	14343	23662	8	9	10	11	12	13	18724	22687	18714	22687	18715	22689
245	925	1508	3702	6032	14807	24126	8	9	10	11	12	13	18137	22250	18127	22250	18128	22252
246	955	1537	3823	6148	15290	24590	8	9	10	11	12	13	17567	21831	17553	21831	17556	21832
247	988	1567	3953	6268	15812	25070	8	9	10	11	12	13	16980	21413	16976	21413	16976	21414
248	1022	1597	4089	6388	16356	25551	8	9	10	11	12	13	16416	21010	16412	21010	16412	21011
249	1058	1628	4236	6512	16943	26047	9	9	11	11	13	13	31714	20610	31685	20610	31686	20611
250	1098	1660	4394	6640	17575	26559	9	9	11	11	13	13	30559	20213	30545	20213	30547	20214
251	1139	1692	4559	6768	18237	27071	9	9	11	11	13	13	29459	19831	29440	19831	29438	19831
252	1184	1725	4737	6900	18949	27599	9	9	11	11	13	13	28339	19451	28333	19451	28332	19452
253	1232	1759	4929	7036	19718	28143	9	9	11	11	13	13	27235	19075	27230	19075	27227	19076
254	1282	1793	5130	7172	20521	28687	9	9	11	11	13	13	26173	18714	26163	18714	26162	18714
255	1336	1828	5347	7312	21387	29247	9	9	11	11	13	13	25115	18355	25101	18355	25102	18356

VP9 SB, CU/PU and TU Sizes - Encoder Only

CU/PU/TU Partitioning Configurations

SB size	CU size	min/max TU range
64x64	64x64	32x32..4x4
	32x32	32x32..4x4
	16x16	16x16..4x4
	8x8	8x8..4x4

PU Options for a Given CU

Current CU size	Possible CU sizes	Allowed CU/PU partition types.
64x64	64x64	2Nx2N, 2NxN, Nx2N
	32x32	2Nx2N, 2NxN, Nx2N
	16x16	2Nx2N, 2NxN, Nx2N
	8x8	2Nx2N, 2NxN, Nx2N, NxN

Definition of the VP9 CU Record Structure - Encoder Only

he following table defines the CU record structure as indirect data to the PAK Object Command. Entries are DW based (4 bytes) and cache aligned. This memory surface is pointed to by the HCP Indirect CU Object Base Address in the HCP_IND_OBJ_BASE_ADDR_STATE Command.

VP9 CU Record Structure Definition (Packed CU)

VP9 Compact CU Packet:

R0.7	31:21	**Reserved MBZ**
	20	**Modified Flag (should not be used by HW)**
		This bit is used by VME to alert kernel of modifications. SRM and FBR can modify the CU Bit Cost and CU Errors.
		0 = No modification to the CU Bit Cost and CU Error.
		1 = Modifications to the CU Bit Cost or CU Error or both.
		Note: HPM will set this to zero.
	19	**Reserved**
	18	**interpred_comp1**
		Interpred comp mode for Part1

		0: Single
		1: Compound(ref_refframe1>intra)
	17	**Reserved**
	16	**interpred_comp0**
		Interpred comp mode for Part0
		0: Single
		1: Compound(ref_refframe1>intra)
	15	**cu_pred_mode1**
		Pred mode for part1 in raster scan order
		cu_pred_mode=intra means ref_refframe0=intra
		0: Intra
		1: Inter
	14	**cu_pred_mode0**
		Pred mode for part0 in raster scan order
		cu_pred_mode=intra means ref_refframe0=intra
		0: Intra
		1: Inter
	13:12	**CU_part_mode**
		0: 2Nx2N,
		1: 2NxN,
		2: Nx2N,
		3: NxN (8x8 only)
	11:8	**intra_chroma_mode[0]**
		0:DC_PRED
		1:V_PRED
		2:H_PRED
		3:TM_PRED
		4:D45_PRED
		5:D135_PRED 6:D117_PRED 7:D153_PRED 8:D207_PRED
		9:D63_PRED
		10-15 Reserved
	7:6	**CU Size**
		0 = 8x8
		1 = 16x16
		2 = 32x32
		3 = 64x64
	5:4	**Reserved**
	3:0	**intra_mode[0]**

		Luma Intra mode for part0
		0:DC_PRED
		1:V_PRED
		2:H_PRED
		3:TM_PRED
		4:D45_PRED
		5:D135_PRED 6:D117_PRED 7:D153_PRED 8:D207_PRED
		9:D63_PRED
R0.6	31:28	**intra_chroma_mode[1]**
		Applicable for part1 of shapes > 8x8
		0:DC_PRED
		1:V_PRED
		2:H_PRED
		3:TM_PRED
		4:D45_PRED
		5:D135_PRED 6:D117_PRED 7:D153_PRED 8:D207_PRED
		9:D63_PRED
		10-15 Reserved
	27:22	**Reserved MBZ**
	21:18	**Reserved MBZ**
		Format: U4
	17:16	**Reserved MBZ**
	15:12	**intra_mode[3][3:0]**
		Luma pred mode for part3
		0:DC_PRED
		1:V_PRED
		2:H_PRED
		3:TM_PRED
		4:D45_PRED
		5:D135_PRED
		6:D117_PRED
		7:D153_PRED
		8:D207_PRED
		9:D63_PRED
	11:10	**Reserved**
	9:6	**intra_mode[2][3:0]**
		Luma pred mode for part2

		0:DC_PRED
		1:V_PRED
		2:H_PRED
		3:TM_PRED
		4:D45_PRED
		5:D135_PRED
		6:D117_PRED
		7:D153_PRED
		8:D207_PRED
		9:D63_PRED
	5:4	**Reserved**
	3:0	**intra_mode[1][3:0]**
		Luma pred mode for part1
		0:DC_PRED
		1:V_PRED
		2:H_PRED
		3:TM_PRED
		4:D45_PRED
		5:D135_PRED
		6:D117_PRED
		7:D153_PRED
		8:D207_PRED
		9:D63_PRED
R0.5	31:30	**Reserved**
	29:28	**MotionComp_flttype[1]**
		0: EIGHTTAP
		1: EIGHTTAP_SMOOTH
		2: EIGHTTAP_SHARP
		3: BILINEAR
		HW will use this filtertype if SWITCHABLE=1 in pic state;
		Used for part1 of blocks > 8x8
	27:26	**Reserved**
	25:24	**MotionComp_flttype[0]**
		0: EIGHTTAP
		1: EIGHTTAP_SMOOTH
		2: EIGHTTAP_SHARP
		3: BILINEAR
		HW will use this filtertype if SWITCHABLE=1 in pic state;

		Used for part0
	23	**Reserved**
	22:20	**SegmentIdx[1]**
		Segment 0 to 7
		SegmentID for part1 of blocks > 8x8
	19	**Reserved**
	18:16	**SegmentIdx[0]**
		Segment 0 to 7
		SegmentID for part0 of blocks > 8x8
	15	**segmentPredFlag1**
		0: Disable
		1: Enable
		Segment Prediction disable for Part1
	14	**segmentPredFlag0**
		0: Disable
		1: Enable
		Segment Prediction disable for Part0
	13:4	**Reserved**
	3:2	**TU_SIZE[1]**
		0 = 4x4
		1 = 8x8
		2 = 16x16
		3 = 32x32
		tu_size[2][1:0], tu_size[3][1:0] must be 0 (4x4)
	1:0	**TU_SIZE[0]**
		0 = 4x4
		1 = 8x8
		2 = 16x16
		3 = 32x32
		tu_size[2][1:0], tu_size[3][1:0] must be 0 (4x4)
R0.4	31:29	**QuantRound1**
		Quantization Round value for Part1 (8x8 and below shapes will have the same round value) In VDEnc mode, this parameter is set to the QuantRound0.
		0:+1/16
		1:+2/16
		2:+3/16
		3:+4/16

		4:+5/16
		5:+6/16(default)
		6:+7/16
		7:+8/16
	28:26	**QuantRound0**
		Quantization Round value for Part0 (8x8 and below shapes will have the same round value) This parameter is equivalent to the RoundingSelect in the HEVC CU packet.
		0:+1/16
		1:+2/16
		2:+3/16
		3:+4/16
		4:+5/16
		5:+6/16(default)
		6:+7/16
		7:+8/16
	25:14	**Reserved**
	13:12	**ref_refframe1[1]**
		frame1(backward)reference frame id for part0
		HW uses if SegmentReferenceEnabled=0 in segment ID command
		0:intra
		1:last
		2:golden
		3:altref
		Format = U2
	11:10	**Reserved**
	9:8	**ref_refframe1[0]**
		frame1(forward)reference frame id for part0
		HW uses if SegmentReferenceEnabled=0 in segment ID command
		0:intra
		1:last
		2:golden
		3:altref
		Format = U2
	7:6	**Reserved**
	5:4	**ref_refframe0[1]**
		frame0(forward)reference frame id for part1
		HW uses if SegmentReferenceEnabled=0 in segment ID command

		0:intra
		1:last
		2:golden
		3:altref
		Format = U2
	3:2	**Reserved**
	1:0	**ref_refframe0[0]**
		frame0(forward)reference frame id for part0
		HW uses if SegmentReferenceEnabled=0 in segment ID command
		0:intra
		1:last
		2:golden
		3:altref
		Format = U2
R0.3	31:16	**mvy_refframe1[1]**
		Frame1(Backward) MVy for part1
		Format = S13.2 (2's comp)
	15:0	**mvx_refframe1[1]**
		Frame1(Backward) MVx for part1
		Format = S13.2 (2's comp)
R0.2	31:16	**mvy_refframe1[0]**
		Frame1(Backward) MVy for part0
		Format = S13.2 (2's comp)
	15:0	**mvx_refframe1[0]**
		Frame1(Backward) MVx for part0
		Format = S13.2 (2's comp)
R0.1	31:16	**mvy_refframe0[1]**
		Frame0(Forward) MVy for part1
		Format = S13.2 (2's comp)
	15:0	**mvx_refframe0[1]**
		Frame0(Forward) MVx for part1
		Format = S13.2 (2's comp)
R0.0	31:16	**mvy_refframe0[0]**
		Frame0(Forward) MVy for part0
		Format = S13.2 (2's comp)
	15:0	**mvx_refframe0[0]**
		Frame0(Forward) MVx for part0
		Format = S13.2 (2's comp)

Quant Scale and Filter Level Table

	Decode mode	Encode BRC first pass, or no BRC	Encode HW multi-pass BRC - Subsequent Passes
Determination of base qindex	N/A	Directly from PIC_STATE, base_qindex	final_base_qindex = clip(0,255, base_qindex (from PIC_STATE) + accumulated delta from previous passes)
Determination of inverse quant scale of each block	Directly from SEGMENT_STATE	final_qindex = clip(0,255, clip(0, 255, base_qindex + // from PIC_STATE segment_q_delta) + // from SEGMENT_STATE chroma or AC delta // from PIC_STATE) iq_lookup_table[final_qindex];	final_qindex = clip(0,255, clip(0, 255, final_base_qindex + // from above segment_q_delta) + // from SEGMENT_STATE chroma or AC delta // from PIC_STATE) iq_lookup_table[final_qindex];
Determination of forward quant scale and shift of each block	N/A	fq_lookup_table[final_qindex];	fq_lookup_table[final_qindex];
Determination of base filter level	N/A	Directly from PIC_STATE (filter_level) final_base_filter_level = filter_level (from PIC_STATE)	final_base_filter_level = clip(0, 63, filter_level (from PIC_STATE) + accumulated delta from previous passes)
Determination of final filter level of each block	Directly from SEGMENT_STATE	mask = final_base_filter_level > 31 ? -2 : -1; final_filter_level = clip(0,63, clip(0,63, base_filter_level + // from PIC_STATE segment_lf_delta) + // from SEGMENT_STATE) (ref_delta & mask) + // from PIC_STATE (mode_delta & mask) // from PIC_STATE);	mask = final_base_filter_level > 31 ? -2 : -1; final_filter_level = clip(0,63, clip(0,63, final_base_filter_level + // from above segment_lf_delta) + // from SEGMENT_STATE) (ref_delta & mask) + // from PIC_STATE (mode_delta & mask) // from PIC_STATE);

When SW/HuC writes the uncompressed header for an encoded frame, it must downshift the mode_delta and ref_delta by 1, if the final_base_filter_level > 31.

VP9 Allowed SB Size Encoder Only

The following table details the SB size allowed and the number of records per SB for the encoder.

Allowed SB Size - Encoder Only

VP9 SB Size Allowed	Number of Records per SB
64x64	64

Note: HW will support partial SBs within a frame bourndary to a minimun CU8x8 granularity

HCP PAK Data Structure

The following documents HCP PAK Data Structure.

Tile Size and CU Stream-out Records

TileSize Record for both HEVC and VP9 Codecs

Fields	Bits	Notes
Address	DW0-1[63:0]	Bitstream start address; baseAddr + Tile offset. Used for stitching purpose in scalability mode. MBZ in single pipe mode.
Length	DW2[31:0]	Bitstream length per tile; Includes header in first tile and tail in last tile. Used for stitching purpose in scalability mode. MBZ in single pipe mode.
Tile Size	DW3[31:0]	Tile Size(no header) used by HW for back annotation. Also HuC uses for BRC purpose in scalability mode for both hevc/vp9. MBZ in HEVC single pipe mode.
AddressOffset	DW4[31:0]	Cacheline Address to be Modified. Used by HW for back annotation. MBZ in HEVC mode.
Offset	DW5[5:0]	Byte offset to be Modified. Used by HW for back annotation. MBZ in HEVC mode.
Reserved	DW5[31:6]	MBZ
HCP_BITSTREAM_SE_BITCOUNT_FRAME	DW6[31:0]	Bitstream size for Syntax element per Tile (see the MMIO register for details). valid only for HEVC in scalability mode and MBZ for VP9
HCP_CABAC_BIN_COUNT_FRAME	DW7[31:0]	Bitstream size for Bin count per Tile (see the MMIO register for details). valid only in scalability mode
Reserved	DW8[31:0]	MBZ
HCP_IMAGE_STATUS_CONTROL	DW9[31:0]	Image Status Mask Control(see the MMIO register for details). Only valid fields are Total-NumPass[11:8](hevc/vp9) and LCUbitCountViolate[0](hevc only).The rest of the fields are MBZ. valid only in scalability mode
HCP_QP_STATUS_COUNT	DW10[31:0]	QP Status count (see the MMIO register for details). valid only for HEVC in scalability mode and MBZ for VP9
HCP_SLICE_COUNT	DW11[31:0]	Slice count (see the MMIO register for details). valid only for HEVC in scalability mode and MBZ for VP9
Reserved	DW12-15	MBZ

HEVC: Streamout0 cacheline is composed of 4 quarter cachelines, each containing information on CU skip flag, coding block flag for the TUs in a PU, residual/coefficient bit count for a PU, total bit count for CU, SB exceed limit flag. A typical streamout0 cacheline, therefore, has information on statistics for 4 PUs and Super Block exceed limit flag.

Pak pipeline streamout enable bit, set by HCP_PIPE_MODE_SELECT command, enables or disables the streamout.

Programming Note				
Context:			CU level statistics	
Level	**Field**	**Width**	**Cacheline**	**Comment**
PU	PU Skip Flag	1	qcacheline[0]	Packed in Quarter Cacheline in PU format
SB	SB exceed limit	1	qcacheline[1]	Packed in Quarter Cacheline in PU format (valid on last PU of SB)
	Reserved	14	qcacheline[15:2	Reserved
PU	TU CBF Y/U/V	48	qcacheline[63:16]	Packed in Quarter Cacheline in PU format
PU	PU Coefficient Bit Count (Only residual)	18	qcacheline[81:64]	Packed in Quarter Cacheline in PU format
PU	PU Bit Count (all PU Syntax)	18	qcacheline[113:96]	Packed in Quarter Cacheline in PU format
	Reserved	14	qcacheline[127:114]	Reserved

Programming Note				
Context:			.	

HEVC Streamout 1: Per Tile Quarter Cacheline

Level	Field	Width	Cacheline	Comment
Tile	Tile Bit Count (header + data + tail)	32	cacheline[31:0]	
	Reserved(MBZ)	32	cacheline[63:32]	
	TilePositionX[15:0]	16	cacheline[79:64]	
	TilePositionY[15:0]	16	cacheline[95:80]	
	Reserved(MBZ)	32	cacheline[127:96]	

VP9: CU statistics record (individual PUs per record down to 8x8 only)

Fields	Bits	
Skip	3:0	Indicates Skip flag Group 4 4x4s -> 4 bits
InterMode	11:4	InterMode: 0 NEARESTMV, 1 NEARMV, 2 ZEROMV, 3NEWMV Group 4 4x4s total 8 bits
Reserved	15:12	
NZ coeff count	28:16	Number of non-zero coeffs; sum of YUV, 13bits

Fields	Bits	
Reserved	31:29	
NumBitsforCoeffs	47:32	Number of Bits for coefficients per block, 16bits
NumBitsforBlock	63:48	Number of Bits in block

Definition of the CU Record Structure- Encoder Only

The following table defines the CU record structure as indirect data to the PAK Object Command. Entries are DW based (4 bytes) and cache aligned. This memory surface is pointed to by the HCP Indirect CU Object Base Address in the HCP_IND_OBJ_BASE_ADDR_STATE Command.

CU Record Structure Definition

Intel restriction max 16 TU per CU, max 256 TUs in a CU.

Definition of the CU Record Structure for VME Interface - Encoder Only

DWord	Bitfield	Definition
R0.7	31	**CU_qp sign** Indicates sign bit for QP. Must be zero for 8bit mode
	30:24	**CU_qp** Note: HPM will set this to zero. This is a pass through for FBR and SRM. Kernel needs to populate this field before calling PAK. Magnitue of CU level QP. Valid range: 0 to 51 for 8bit mode -12 to 51 for 10bit mode -24 to 51 for 12bit mode QP can change at CU level Restriction: diff_cu_qp_delta_depth must be equal to either 0 or (lcu_size - min_cu_size)
	23	**zero_out_coefficients** 0: Do not force coefficients to zero for entire CU 1: Force coefficients to zero for entire CU *This bit must be zero in VDenc mode*
	23	**zero_out_coefficients_Y** 0: Do not force coefficients to zero in Luma block 1: Force coefficients to zero in Luma block
	22	IPCM_enable If IPCM is enabled, then entire CU is IPCM predicted. Both PU and TU sizes should be same as CU size. Cu_pred_mode is ignored when IPCM is enbled. 1- enable IPCM *0-disable IPCM* Note: Supports 8b only

<table>
<tr><th>DWord</th><th>Bitfield</th><th>Definition</th></tr>
<tr><td></td><td></td><td>Note: Supports 8, 10 and 12bits.
Note: HW ignores this bit for RhoDomain calculations so the statics will be slightly inaccurate.</td></tr>
<tr><td></td><td>21</td><td>Last CU of LCU Flag
Set to 1, if the current CU is the last one inside the current LCU (for VDENC only).</td></tr>
<tr><td></td><td>20</td><td>Modified Flag (should not be used by HW)</td></tr>
<tr><td></td><td>19:18</td><td>InterPred_IDC_ MV1</td></tr>
<tr><td></td><td>17:16</td><td>InterPred_IDC_ MV0</td></tr>
<tr><td></td><td>15</td><td>cu_pred_mode</td></tr>
<tr><td></td><td>14:12</td><td>CU_part_mode
Note: NxN CU_part_mode is used by RPM only in VME. It is used to generate predicted pixels using different prediction modes per 4x4 sub-block. i.e each 4x4 sub-block can have its own prediction mode.
Note: 2NxN and N2XN intra is only valid for VP9. VP9 supports 32x16, 16x32, 16x8 and 8x16 Intra parititions.
Luma Intra Mode indicates the intra prediction mode for 4x4_0. The additional prediction modes are overloaded on R0.6 [23:0] in this case.</td></tr>
<tr><td></td><td>11</td><td>CU_transquant_bypass_flag</td></tr>
<tr><td></td><td>10:8</td><td>Chroma Intra Mode
0: DM (use Luma mode, from block 0 if NxN)
1 : Reserved (supposedly to be defined for LM mode)
2: Planar
3: Vertical
4: Horizontal
5: DC
Note: Also indicates Chroma Intra Mode for block0 for 4:2:2 and 4:4:4.</td></tr>
<tr><td></td><td>7:6</td><td>CU Size</td></tr>
<tr><td></td><td>5:0</td><td>Luma Intra Mode</td></tr>
<tr><td>R0.6</td><td>31:30</td><td>SCC CU Coding Mode (for VDENC only)
<table><tr><th>Bit 31:30</th><th>Definition</th></tr><tr><td>00</td><td>Not IBC and Not Palette</td></tr><tr><td>01</td><td>IBC</td></tr><tr><td>10</td><td>Palette</td></tr><tr><td>11</td><td>Illegal</td></tr></table></td></tr>
<tr><td></td><td>29</td><td>Palette Transpose Flag (for VDENC only)
Set to 1, if the palette indices of the current CU are to be transposed.</td></tr>
</table>

DWord	Bitfield	Definition
	28	**Palette Transpose Flag (for VDENC only)** Set to 1 when there is at least one pixel is coded with escape code.
	27:24	**TU Count M1** Note: Intel restriction max 16 TU per CU (however spec allows up to 256 TUs).
	23:22	**Reserved MBZ**
	21:18	**Rounding Select** This select is used to pick up the threshold table in the PAK for RhoDomain. It is also used for quantization.This is generated based on CU type, rounding threshold and rounding offset in the VDEnc. Note: This Rounding Select used after Quantization only when RhoDomain is ON in VDenc Mode. Format: U4
	17:12	**Luma Intra Mode 4x4_3** Final explicit Luma Intra Mode 4x4_1. Valid values 0..34 Note: CU_part_mode==NxN
	11:6	**Luma Intra Mode 4x4_2** Final explicit Luma Intra Mode 4x4_1. Valid values 0..34 Note: CU_part_mode==NxN
	5:0	**Luma Intra Mode 4x4_1**
R0.5	31:0	**TU Size** Note: HPM will set this to zero. This is a pass through for FBR and SRM. Kernel needs to populate this field before calling PAK.
R0.4	31:16	**TU_YUV_Transform_Skip** 0: TU transform skip flag is not set (normal transform) 1: TU transform skip flag is set Note: HPM will set this to zero. This is a pass through for FBR and SRM. Kernel needs to populate this field before calling PAK. UV flags are overloaded on Y
	15:12	**L1_MV1 RefID/0&Chroma Intra Mode1[14:12]** Format = U4/U3 overload **Chroma Intra Mode** if **cu_pred_mode=Intra of** block1 for 4:2:2 and 4:4:4
	11:8	**L1_MV0 RefID[11:8]/0&Chroma Intra Mode2[10:8]** Format = U4/U3 overload **Chroma Intra Mode** if **cu_pred_mode=Intra of** block2 for 4:2:2 and 4:4:4
	7:4	**L0_MV1 RefID[7:4]/0&Chroma Intra Mode3[6:4]** Format = U4/U3 overload **Chroma Intra Mode** if **cu_pred_mode=Intra of** block3 for 4:2:2 and 4:4:4
	3:0	**L0_MV0 RefID**
R0.3	31:16	**L1_MV1.Y**
	15:0	**L1_MV1.X**
R0.2	31:16	**L1_MV0.Y**

DWord	Bitfield	Definition
	15:0	**L1_MV0.X**
R0.1	31:16	**L0_MV1.Y**
	15:0	**L0_MV1.X**
R0.0	31:16	**L0_MV0.Y** **Programming Note** Intra MV Y for IBC
	15:0	**L0_MV0.X** **Programming Note** Intra MV X for IBC

R0.0 Exists if (CU_Pred_Mode=0, IntraCU)	31:27	**Reserved MBZ**
	26:24	**Chroma Intra Mode second best** DM This is the chroma Intra mode that corresponds to the "Luma Intra Mode second best" that is derived assuming the top or left neighbor is not available. 0: DM (use Luma mode, from block 0 if NxN) 1 : Reserved (supposedly to be defined for LM mode) 2: Planar 3: Vertical *4: Horizontal* 5: DC
	23:18	**Luma Intra Mode second best 4x4_3** *Valid only when CU_part_mode==NxN.* Valid values 0..34
	17:12	**Luma Intra Mode second best 4x4_2** Valid only when CU_part_mode==NxN.
	11:6	**Luma Intra Mode second best 4x4_1** *Valid only when CU_part_mode==NxN.* Valid values 0..34
	5:0	**Luma Intra Mode second best** When slices are dynamically terminated in the PAK, this Luma Intra Mode is used for the CUs when only the left or only the top neighbor is available. *When both top and left neighbors are not available, PAK defaults to DC mode for those CUs for both Luma and Chroma Intra modes.* Valid values 0..34

Programming the CU Palette Map:

Code Name (Command Type)	Code[1:0] = Data[76:75]	End = Data[74]	Valid[1:0]= Data[73:72]	Payload[...]
REUSED FLAG	00			Data[63:0]
REUSED FLAG	00			Data[63:0]
NEW COLORS	01		Max: 2 Colors per clock. Valid[1:0]=11 or 01 Valid[1:0]=00 means No NEW COLORS at all	Data[71:36], Data[35:0] 36 bits/Pixel with 12 bits/component.
NEW COLORS ...	01	End=1 on the last New Colors		
INDEX	10			Data[29:24], Data[21:16], Data[13:8], Data[5:0] 6 bits/INDEX (Max Palette Table size is 64). 4 INDEXes per clock in Raster order within a CU: 4x1 after 4x1 ... till CU right edge. Escape_Map[3:0]=Data[35:32]
INDEX ...	10	End=1 on the last Index of CU		
ESCAPE	11		Max: 2 Escapes per clock. Valid[1:0]=11 or 01 Valid[1:0]=00 means No ESCAPE at all.	Data[71:36], Data[35:0] 36 bits/Pixel with 12 bits/component Max number of ESCAPE = 25% of CU Pixels.
ESCAPE ...	11	End=1 on the last Escape of CU		

HEVC LCU, CU, TU, and PU Sizes - Encoder Only

HEVC LCU/CU Partitioning Configurations

LCU size	min CU size	CU Depth	Hierarchical Depth=CU Depth+1
64x64	64x64	0	1
	32x32	1	2
	16x16	2	3
	8x8	3	4
32x32	32x32	0	1
	16x16	1	2
	8x8	2	3
16x16	16x16	0	1
	8x8	1	2
8x8	8x8	X	Not allowed in spec

HEVC PU Options for a Given CU

Current CU size (leaf node)	min CU sizes (Pic State)	Allowed PU partition types.
64x64 (2Nx2N) Must be a LCU	64x64	Skip : 2Nx2N Intra : 2Nx2N, NxN Inter : 2Nx2N, 2NxN, Nx2N, NxN
	32x32	Skip : 2Nx2N Intra : 2Nx2N (no NxN defined in the spec.) Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
	16x16	Skip : 2Nx2N Intra : 2Nx2N (no NxN

Current CU size (leaf node)	min CU sizes (Pic State)	Allowed PU partition types.
		defined in the spec.) Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
	8x8	Skip : 2Nx2N Intra : 2Nx2N (no NxN defined in the spec.) Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
32x32 (2Nx2N) Can or is not a LCU	32x32	Skip : 2Nx2N Intra : 2Nx2N, NxN Inter : 2Nx2N, 2NxN, Nx2N, NxN
	16x16	Skip : 2Nx2N Intra : 2Nx2N Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
	8x8	Skip : 2Nx2N Intra : 2Nx2N Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD,

Current CU size (leaf node)	min CU sizes (Pic State)	Allowed PU partition types.
		nLx2N, nRx2N
16x16 (2Nx2N) Can or is not a LCU	16x16	Skip : 2Nx2N Intra : 2Nx2N, NxN Inter : 2Nx2N, 2NxN, Nx2N, NxN
	8x8	Skip : 2Nx2N Intra : 2Nx2N Inter : 2Nx2N, 2NxN, Nx2N, 2NxnU, 2NxnD, nLx2N, nRx2N
8x8 (2Nx2N) Cannot be a LCU	8x8	Skip : 2Nx2N Intra : 2Nx2N and NxN Inter : 2Nx2N, 2NxN, Nx2N (both NxN and AMP are not allowed for 8x8 inter CU)

Note: In an 8x8 Inter CU NxN isn't allowed if the SPS parameter disable_inter_4x4 is 1. In Main profile currently this flag is always 1.

U.D, L and R (Up, Down, Left and Right)

n = 1/4 or 3/4 .

HEVC TU Partitioning for a Given CU

CU size	TU size	TU Depth	Max Depth=TU Depth+1	PAK supported TU sizes and corresponding number of TUs in CU
64x64	64x64	0	1	no 64x64 transform, so automatically breakdown into 4 32x32 TUs.
	32x32	1	2	number of TUs in CU = 4
	16x16	2	3	number of TUs in CU = 16
	8x8	3	4	this configuration is currently not supported.
	4x4	4	5	this configuration is currently not supported.
32x32	32x32	0	1	number of TUs in CU = 1
	16x16	1	2	number of TUs in CU = 4
	8x8	2	3	number of TUs in CU = 16
	4x4	3	4	this configuration is currently not supported.
16x16	16x16	0	1	number of TUs in CU = 1
	8x8	1	2	number of TUs in CU = 4
	4x4	2	3	number of TUs in CU = 16
8x8	8x8	0	1	number of TUs in CU = 1
	4x4	1	2	number of TUs in CU = 4

The actual level of partitioning is governed by

- MaxTUSize and MinTUSize in Pic State.
- max_transform_hierarchy_depth_inter <= 2 (intel restriction) DW4 bit 3:2 Pic State
- max_transform_hierarchy_depth_intra <= 2 (intel restriction) DW4 bit 1:0 Pic State

Allowed LCU Size - Encoder Only

The following table details the LCU size allowed and the number of records per LCU for the encoder.

LCU Size Allowed	Fixed Number of Records per LCU
64x64	64
32x32	16
16x16	4

Note: 0.5 CL per CU record in VME mode and 1 CL per CU record in extENC mode.

HEVC/VP9 PAK Frame Statistics

PAK outputs frame level statistics for RhoDomain, SSE, slice size conformance features and LCU statistics. The RhoDomain and Slice Size conformance parameters are exclusive only to HEVC. The SSE and LCU statistics are for both HEVC and VP9.

HEVC Frame Statistics

SliceSizeConformance

DWord	Bit	Description
0	31:17	**Reserved: MBZ**
	16	**Slice Overflow Occured** Format: Enable This field indicates that one or more slices in the current frame exceeded the Target size. When the actual slice size exceeds "Target slice size in Bytes", HW sets "Slice Overflow Occurred" bit in the PAK Frame Statistics.
	15:8	**MaxFrameQP** This parameter indicates the maximum CU QP in the frame. Format: U8 Valid Range is 0-63 for 10bit and 0-51 for 8bit hevc
	7:0	**MinFrameQP** This parameter indicates the minimum CU QP in the frame. Format: U8 Valid Range is 0-63 for 10bit and 0-51 for 8bit hevc
1	31:0	**Max Slice Size in Bytes** Format: U32 This parameter indicates the largest Slice in Bytes in the current frame.
2..3	31:0	**Reserved: MBZ**

VP9 Frame statistics

SSE Statistics

..33	63:48	**Reserved: MBZ**
	47:0	**Frame Luma SSE** MSB word of the Sum square Error statistics for the luma pixels in the current frame. The internal 4x4 subblock SSE is 24-bits. This is accumulated across the frame for all 4x4s and clamped to 48-bits. Format: U48
34..35	63:48	**Reserved: MBZ**
	47:0	**Frame Chroma Cb SSE** Sum square Error statistics for Cb pixels in the current frame. The internal 4x4 subblock SSE is 24-bits. This is accumulated across the frame for all 4x4s and clamped to 48-bits. Format: U48
36..37	63:48	**Reserved: MBZ**
	47:0	**Frame Chroma Cr SSE** Sum square Error statistics for Cr pixels in the current frame. The internal 4x4 subblock SSE is 24-bits. This is accumulated across the frame for all 4x4s and clamped to 48-bits. Format: U48
38	31:16	**Class0 Zone1 4X4 SUBBLKS SSE Count** This parameter indicates the count of the nu4x4 subblkser of macro-blocks in the current frame whose Sum square Error (SSE) met the Class0 Zone1 SSE threshold requirements. The output count is a multiple of 16. The value is internally » 4. Format: U16
	15:0	**Class0 Zone0 4X4 SUBBLKS SSE Count** This parameter indicates the count of the nu4x4 subblkser of macro-blocks in the current frame whose Sum square Error (SSE) met the Class0 Zone0 SSE threshold requirements. The output count is a multiple of 16. The value is internally » 4. Format: U16
39	31:16	**Max Class0 4X4 SUBBLKS SSE** The maximum macro-block sum square error for the Y+U+V pixels for the macro-blocks that

		were in Class 0. This is clamped to 16-bits. The internal 4x4 subblock SSE is 24-bits. It is » 4 before the threshold check for zone classification and Max 4x4 SSE clamping. Format: U16
	15:0	**Class0 Zone2 4X4 SUBBLKS SSE Count** This parameter indicates the count of the nu4x4 subblkser of macro-blocks in the current frame whose Sum square Error (SSE) met the Class0 Zone2 SSE threshold requirements. The output count is a multiple of 16. The value is internally » 4. Format: U16
40-55	31:0 (Each)	**SSE Statistics for Class1-8.** **Class1-8 Zone0 4X4 SUBBLKS SSE Count** **Class1-8 Zone0 4X4 SUBBLKS SSE Count** **Class1-8 Zone0 4X4 SUBBLKS SSE Count** **Max Class1-8 4X4 SUBBLKS SSE** SSE statistics for Class 1-8, see DW SSE Class 0 statistics for format.
56-63	31:0	**Reserved: MBZ**

HEVC Error Concealment

The HCP implements an error concealment policy, which is always enabled and cannot be disabled. The objective is that the HCP will always complete a frame/field workload by either decoding the bit stream normally until it finishes the workload or by concealing blocks until the slice or workload is completed. It should never be allowed to hang.

Error concealment, implemented by the HCP hardware, is configured for each slice in the HCP_BSD_OBJECT command. The following information in the HCP_BSD_OBJECT command is utilized for error concealment.

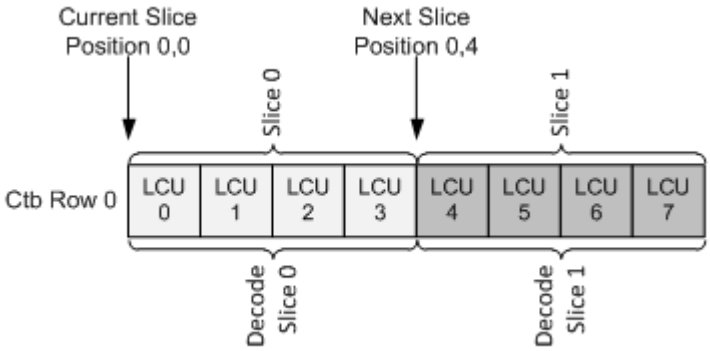
- **SliceStartCtbY, SliceStartCtbX:** The current slice position specified in Ctb coordinates.
- **NextSliceStartCtbY, NextSliceStartCtbX:** The next slice position specified in Ctb coordinates. If the current slice is the last slice in the picture, the next slice values are set to (0,0).
- **LastSliceofPic:** Indicates that the current slice is the last slice in the picture.
- **slice_type:** Indicates the picture type: I, P or B.

The host software will remove all extra slices in the picture. The HCP will not be given a workload that includes extra slices beyond the picture. The last slice in the picture will always be marked by the host software.

The host software will remove any overlapping slices in the picture. The HCP will not be given a workload that includes overlapping slices in the picture.

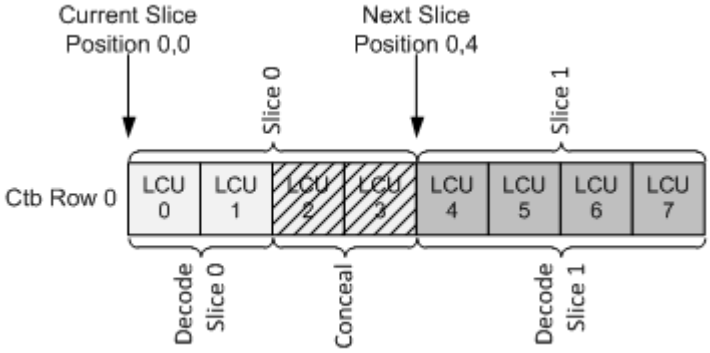
A HCP_BSD_OBJECT command will include the current slice position and the next slice position. For non-errored streams, it is guaranteed that the slice bit stream will be decoded by the HCP starting from the current slice position through to the Ctb (inclusive) adjacent to the Ctb indicated by the next slice position. HEVC Slice Decode for Non-errored Stream Cases illustrates the example of a non-errored stream decode starting with XXX.

HEVC Slice Decode for Non-errored Stream Cases



For error stream cases where the next slice position does not align itself with the last successfully decoded Ctb in the current slice, the HCP will conceal Ctbs from the last decoded Ctb in the current slice through to the last Ctb prior to the Ctb indicated by the next slice position. If the error occurs such that the current decoded Ctb cannot be decoded, the HCP will ensure that the current Ctb is written out by any means before writing out concealed Ctbs for the remaining Ctbs in the current slice. In the case of the last slice in a picture, the HCP will conceal Ctbs from the last decoded Ctb in the current slice through to the last Ctb position in the picture indicated by the resolution of the picture in the HCP_PICT_STATE command. HEVC Slice Decode for Missing Blocks in a Slice illustrates the case described.

HEVC Slice Decode for Missing Blocks in a Slice



Since the host software removes overlapping slices, the next slice position will never be equal to or less than the current slice position.

A concealed Ctb for an I-slice is constructed by the HCP specifying the Intra_Planar prediction mode for the Ctb.

A concealed Ctb for a P-slice is constructed by the HCP specifying the skip_flag.

A concealed Ctb for a B-slice is constructed by the HCP specifying the skip_flag.

HEVC Register Definitions

The Message Channel Interface is a read-only bus used to access the HCP and HUC status registers. All registers are 32 bits where reserved bits return a value of zero and subtractive-decode is used to return 0x0000 for all register holes. The Unit ID is 28h. For HCP, the address range is 0x0001E900h to 0001E9FFh and for HUC, the address range is 0x0000D000h to 0000D7FFh.

HCP Encoder Register Read/Write

Register Name	Address	Tile-Based Engine OFF (Reads at FRAME boundary)	Tile-Based Engine ON (Reads at TILE ROW boundary)
HCP_BITSTREAM_BYTECOUNT_FRAME	8'hA0	Frame Byte Count per frame	Accumulated Frame Byte Count until current tile
HCP_BITSTREAM_BYTECOUNT_FRAME_NO_HDR	8'hA4	Frame Byte Count without header per frame	Accumulated Frame Byte Count without header until current tile
HCP_BITSTREAM_SE_BITCOUNT_FRAME	8'hA8	Syntax element bit count per frame	Accumulated Syntax element bit count until current tile
HCP_CABAC_BINCOUNT_FRAME	8'hAC	Bin count per frame	Accumulated Bin count until current tile
HCP_CABAC_INSERTION_COUNT	8'hB0	Cabac Zero Word insertion byte count per frame	Cabac Zero Word insertion byte count per frame. (only available at the last tile of the frame)
HCP_MIN_FRAME_PADDING_COUNT	8'hB4	Min Frame Padding byte count per frame	>Min Frame Padding byte count per frame. (only available at the last tile of the frame)
HCP_IMAGE_STATUS_MASK	8'hB8	Frame level Mask bits (BRC Min, BRC Max, LCU Max mask bits)	>Frame level Mask bits (BRC Min, BRC Max, LCU Max mask bits). Updates every HCP_PIC_STATE i.e. per tile
HCP_IMAGE_STATUS_CONTROL	8'hBC	Frame level Status Control bits. Cumulative Delta Qp, Frame Underflow/Overflow, LCU size exceed	>Some bits are updated tile/frame level. Driver updates Cumulative Delta Qp at the start of every tile row.

Register Name	Address	Tile-Based Engine OFF (Reads at FRAME boundary)	Tile-Based Engine ON (Reads at TILE ROW boundary)
		flag are reflected per frame level.	LCU size exceed flag updates every tile. Frame Underflow/Overflow updates only at last tile in a frame.
HCP_QP_STATUS_COUNT0	8'hC0	Min and Max Qp from HFQ per frame	Min and Max Qp per tile. (if read at TILE ROW then last TILE of the row's update will be available)
HCP_QP_STATUS_COUNT1	8'hC4	Cumulative Qp from HFQ per frame	Cumulative Qp per tile. (if read at TILE ROW then last TILE of the row's update will be available)
HCP_SLICE_COUNT	8'hC8	Slice count per frame	Slice count in a frame until current tile
HCP_BITSTREAM_BYTECOUNT_TILE	8'hCC	Tile Byte Count. Since read happens at frame boundary, it will reflect tile size of the last tile in a frame	Current Tile Byte Count per tile. (if read at TILE ROW then last TILE of the row's update will be available)

Register Attributes Description

Host Register Attributes gives the defined register tags and their description.

Host Register Attributes

Tag	Name	Description
R/W	Read/Write	Bit is read and writeable.
R/SW	Read/Special Write	Bit is readable. Write is only allowed once after a reset.
RO	Read Only	Bit is only readable, but writes have no effects.
WO	Write Only	Bit is only writeable, reads return zeros.
RV	Reserved	Bit is reserved and not visible. Reads will return 0, and writes have no effect.
NA	Not Accessible	This bit is not accessible.

HCP Decoder Register Map

This documents all HEVC Decoder MMIO Registers.

HCP Decoder Register Descriptions

The HCP implements the following MMIO registers. A description of the register including its address and DWord descriptions are provided.

Column Title3
HCP Decode Status
HCP_CABAC_Status
HCP Last Position
HCP PMU Status
HCP Picture Checksum cIdx0
HCP Picture Checksum cIdx1
HCP Picture Checksum cIdx2
HCP_Picture_CRC

HCP Encoder Register Descriptions

Register
HCP_BIN_CT - HCP Frame BitStream BIN Count
HCP_BITSTREAMSE_BITCOUNT_FRAME - Reported Bitstream Output Bit Count for Syntax Elements Only
HCP_CABAC_BIN_COUNT_FRAME - Reported Bitstream Output CABAC Bin Count Register
HCP_CABAC_INSERTION_COUNT - Reported Bitstream Output CABAC Insertion Count
HCP_MINSIZE_PADDING_COUNT - Bitstream Output Minimal Size Padding Count Report Register
HCP_IMAGE_STATUS_CONTROL - HCP Image Status Control
HCP_QP_STATUS_COUNT - HCP Qp Status Count
HCP_UNIT_DONE - HCP Unit Done
HCP_IMAGE_STATUS_MASK - HCP Image Status Mask
HCP_SLICE_COUNT
Reported Bitstream Output Byte Count per Tile

Acronyms and Applicable Standards

Acronyms and Abbreviations

The table below defines acronyms and abbreviations used in this document.

Acronyms

Acronym	Meaning
AAC	Advanced Audio Coding -- part of the MPEG specification, AAC is the latest development in audio compression. It provides higher-quality audio reproduction than MPEG-1 Layer 3 (MP3), while requiring nearly 50% less data. It is defined in ISO/IEC 13818-7.
ADSL	Asymmetrical Digital Subscriber Line -- an asymmetrical DSL technology that takes advantage of the one-way nature of most multimedia communication, and provides much faster data rates for downstream (to the subscriber) then the upstream.
API	Application Programming Interface -- a set of routines used by an application program to request and carry out low-level services performed by the operating system.
ARGB	Alpha Red Green Blue -- color channel components.
ARIB	Association of Radio Industries and Business -- designated by the Ministry of Public Management, Home Affairs, Posts and Telecommunications (MPHPT) in Japan. ARIB members include broadcasters, radio equipment manufacturers, telecommunication operators, and related organizations.
ASP	Advanced Simple Profile - MPEG4-2
ATSC	ATSC Advanced Television Systems Committee - an organization in US that establishes and promotes technical standards for advanced television systems, such as digital television (DTV).
BDU	Bit-stream Data Unit
BIST	Built In Self Test
BPP	Bits Per Pixel
BSD	Byte Stream Decoder
CA, CAM	Conditional Access, Conditional Access Module - the removable descrambling module implemented in digital cable or satellite television system. The data flows through the module, which can have any proprietary scrambling algorithm implemented, yet maintaining system interface compatibility. The CAMs are usually provided by the operators in the TV network.
CPU	Central Processing Unit
DAA	Direct Access Arrangement
DAC	Digital-to-Analog Converter
DDA	Digital Difference Analyzer
DDS	Direct Digital Synthesizer
DPB	Decoded Picture Buffer. This buffer holds the decoded pictures for reference and for output along with the currently decoding picture. This differs from the DPB in the standard, which only holds the decoded pictures for reference.
DVB	Digital Video Broadcasting -- a set of open worldwide standards that define digital broadcasting

Acronym	Meaning
	using existing satellite, cable, and terrestrial infrastructures. It uses MPEG-2 specification as a universal foundation and expands it with DVB data structures and processes DVB-compliant digital broadcasting and equipment is widely available to consumers and is indicated with the DVB logo.
DVB-S	Satellite television DVB standards, based on QPSK and 8-DPSK modulation.
DVB-T	Terrestrial television DVB standards, based on 2k and 8k OFDM modulation.
DVD	Digital Versatile Disc
DVD-R	Recordable DVD. Since different disk formats are currently in use, including DVD-R,DVD+R, they are collectively mentioned as DVD-R in this document
DVI	Digital Visual Interface standard (EIA/CEA-861A). The standard defines a method for sending digital video signals over DVI and OpenLDI interface specifications. The standard is fully backward compatible with earlier DVI standards. New features include carrying auxiliary video information, such as aspect ratio and native video format information.
DSL xDSL	Digital Subscriber Line - transmission of data over copper telephone lines capable of bringing high-bandwidth to subscribers. Many flavors of DSL are currently in use, which are collectively called xDSL throughout the document.
DSP	Digital Signal Processor
DST	Destination
DWord	A 32-bit word
ES	Elementary Streams -- the raw output of an encoder, containing only what is necessary for a decoder to approximate the original picture or audio.
FIFO	First in First Out
FIR	Finite Impulse Response
FPU	Floating Point Unit
FW	Firmware running on the decoder controller, as used in Volume 4 of the *Olo River Plus Silicon EAS*
IDR	Instantaneous Decoding Refresh
IEEE 1394 1394	IEEE 1394 or iLink* or FireWire* An IEEE electronics industry standard for connecting multimedia and computing Up to 63 devices can be attached to your PC via a single plug-and-socket connection.
IEEE 802.11 802.11	The Institute for Electronics and Electrical Engineers (IEEE) wireless network specification. 802.11g and 802.11a networks can transmit payload at the rates in excess 34Mbits/s and allow for the wireless transmission at distances from several dozen to several hundred feet indoors.
IF	Intermediate Frequency -- the fixed, relatively low-frequency carrier to which current programs are ported by the tuner.
GMCH	Graphics and Memory Control Hub -- a chip that connects the IA processor to memory and other components in PC.
HDD	Hard Disk Drive -- magnetic mass storage device used in media centers for audiovisual program recording.
HDMI	High Definition Multimedia Interface (HDMI). This interface is used between any audio/video source, such as a set-top box, DVD player, or A/V receiver, and an audio or video monitor, such as a DTV. HDMI supports standard, enhanced or high-definition video, plus multi-channel digital audio on a single cable. The format transmits all ATSC HDTV standards and supports eight-channel digital

Acronym	Meaning
	audio (at up to a 192kHz sampling rate), with bandwidth to spare for future enhancements.
HDTV	High-Definition Television -- HDTV specifically refers to the highest-resolution formats of the 18 total DTV formats, true HDTV is generally considered to be 1,080-line interlaced (1080i) or 720-line progressive (720p).
HSR	Hidden Surface Removal
HW	Hardware
I/F	Interface
IEEE	IEEE 32-bit Floating Point number format representation
ISP	Image Synthesis Processor -- A collective term to describe all components of the hidden surface removal operation within the PowerVR architecture.
LOD	Level Of Detail -- used in texturing calculations.
LSB	Least Significant Bit
LUT	Look-up table
MBAFF	Block Adaptive Field Frame mode
MFD	Multi-Format Decoder
MMU	Memory Management Unit
MMMC	Multi-port, Multi-channel Memory Controller
MSA	Intel Micro Signal Architecture -- microprocessor architecture combining the features of microcontroller and digital signal processor. MSA is used here as a synonym of the processor core used in Olo River Plus
MSB	Most Significant Bit
MPEG	Motion Picture Experts Group - Organization that develops standards for digital video and digital audio compression.
MPR	Inter Prediction Module
NAL	Network Abstraction Layer
NAL unit	Syntax structure in a H.264 stream
NTSC	National Television System Committee, North American 525-line analog broadcast TV standard.
NIM	Network Interface Module - the integrated tuner and digital demodulator in the (satellite) TV systems. The DVB NIMs output MPEG transport stream.
NOP	No operation
OEM	Original Equipment Manufacturer
OGL/OpenGL	Open GL application programming interface
PAL	Phase Alternation Line - TV standard used in Europe. PAL uses 625 lines per frame, a 25 frames per second update rate and YUV color encoding. The number of visible pixels for PAL video is 768 x 576.
PCI	Peripheral Component Interconnect bus, a bi-directional bus defined in PCI 2.x specification
PES	Packetized Elementary Streams -- packetized streams are the ES streams arranged in data packets with PES header starting every packet. The syntax of the ES and PES is defined in MPEG. See definition for ES.

Acronym	Meaning
PIP	Picture In Picture display mode
POD	Point of Deployment conditional access module -- the removable conditional access module defined in the OpenCable* specification in US.
PPS	Picture Parameter set
PTS	Presentation time stamp
PVR	Personal Video Recorder, also PDR or personal digital recorder -- an interactive TV-recording device that records programs in digital format and allows users to search for/record shows based on type (for instance all basketball games or all episodes of a particular program). Users can also pause, rewind, stop, or fast-forward live programs with only a small time lag.
PWL	Piece-wise Linear
PXD	Pixel Decoder Module
RF	Radio Frequency - usually, modulated carriers which can be directly received by the tuners of TVs or radio receivers
RISC	Reduced Instruction Set Computer
RHW	Reciprocal Homogenous W -- W is a 3-D coordinate representation like X Y Z
RSB	Row Store Buffer
RTL	Register Transfer Language/Level
SEI	Supplementary Enhancement Information
SIF	Semaphore Interface Module
SIMD	Single Instruction Multiple Data
SMPTE	Society of Motion Picture and Television Engineers
SOC	System on chip
SP	Simple Profile - MPEG4-2
SPS	Sequence Parameter set
SRC	Source
SDTV	Standard-Definition Television -- a digital television system that is similar to current analog TV standards in picture resolution and aspect ratio. Typical SDTV resolution is 480i or 480p.
STB	Set Top Box -- a device that effectively turns a television set into an interactive Internet device and/or allows the television to receive and decode digital television (DTV) broadcasts.
TA	Tile Accelerator
TS	MPEG-2 Transport Stream -- a sequence of 188-byte packets carrying the multi-program audiovisual data
TSP	Texture Shading Processor -- a collective term to describe all components of the texture, shading and pixel blending operations within the PowerVR architecture.
VCL	Video Coded Layer
VCXO	Voltage Controlled Crystal Oscillator
VGP/ VGP Lite	Vertex Geometry Processor
VLC	Variable length coded. This refers to the collection of coding techniques that are used in VC1, and include CABAC, CAVLC and Exp-Golomb.

Acronym	Meaning
VOL	Video Object Layer
VOP	Video Object Plane
WAN	Wide Area Network
WSS	Wide Screen Signaling
XDS	Extended Data Services -- data services sending data in line 21/283 of the analog NTSC TV signal
XSI	Intel(R) XScale(R) System Interconnect
X, Y, Z, W	3-D coordinate representations
YUV	YUV texture format, primarily for video formats

VP9 Register Definitions

This section describes the VP9 Register Definitions as follows:

- Register Attributes Description
- VP9 Register Map
- VP9 Encoder Register Descriptions

Register Attributes Description

Host Register Attributes gives the defined register tags and their description.

Host Register Attributes

Tag	Name	Description
R/W	Read/Write	Bit is read and writeable.
R/SW	Read/Special Write	Bit is readable. Write is only allowed once after a reset.
RO	Read Only	Bit is only readable, but writes have no effects.
WO	Write Only	Bit is only writeable, reads return zeroes.
RV	Reserved	Bit is reserved and not visible. Reads will return 0, and writes have no effect.
NA	Not Accessible	This bit is not accessible.

VP9 Encoder Register Descriptions

HCP_IMAGE_STATUS_MASK - HCP Image Status Mask

HCP_IMAGE_STATUS_CONTROL - HCP Image Status Control

HCP_UNIT_DONE - HCP Unit Done

MFX Pipe

MFC_AVC_PAK_OBJECT Command

PAK Object Inline Data Description

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. CABAC/CAVLC encoding
6. Bit stream packing
7. Intra and inter-Prediction decoding loop
8. Internal error handling

These state/parameter values may subject to change on a per-MB basis, and must be provided in each MFC_AVC_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_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
3	31	**ExtendedForm** This field specifies that **LumaIntraMode** and **RefPicSelect** are fully replicated in 4x4 and 8x8 sub-blocks respectively. This non-DXVA form is used for optimal kernel performance.
	30	Reserved: MBZ
	29:24	Reserved
	23	Reserved : MBZ (reserved for future use as ExternalMvBufFlag)
	22:20	MvFormat (Motion Vector Size). This field specifies the size and format of the output motion

DWord	Bit	Description
		vectors. This field is reserved (MBZ) when the IntraMbFlag = 1. The valid encodings are: 000 = 0: No motion vector 100 = 8MV: Four 8x8 motion vector pairs 110 = 32MV: 16 4x4 motion vector pairs Others are reserved. (The following encodings are intended for future usages: 001 = 1MV: one 16x16 motion vector 010 = 2MV: One 16x16 motion vector pair 011 = 4MV: Four 8x8 motion vectors 101 = 16MV: 16 4x4 motion vectors 111 = Packed, number of MVs is given by PackedMvNum.)
	19	CbpDcY. This field specifies if the Luma DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process. 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. 0 – no 4x4 DC-only Luma sub-block is present; either not in Intra16x16 MB mode or all DC coefficients are zero.
	18	CbpDcU. This field specifies if the Chroma Cb DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process. 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. 0 – no 2x2 DC-only Chroma Cb sub-block is present; all DC coefficients are zero.
	17	CbpDcV. This field specifies if the Chroma Cb DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process. 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. 0 – no 2x2 DC-only Chroma Cr sub-block is present; all DC coefficients are zero.
	16	Reserved: MBZ

DWord	Bit	Description
		(reserved for future use as ExternalResidBufFlag for turbo mode)
	15	Transform8x8Flag This field indicates that 8x8 transform is used for the macroblock. When it is set to 0, the current MB uses 4x4 transform. When it is set to 1, the current MB uses 8x8 transform. The transform_size_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 other conditions. This field is only allowed to be set to 1 for two conditions: It must be 1 if IntraMbFlag = INTRA and IntraMbMode = INTRA_8x8 It may be 1 if IntraMbFlag = INTER and there is no sub partition size less than 8x8 Otherwise, this field must be set to 0. 0: 4x4 integer transform 1: 8x8 integer transform
	14	FieldMbFlag This field specifies the field polarity of the current macroblock, as the mb_field_decoding_flag syntax element in AVC spec. This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode. It is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode. 0 = Frame macroblock 1 = Field macroblock
	13	IntraMbFlag This field specifies whether the current macroblock is an Intra (I) macroblock. I_PCM is considered as Intra MB. For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) 1: INTRA (intra macroblock)
	12:8	MbType5Bits 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 AVC Spec.
	7	FieldMbPolarityFlag This field indicates the field polarity of the current macroblock.

DWord	Bit	Description
		Within an 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. 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. This field is reserved and MBZ for a progressive frame picture. 0 = Current macroblock is a field macroblock from the top field 1 = Current macroblock is a field macroblock from the bottom field Programming Note: Here bits [26:24] (MbAffFieldFlag and FiedlMbPolarityFlag) match with bits [10:8] of the Media Block Read message descriptor, simplifying the programming for message generation, as when MbAffFieldFlag is "1", kernels need to override the original "frame" surface state set for MBAFF frame picture.
	6	Reserved: MBZ
	5:4	IntraMbMode This field is provided to carry information partially overlapped with MbType. This field is only valid if IntraMbFlag = INTRA, otherwise, it is ignored by hardware..
	3	Reserved: MBZ
	2	SkipMbFlag 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, indicating that a macroblock is inferred as a P_Skip (or B_Skip) in a P Slice (or B Slice). Hardware honors input MVs for motion prediction and forces CBP to zero. 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. This field can only be set to 1 for certain values of MbType. See details later. This field is only valid for an inter macroblock. Hardware ignores this field for an intra macroblock. 0 = not a skipped macroblock 1 = is coded as a skipped macroblock
	1:0	InterMbMode This field is provided to carry redundant information as that encoded in MbType. This field is only valid if IntraMbFlag =0, otherwise, it is ignored by hardware.
4	31:24	Reserved for future MbYCnt expansion.
	23:16	MbYCnt (Vertical Origin). This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks.

DWord	Bit	Description
		Format = U8 in unit of macroblock.
	15:8	Reserved for future MbXCnt expansion.
	31:16	Reserved
	15:8	Reserved for future MbXCnt expansion.
	15:8	Reserved
	7:0	MbXCnt (Horizontal Origin). This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
5	31:24	Reserved for future CbpAcUV exopansion for 4.2.2. and 4.4.4 For 4.2.2, [23:16] for U(Cb), and [31:24] for C(Cr). For 4.4.4, the field [31:16] is interpreted as CbpAdJ|CbpAcV for 16 sub-blocks.
	31:16	Cbp4x4V (Coded Block Pattern Cr) Only the lower 4 bits [3:0] are valid for 4:2:0. The 4x4 Cr sub-blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpCr bit assignment is cbpCr bit [3 - X] for sub-block_num X. 0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK. 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). For monochrome, this field is ignored. For 4.2.2, [23:16] for U(Cb), and [31:24] ignored. For 4.4.4, the definition is the same as for luma component: 1bit per 4x4 block.
	23:20	CbpAcV (Coded Block Pattern Cr) Only the lower 4 bits [3:0] are valid for 4:2:0. The 4x4 Cr sub-blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpCr bit assignment is cbpCr bit [3 - X] for sub-block_num X. 0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK. 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). For monochrome, this field is ignored.

DWord	Bit	Description
	19:16	Cbp4x4U (Coded Block Pattern Cb) Only the lower 4 bits [3:0] are valid for 4:2:0. The 4x4 Cb sub-blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpCb bit assignment is cbpCb bit [3 - X] for sub-block_num X. 0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK. 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). For monochrome, this field is ignored.
	15:0	Cbp4x4Y[bit 15:0] (Coded Block Pattern Y) For 4x4 sub-block (when Transform8x8flag = 0 or in intra16x16) : 16-bit cbp, one bit for each 4x4 Luma sub-block (not including the DC 4x4 Luma block in intra16x16) in a MB. The 4x4 Luma sub-blocks are numbered as blk0 1 4 5 bit15 14 11 10 blk2 3 6 7 bit13 12 9 8 blk8 9 12 13 bit7 6 3 2 blk10 11 14 15 bit5 4 1 0 The cbpY bit assignment is cbpY bit [15 - X] for sub-block_num X. For 8x8 block (when Transform8x8flag = 1) Only the lower 4 bits [3:0] are valid; the remaining upper bits [15:4] are ignored. The 8x8 Luma blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpY bit assignment is cbpY bit [3 - X] for block_num X. 0 in a bit - indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK. 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).

DWord	Bit	Description
	15:0	Only the lower 4 bits [3:0] are valid for 4:2:0. The 4x4 Cb sub-blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpCb bit assignment is cbpCb bit [3 - X] for sub-block_num X. 0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK. 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). For monochrome, this field is ignored. For 4.2.2, [7:0] for U(Cb), and [15:8] ignored. For 4.4.4, the definition is the same as for luma component: 1bit per 4x4 block.
6	31:28	Skip8x8Pattern 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. This field is only valid for a B slice. It is ignored by hardware for a P slice. Hardware also ignores this field for an intra macroblock. 0 in a bit – Corresponding MVs are sent in the bitstream 1 in a bit – Corresponding MVs are not sent in the bitstream
	27	EnableCoeffClamp 1 = the magnitude of coefficients of the current MB will be clamped based on the clamping matrix after quantization 0 = no clamping
	26	LastMbFlag 1 – the current MB is the last MB in the current Slice 0 – the current MB is not the last MB in the current SliceReserved MBZ.
	25	SkipMbConvDisable 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 Section Macroblock Type Conversion Rules 0 - Enable skip type conversion for the current macroblock 1 - Disable skip type conversion for the current macroblock
	24	Reserved MBZ.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td>23:16</td><td>Reserved. Ignored by HW, this field will be re-derived internally.
(was QpPrimeV. 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>Reserved. IIgnored by HW, this field will be re-derived internally.
(Was QpPrimeU. 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>
<tr><td></td><td>7:0</td><td>QpPrimeY
This is the per-MB QP value specified for the current MB.
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.
Note: This value may differ from the actual codes, when HW QRC is on</td></tr>
<tr><td>7 to 9</td><td>31:0
Each</td><td>For intra macroblocks, definition of these fields are specified in Inline data subfields for an Intra Macroblock.
For inter macroblocks, definition of these fields are specified in Inline data subfields for an Inter Macroblock .</td></tr>
<tr><td>10</td><td>31:24</td><td>MaxSizeInWord
PAK should not exceed this budget accumulatively, otherwise it will trickle the PANIC mode.</td></tr>
<tr><td></td><td>23:16</td><td>TargetSizeInWord
PAK should use this budget accumulatively to decide if it needs to limit the number of non-zero coefficients.</td></tr>
<tr><td></td><td>15:0</td><td>Lambda_Or_RoundingOffset
When TQEnb=1, this 16-bit unsigned value multiplied by 2 is used as a lambda for the rate-distortion cost estimation in Trellis quantization (TQ). If the upper 4 bits are all set to 1 (0xFXXX), TQ is disabled and the regular quantizer is used. Thus, the valid range is 0~0xEFFF.
When TQEnb=0 or the upper 4 bits are all set to 1, the lower 4-bit value indicates the rounding precision (offset) for the regular quantizer
<table>
<tr><th>Value</th><th>Name</th></tr>
<tr><td>0000b</td><td>RoundInterEnable, RoundInter, RoundIntraEnable, and RoundIntra defined in MFC_AVC_SLICE_STATE are used as rounding precision.</td></tr>
<tr><td>1000b</td><td>+1/16</td></tr>
</table></td></tr>
</table>

DWord	Bit	Description
		1001b +2/16 1010b +3/16 1011b +4/16 1100b +5/16 1101b +6/16 1110b +7/16 1111b +8/16

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
7	31:16	LumaIntraMode[1] Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. See the bit assignment table later in this section.
	15:0	LumaIntraMode[0] Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 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. 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. See the bit assignment table later in this section.
8	31:16	LumaIntraMode[3] Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. See the bit assignment table later in this section.
	15:0	LumaIntraMode[2] Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.

<table>
<tr><th>Dword</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>See the bit assignment later in this section.</td></tr>
<tr><td>9</td><td>31:8</td><td>Reserved : MBZ

(Reserved for encocder turbo mode IntraResidueDataSize, when this is not 0, optional residue data are provided to the PAK; Reserved for decoder)</td></tr>
<tr><td></td><td>7:0</td><td>IntraStruct

This field contains 6 bits for IntraPredAvailFlags[5:0] and 2 bits for ChromaIntraPredMode. The IntraPredAvailFlags[4:0] (the lower 5 bits) have already included the effect of the constrained_intra_pred_flag. See the diagram later for the definition of neighbor position around the current MB or MB pair (in MBAFF mode).

1 – IntraPredAvailFlagY, indicates the values of samples of neighbor Y can be used in intra prediction for the current MB.

0 – IntraPredAvailFlagY, indicates the values of samples of neighbor Y is not available for intra prediction of the current MB.

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).

IntraPredAvailFlag-F is used only if

it is in MBAFF mode, i.e. MbaffFrameFlag = 1,

the current macroblock is of frame type, i.e. MbFieldFag = 0, and

the current macroblock type is Intra8x8, i.e.

IntraMbFlag = INTRA, IntraMbMode = INTRA_8x8, and Transform8x8Flag = 1.

In any other cases IntraPredAvailFlag-A shall be used instead.

<table>
<tr><th>Bits</th><th>IntraPredAvailFlags 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 (Left neighbor top half)</td></tr>
<tr><td>5</td><td>IntraPredAvailFlagE – E (Left neighbor bottom half)</td></tr>
<tr><td>4</td><td>IntraPredAvailFlagB – B (Top neighbor)</td></tr>
<tr><td>3</td><td>IntraPredAvailFlagC – C (Top right neighbor)</td></tr>
<tr><td>2</td><td>IntraPredAvailFlagD – D (Top left corner neighbor)</td></tr>
<tr><td>1:0</td><td>ChromaIntraPredMode – 2 bits to specify 1 of 4 chroma intra prediction modes, see the table in later section.</td></tr>
</table>
</td></tr>
</table>

Inline data subfields for an Inter Macroblock

DWord	Bit	Description
7	31:16	Reserved : MBZ
	15:8	SubMbPredMode (Sub-Macroblock Prediction Mode): If InterMbMode is INTER8x8, this field describes the prediction mode of the sub-partitions in the four 8x8 sub-macroblock. It contains four subfields each with 2-bits, corresponding to the four 8x8 sub-macroblocks in sequential order. This field is derived from sub_mb_type for a BP_8x8 macroblock. This field is derived from MbType for a non-BP_8x8 inter macroblock, and carries redundant information as MbType). If InterMbMode is INTER16x16, INTER16x8 or INTER8x16, this field carries the prediction modes of the sub macroblock (one 16x16, two 16x8 or two 8x16). The unused bits are set to zero. Bits [1:0]: SubMbPredMode[0] Bits [3:2]: SubMbPredMode[1] Bits [5:4]: SubMbPredMode[2] Bits [7:6]: SubMbPredMode[3]
	7:0	SubMbShape (Sub Macroblock Shape) 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. This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA). Otherwise, this field is ignored by hardware Bits [1:0]: SubMbShape[0] – for 8x8 Block 0 Bits [3:2]: SubMbShape[1] – for 8x8 Block 1 Bits [5:4]: SubMbShape[2] – for 8x8 Block 2 Bits [7:6]: SubMbShape[3] – for 8x8 Block 3 Blocks of the MB is numbered as follows : 01 23 Each 2-bit value [1:0] is defined as : 00 – SubMbPartWidth=8, SubMbPartHeight=8 01 – SubMbPartWidth=8, SubMbPartHeight=4 10 – SubMbPartWidth=4, SubMbPartHeight=8 11 – SubMbPartWidth=4, SubMbPartHeight=4
8	31:24	RefPicSelect[0][3]

DWord	Bit	Description
		Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	23:16	RefPicSelect[0][2] Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	15:8	RefPicSelect[0][1] Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	7:0	RefPicSelect[0][0] Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
9	31:24	RefPicSelect[1] [3] Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table.
	23:16	RefPicSelect[1][2] Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table.
	15:8	RefPicSelect[1][1] Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table.
	7:0	RefPicSelect[1][0] Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table.

Luma Intra Prediction Modes

Luma Intra Prediction Modes (LumaIntraPredModes) is defined in Definition of LumaIntraPredModes. It is further categorized as Intra16x16PredMode (Definition of Intra16x16PredMode), Intra8x8PredMode (Definition of Intra8x8PredMode) and Intra4x4PredMode (Definition of Intra4x4PredMode), operating on 16x16, 8x8 and 4x4 block sizes, respectively. The Figure illustrates the intra prediction directions geometrically for the Intra4x4 prediction. When a macroblock is subdivided, the intra prediction is performed for the subdivision in a predetermined order. For example, Numbers of Block4x4 in a 16x16 region shows the block order for Intra4x4 prediction. And Numbers of Block4x4 in an 8x8 region or

numbers of Block8x8 in a 16x16 region shows the block order of Block8x8 in a 16x16 region or Block4x4 in an 8x8 region.

Definition of LumaIntraPredModes

LumaIntraPredModes [index]		Intra16x16PredMode	Intra8x8PredMode	Intra4x4PredMode
Index	Bit	MbType = [1...24] Transform8x8Flag = 0	MbType = 0 Transform8x8Flag = 1	MbType = 0 Transform8x8Flag = 0
0	15:12	MBZ	Block8x8 3	Block4x4 3 (0_0)
	11:8	MBZ	Block8x8 2	Block4x4 2 (0_1)
	7:4	MBZ	Block8x8 1	Block4x4 1 (0_2)
	3:0	Block16x16	Block8x8 0	Block4x4 0 (0_3)
1	15:12	MBZ	MBZ	Block4x4 7 (1_0)
	11:8	MBZ	MBZ	Block4x4 6 (1_1)
	7:4	MBZ	MBZ	Block4x4 5 (1_2)
	3:0	MBZ	MBZ	Block4x4 4 (1_3)
2	15:12	MBZ	MBZ	Block4x4 11 (2_0)
	11:8	MBZ	MBZ	Block4x4 10 (2_1)
	7:4	MBZ	MBZ	Block4x4 9 (2 2)
	3:0	MBZ	MBZ	Block4x4 8 (2_3)
3	15:12	MBZ	MBZ	Block4x4 15 (3_0)
	11:8	MBZ	MBZ	Block4x4 14 (3_1)
	7:4	MBZ	MBZ	Block4x4 13 (3_2)
	3:0	MBZ	MBZ	Block4x4 12 (3_3)

Definition of Intra16x16PredMode

Intra16x16PredMode	Description
0	Intra_16x16_Vertical
1	Intra_16x16_Horizontal
2	Intra_16x16_DC
3	Intra_16x16_Plane
4 – 15	Reserved

Definition of Intra8x8PredMode

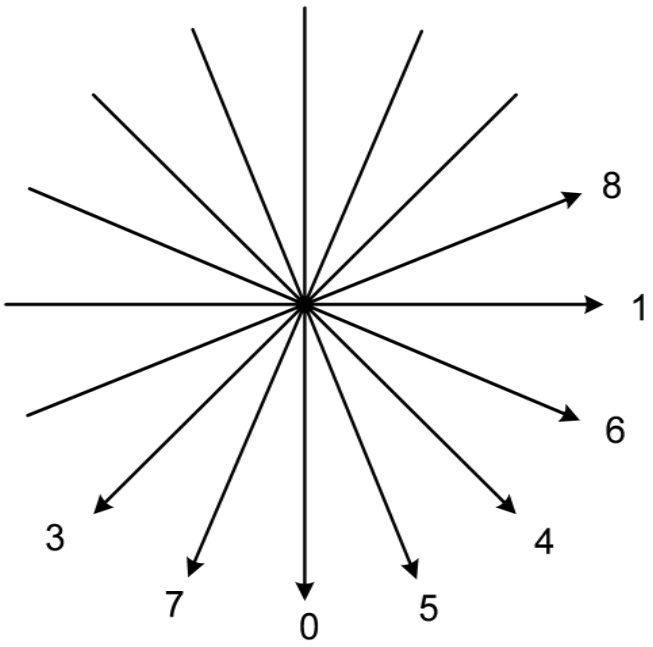
Intra8x8PredMode	Description
0	Intra_8x8_Vertical
1	Intra_8x8_Horizontal
2	Intra_8x8_DC
3	Intra_8x8_Diagonal_Down_Left

Intra8x8PredMode	Description
4	Intra_8x8_Diagonal_Down_Right
5	Intra_8x8_Vertical_Right
6	Intra_8x8_Horizontal_Down
7	Intra_8x8_Vertical_Left
8	Intra_8x8_Horizontal_Up
9 – 15	Reserved

Definition of Intra4x4PredMode

Intra4x4PredMode	Description
0	Intra_4x4_Vertical
1	Intra_4x4_Horizontal
2	Intra_4x4_DC
3	Intra_4x4_Diagonal_Down_Left
4	Intra_4x4_Diagonal_Down_Right
5	Intra_4x4_Vertical_Right
6	Intra_4x4_Horizontal_Down
7	Intra_4x4_Vertical_Left
8	Intra_4x4_Horizontal_Up
9 – 15	Reserved

Intra_4x4 prediction mode directions



Numbers of Block4x4 in a 16x16 region

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Numbers of Block4x4 in an 8x8 region or numbers of Block8x8 in a 16x16 region

0	1
2	3

Definition of Chroma Intra Prediction Mode

ChromaIntraPredMode (intra_chroma_pred_mode)	Name of intra_chroma_pred_mode
0	Intra_Chroma_DC (prediction mode)
1	Intra_Chroma_Horizontal (prediction mode)
2	Intra_Chroma_Vertical (prediction mode)
3	Intra_Chroma_Plane (prediction mode)

Reference Indices defined for each MB partition type and Bit Assignment

		frame/field MB/Picture			
MB partitioning	16x16	16x8	8x16	8x8	
RefIdxL0/1[0]	blk0	blk0	blk0	blk0	Bit 7:0
RefIdxL0/1[1]	x	blk1	blk1	blk1	Bit 15:8
RefIdxL0/1[2]	x	x	x	blk2	Bit 23:16
RefIdxL0/1[3]	x	x	x	blk3	Bit 31:24

MB Neighbor Availability in Intra-Prediction Modes (IntraPredAvailFlags)

Current MB is labelled as X. For non-MBAFF mode, 4 neighbors, A, B, C, D, are depicted in the following picture and are defined as the following.

- MB D: top left neighbor of current MB X
- MB C: top right neighbor of current MB X
- MB B: top neighbor of current MB X
- MB A: left neighbor of the current MB X

mbAddrD D (top-left)	**mbAddrB** **B** (top)	**mbAddrC** **C** (top-right)
mbAddrA A (left)	**X** **CurrMbAddrX**	**N/A**
N/A	**N/A**	**N/A**

For MBAFF mode, the current MB is labelled as X0 or X1, 4 neighbor pairs, A0/A1, B0/B1, C0/C1, D0/D1, are depicted in the following picture and are defined as the following.

- MB D0: first MB of top left neighbor MB pair of current MB pair X0/X1
- MB D1: second MB of top left neighbor MB pair of current MB pair X0/X1
- MB C0: first MB of top right neighbor MB pair of current MB pair X0/X1
- MB C1: second MB of top right neighbor MB pair of current MB pair X0/X1
- MB B0: first MB of top neighbor MB pair of current MB pari X0/X1
- MB B1: second MB of top neighbor MB pair of current MB pari X0/X1
- MB A0: first MB of left neighbor MB pair of the current MB pair X0/X1
- MB A1: second MB of left neighbor MB pair of the current MB pair X0/X1

mbAddrD D0	mbAddrB B0	mbAddrC C0
mbAddrD+1 D1	mbAddrB+1 B1	mbAddrC+1 C1
mbAddrA A0	CurrMbAddrX X0 or	N/A
mbAddrA+1 A1	CurrMbAddrX X1	N/A

For a given macroblock X (or X0/X1), the 6 neighbor availability signals, namely, A, B, C, D, E, F, are defined as the following.

- IntraPredAvailFlagF – F: (Single neighbor pixel at the left 8th row (-1,7)
- IntraPredAvailFlagA – A (Left neighbor top half pixel group)
- IntraPredAvailFlagE – E (Left neighbor bottom half pixel group)
- IntraPredAvailFlagB – B (Top neighbor pixel group)
- IntraPredAvailFlagC – C (Top right neighbor pixel group)
- IntraPredAvailFlagD – D (Top left corner neighbor pixel)

The following table depicts the generation of IntraPredAvailFlags[5:0] signals in a condensed form. It should note that for most cases only one input neighbor signal is assigned for each condition. The exception is in the four places for deriving left neighbor A and E where the neighbor is only available if left neighbors (A0 and A1) are both available (A0&A1). Also note that F takes output value very similar to that for A except the two "AND" conditions, where F is assigned to A1 instead of (A0&A1).

Definition of intra-prediction neighbor availability calculation in MBAFF mode

Output è		D		B		C		A		E		F		
Current X \ Neighbor Y		Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	
X_0 (Top)	X-Frame	D_1	D_1	B_1	B_1	C_1	C_1	A_0	A_0 & A_1	A_0	A_0 & A_1	A_0	A_1	
	X-Field	D_1	D_0	B_1	B_0	C_1	C_0	A_0	A_0	A_1	A_0	A_0	A_0	
X_1 (Bottom)	X-Frame	A_0	A_1	X_0	N/A	0	0	A_1	A_0 & A_1	A_1	A_0 & A_1	A_1	A_1	
	X-Field	D_1	D_1	B_1	B_1	C_1	C_1	A_0	A_1	A_1	A_1	A_0	A_1	

In Definition of intra-prediction neighbor availability calculation in MBAFF mode, X-Frame or X-Field indicates the frame/field mode of the current MB; and Y-Frame or Y-Field indicates the corresponding neighbor MB for the given neighbor location, being upper left (D) or left (A) for example. Therefore, "Y-" takes the selected neighbor MB name as in the output cell in the same column. For example, for output D, if X1 is a frame MB, Y = A, if X1 is a field MB, Y = D.

For non-MBAFF mode, as A0=A1, B0=B1, C0=C1 and D0=D1, the neighbor assignment is degenerated into the following simple table. Here, E is assigned to the same as A and F is forced to 0.

Definition of intra-prediction neighbor availability calculation in non-MBAFF mode

Output è	D	B	C	A	E	F
X	D0	B0	C0	A0	A0	0

To further explain the neighbor assignment rules in Definition of intra-prediction neighbor availability calculation in MBAFF mode, the following table provides description for each condition. Please note that this table is informative as it provides redundant information as in Definition of intra-prediction neighbor availability calculation in MBAFF mode.

Detailed explanation of intra-prediction neighbor availability calculation in MBAFF mode

Current MB	Current MB Field	Neighbor MB Field	Neighbor MB Select (Y=?)	Neighbor Avail Result (OUTPUT)	Description
				D	
X0	X-Frame	Y-Frame	D	D1	Top Frame MB uses [-1,-1] = D_31, thus D1 only, regardless D frame or field pair
(Top)	X-Frame	Y-Field	D	D1	
	X-Field	Y-Frame	D	D1	Top Field MB uses [-1,-2] = D_30, thus if D is frame pair, takes D1 (D1_14 pixel), and if D is field pair, takes D0 (D0_15 pixel)
	X-Field	Y-Field	D	D0	
X1	X-Frame	Y-Frame	A	A0	Bottom Frame MB uses [-1,15] = A_15, thus A0 (A0_15 pixel) if A is a frame pair, or A1 (A1_7 pixel), if A is a field pair
(Bottom)	X-Frame	Y-Field	A	A1	
	X-Field	Y-Frame	D	D1	Bottom Field MB uses [-1,-1] = D_31, thus D1 only, regardless D frame or field pair
	X-Field	Y-Field	D	D1	
				B	
X0	X-Frame	Y-Frame	B	B1	Top Frame MB uses [0…15,-1] = B_31, thus B1 only, regardless B frame or field pair
(Top)	X-Frame	Y-Field	B	B1	
	X-Field	Y-Frame	B	B1	Top Field MB uses [0…15,-2] = B_30, thus if B is frame pair, takes B1 (B1_14 row), and if B is field pair, takes B0 (B0_15 row)
	X-Field	Y-Field	B	B0	
X1	X-Frame	Y-Frame	X	X0	Bottom Frame MB uses [0…15,15], thus X0 (X0_15 row)
(Bottom)	X-Frame	Y-Field	X	n/a	Note: X0 and X1 must have the same field type, this row is n/a.
	X-Field	Y-Frame	B	B1	Bottom Field MB uses [0…15,-1] = B_31, thus B1 only, regardless B frame or field pair
	X-Field	Y-Field	B	B1	
				C	
X0	X-Frame	Y-Frame	C	C1	Top Frame MB uses [16…23,-1] = C_31, thus C1 only, regardless C frame or field pair
(Top)	X-Frame	Y-Field	C	C1	
	X-Field	Y-Frame	C	C1	Top Field MB uses [16…23,-2] = C_30, thus if C is frame pair, takes C1 (C1_14 row), and if C is field pair, takes C0 (C0_15 row)
	X-Field	Y-Field	C	C0	
X1	X-Frame	Y-Frame	n/a	0	Bottom Frame MB doesn't have left-top neighbor by definition, thus forced to 0
(Bottom)	X-Frame	Y-Field	n/a	0	
	X-Field	Y-Frame	C	C1	Bottom Field MB uses [16…23,-1] = C_31, thus C1 only, regardless C frame or field pair
	X-Field	Y-Field	C	C1	

				A	
X0	X-Frame	Y-Frame	A	A0	First Half of Top Frame MB uses [-1,0…7], thus A0 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
(Top)	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A0	First Half of Top Field MB uses [-1,0..2..4..14], thus take A0 (if A is frame pair, takes A0 even lines, and if A is field pair, takes A0 first half)
	X-Field	Y-Field	A	A0	
X1	X-Frame	Y-Frame	A	A1	First Half of Bottom Frame MB uses [-1,16…23], thus A1 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
(Bottom)	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A0	First Half of Bottom Field MB uses [-1,1..3..15], thus take A0 (if A is frame pair, takes A0 odd lines, and if A is field pair, takes A1 first half)
	X-Field	Y-Field	A	A1	
				E	
X0	X-Frame	Y-Frame	A	A0	Second Half of Top Frame MB uses [-1,8…15], thus A0 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
(Top)	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A1	Second Half of Top Field MB uses [-1,16..18..30], thus take A1 (if A is frame pair, takes A1 even lines, and if A is field pair, takes A0 second half)
	X-Field	Y-Field	A	A0	
X1	X-Frame	Y-Frame	A	A1	Second Half of Bottom Frame MB uses [-1,24…31], thus A1 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
(Bottom)	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A1	Second Half of Bottom Field MB uses [-1,17..19..31], thus takes A1 (if A is frame pair, takes A1 odd lines, and if A is field pair, takes A1 second half)
	X-Field	Y-Field	A	A1	
				F	
X0	X-Frame	Y-Frame	A	A0	Top Frame MB uses [-1,7] = A_7 (odd location), thus A0 if A is frame pair and A1 if field pair
(Top)	X-Frame	Y-Field	A	A1	
	X-Field	Y-Frame	A	A0	Top Field MB uses [-1,14] = A_14 (even location), thus A0 regardless A frame or field pair
	X-Field	Y-Field	A	A0	
X1	X-Frame	Y-Frame	A	A1	Bottom Frame MB uses [-1,23] = A_23 (odd location), thus A1 regardless A frame or field pair
(Bottom)	X-Frame	Y-Field	A	A1	
	X-Field	Y-Frame	A	A0	Bottom Field MB uses [-1,15] = A_15 (odd

Current MB	Current MB Field	Neighbor MB Field	Neighbor MB Select (Y=?)	Neighbor Avail Result (OUTPUT)	Description
				D	
	X-Field	Y-Field	A	A1	location), thus A0 if A is frame pair and A1 if A is field pair

Macroblock Type for Intra Cases

MbType follows two different tables according to whether the macroblock is an inter or intra macroblock according to IntraMbFlag.

For an intra macroblock, MbType, as defined in MbType definition for Intra Macroblock, carries redundant information as IntraMbMode. The notation I_16x16_x_y_z used in the table, 'x' is Intra16x16LumaPredMode, 'y' is ChromaCbpInd, and 'z' is LumaCbpInd, as defined in Sub field definition used by MbType for a macroblock with Intra16x16 prediction.

MbType definition for Intra Macroblock

Macroblock Type	MbType
I_4x4	0
I_8x8	0
I_16x16_0_0_0	1
I_16x16_1_0_0	2
I_16x16_2_0_0	3
I_16x16_3_0_0	4
I_16x16_0_1_0	5
I_16x16_1_1_0	6
I_16x16_2_1_0	7
I_16x16_3_1_0	8
I_16x16_0_2_0	9
I_16x16_1_2_0	Ah
I_16x16_2_2_0	Bh
I_16x16_3_2_0	Ch
I_16x16_0_0_1	Dh
I_16x16_1_0_1	Eh
I_16x16_2_0_1	Fh
I_16x16_3_0_1	10h
I_16x16_0_1_1	11h
I_16x16_1_1_1	12h
I_16x16_2_1_1	13h
I_16x16_3_1_1	14h

I_16x16_0_2_1	15h
I_16x16_1_2_1	16h
I_16x16_2_2_1	17h
I_16x16_3_2_1	18h
I_PCM	19h (used by HW)

Note: MbType here is identical as specified in DXVA 2.0.

For Intra_16x16 modes, the 5 bits of value (MbType – 1) have the following meanings.

Sub field definition used by MbType for a macroblock with Intra16x16 prediction

Bits	Description
4	LumaCbpInd – Luma Coded Block Pattern Indicator 0 means none of the luma blocks are coded. 1 means that at least one luma block is coded. 0 = SUBMODE_I16_L_0 1 = SUBMODE_I16_L_NZ In VME output, this field is forced to be 1 before adding 1 in Intra_16x16 mode.
3:2	ChromaCbpInd – Chroma Coded Block Pattern Indicator 00 means none of chroma blocks are coded. 01 means that only the chroma DC block is coded, but all AC blocks are not coded. 10 means that at least one AC chroma block is coded. 00 = SUBMODE_I16_C_0 01 = SUBMODE_I16_C_DC 10 = SUBMODE_I16_C_NZ 11 = Reserved In VME output, this field is forced to be 10 before adding 1 in Intra_16x16 mode. Programming Note: Adding 1 to MbType by VME hardware may have carry in to this field. But as '11' is reserved, the carry-in doesn't propagate into bit 4 or higher. This allows software to update MbType, if desired, using the redundant LumaIntraPredModes information.
1:0	Intra16x16PredMode – Intra16x16 Prediction Mode These two bits carries redundant (identical) information as that in LumaIntraPredModes[0][0]. 0 = SUBMODE_I16_VER 1 = SUBMODE_I16_HOR 2 = SUBMODE_I16_DC 3 = SUBMODE_I16_PLANE

IntraMbMode definition

IntraMbMode [1:0]	Description	Supported by VME?	Used by PAK?
0	INTRA_16x16 (redundant with MbType)	Yes	Ignored
1	INTRA_8x8	Yes	Yes
2	INTRA_4x4	Yes	Yes
3	IPCM (redundant with MbType)	No	Ignored

As an alternative representation, MbType is logically the same as the following, except the I_PCM and I_NxN (i.e. I_4x4 and I_8x8) cases:

- 24 types of 16x16 intra modes: A+B+C+D:(1h – 18h)

MBTYPE_INTRA_16x16 1hA

- 4 Intra16x16 modes:

SUBMODE_I16_VER 0B

SUBMODE_I16_HOR 1B

SUBMODE_I16_DC 2B

SUBMODE_I16_PLN 3B

- 3 Chroma Cbp indices:

SUBMODE_I16_C_0 0C

SUBMODE_I16_C_DC 4C

SUBMODE_I16_C_NZ 8C

- 2 Luma Cbp indices:

SUBMODE_I16_L_0 0D

SUBMODE_I16_L_NZ ChD

Macroblock Type for Inter Cases

Sub-Macroblock Prediction Mode, SubMbPredMode, indicates the prediction mode for the sub-partitions. Prediction mode specifies prediction direction being forward (from L0), backward (from L1) or bi-directional (from both L0 and L1). Its meaning depends on InterMbMode. Definition of SubMbPredMode[i] provides the definition of the field.

- If InterMbMode is INTER16x16, only SubMbPredMode[0] is valid, it describes the prediction mode of the 16x16 macroblock. The other entries are set to zero by hardware.
- For AVC, SubMbPredMode[0] contains redundant information as encoded in MbType parameter.
- Note: SubMbPredMode[1]-[3] are intentionally set to zero to allow a simple LUT to derive MbType as described later.

- If InterMbMode is INTER16x8, and INTER8x16, only the first two entries SubMbPredMode[0] and SubMbPredMode[1] are valid, describing the sub-macroblock prediction mode.
- For AVC, SubMbPredMode[0]/[1] contains redundant information as encoded in MbType parameter.
- Note: SubMbPredMode[2]-[3] are intentionally set to zero to allow a simple LUT to derive MbType as described later.
- If InterMbMode is INTER8x8, each entry of SubMbPredMode describes the prediction mode of the sub-partition of an 8x8 sub-macroblock.
- For AVC, SubMbPredMode can be derived from sub_mb_type field for BP_8x8 macroblocks as defined in AVC spec.
- Note on Direct Sub-macroblock Prediction Mode: Direct prediction is not conveyed through SubMbPredMode, instead, it is carried through Direct8x8Pattern.

InterMbMode definition

MbSkipFlag	InterMbMode	Description
0	0	INTER16x16
0	1	INTER16x8
0	2	INTER8x16
0	3	INTER8x8
1	0	PSKIP/BSKIP16x16*
1	3	BSKIP
1	1, 2	Reserved
Used by PAK	Ignored by PAK	

* BSKIP16x16 is an optional non-standard but equivalent optimization.

Definition of SubMbPredMode based on InterMbMode

SubMbPredMode	INTER16x16	INTER16x8	INTER8x16	INTER8x8
Bit	MbType = [1…3]	MbType = [16h]	MbType = [4…15h]	MbType = [16h]
7:6	MBZ	MBZ	MBZ	Block8x8 3
5:4	MBZ	MBZ	MBZ	Block8x8 2
3:2	MBZ	Block16x8 1	Block8x16 1	Block8x8 1
1:0	Block16x16	Block16x8 0	Block8x16 0	Block8x8 0
	Ignored by PAK	Ignored by PAK	Ignored by PAK	Used by PAK

Definition of SubMbPredMode[i]

SubMbPredMode	Description	InterMbMode	VME Output	MvCountPred	Notes
0	Pred_L0	All	Yes	1	P or B Slice
1	Pred_L1	All	Yes	1	B Slice Only
2	BiPred	All	Yes	2	B Slice Only
3	Reserved	Reserved	Reserved	Reserved	Reserved

Sub-Macroblock Shape, SubMbShape[i], for i = 0…3, describes the shape of the sub partitions of the 8x8 sub-macroblock of a BP_8x8 macroblock. This field is only valid if InterMBMode is INTER8x8. They are defined in Definition of SubMbShape for an 8x8 region of a BP_8x8 macroblock (including BSKIP, BDIRECT). The parameters can be derived from sub_mb_type field as defined in AVC spec.

Note: These fields must be correctly set even for Direct or Skip 8x8 cases, the individual B_Direct_8x8 block is flagged by the Direct8x8Pattern variable.

Definition of SubMbShape for an 8x8 region of a BP_8x8 macroblock (including BSKIP, BDIRECT)

	Description			
SubMbShape	NumSubMbPart	SubMbPartWidth	SubMbPartHeight	MvCountShape
0	1	8	8	1
1	2	8	4	2
2	2	4	8	2
3	4	4	4	4

For an inter macroblock, MbType, carries redundant information as InterMbMode and SubMbPredMode. The next table provides the typical inter macroblock types and the following table Additional MbType definition with Direct/Skip for Inter Macroblock provides that with skip and direct modes. The definition of MbType for both P slice and B slice is the same and is equivalent to that for mb_type of a B slice in the AVC spec. As direct mode is indicated using a separate field Direct8x8Pattern, 0 is reserved for MbType.

Here, MVCount is the number of motion vectors actually encoded in the bitstream. It is informative. For a BP_8x8 or equivalent Skip/Direct macroblock, MVCount is the sum of the following term for the four 8x8 sub macroblock (with i = 0…3):

MvCountShape[i] * MvCountPred[i] * MvCountDirect[i]

where MvCountShape[i] is block count for sub macroblock [i], MvCountPred[i] is the motion vector count for each block of sub macroblock[i], and MvCountDirect[i] is the multipler for direct mode for B Slice, indicating whether motion vectors are coded or not. It must be set to 1 for P slice. For B Slice, MvCountDirect[i] = !Direct8x8Pattern[i], which is 0 for a sub macroblock coded as direct mode and 1 otherwise.

In the tables, “DC” stands for “Don’t Care” as PAK hardware ignores these fields.

MbType definition for Inter Macroblock (and MbSkipflag = 0)

Macroblock Type	MbType	MbSkipFlag	Direct8x8Pattern	SubMbShape	SubMbPredMode	MVCount
Reserved	0	-	-	-	-	-
BP_L0_16x16	1	0	0	DC	DC	1
B_L1_16x16	2	0	0	DC	DC	1
B_Bi_16x16	3	0	0	DC	DC	2
BP_L0_L0_16x8	4	0	0	DC	DC	2
BP_L0_L0_8x16	5	0	0	DC	DC	2
B_L1_L1_16x8	6	0	0	DC	DC	2
B_L1_L1_8x16	7	0	0	DC	DC	2
B_L0_L1_16x8	8	0	0	DC	DC	2
B_L0_L1_8x16	9	0	0	DC	DC	2
B_L1_L0_16x8	0Ah	0	0	DC	DC	2
B_L1_L0_8x16	0Bh	0	0	DC	DC	2
B_L0_Bi_16x8	0Ch	0	0	DC	DC	3
B_L0_Bi_8x16	0Dh	0	0	DC	DC	3
B_L1_Bi_16x8	0Eh	0	0	DC	DC	3
B_L1_Bi_8x16	0Fh	0	0	DC	DC	3
B_Bi_L0_16x8	10h	0	0	DC	DC	3
B_Bi_L0_8x16	11h	0	0	DC	DC	3
B_Bi_L1_16x8	12h	0	0	DC	DC	3
B_Bi_L1_8x16	13h	0	0	DC	DC	3
B_Bi_Bi_16x8	14h	0	0	DC	DC	4
B_Bi_Bi_8x16	15h	0	0	DC	DC	4
BP_8x8	16h	0	!= Fh	vary	vary	Sum
Reserved	17h-1Fh	-	-	-	-	-

Additional MbType definition with Direct/Skip for Inter Macroblock

Macroblock Type	MbType	Xfrm 8x8	MbSkipFlag	Direct8x8Pattern	SubMbShape	SubMbPredMode	MvCount	Notes
P_Skip_16x16	1	-	1	DC	DC	DC	0	Skipped macroblock. Motion compensation like P_L0_16x16
B_Skip_16x16_4MVPair	16h	Vary	1	Fh	0	vary	0	Skipped macroblock. Motion compensation like B_8x8 with 8x8 subblocks, when direct_8x8_inference_flag is set to 1
B_Skip_16x16_16MVPair	16h	0	1	Fh	FFh	vary	0	Skipped macroblock. Motion compensation like B_8x8 with 4x4 subblocks, when direct_8x8_inference_flag is set to 0
B_Direct_16x16_4MVPair	16h	vary	0	Fh	0	vary	0	MbType coded as B_Direct_16x16. Motion compensation like B_8x8 with 8x8 subblocks, when direct_8x8_inference_flag is set to 1
B_Direct_16x16_16MVPair	16h	0	0	Fh	FFh	vary	0	MbType coded as B_Direct_16x16. Motion compensation like B_8x8 with 4x4 subblocks, when direct_8x8_inference_flag is set to 0

People might notice that B_DIRECT_16x16 and B_SKIP are mapped on BP_8x8 for AVC decoding interface in IT mode as the motion compensation operation for both modes are the same as BP_8x8. According to AVC Spec, motion vectors for B_DIRECT_16x16 and B_SKIP are derived from temporally co-located

macroblock on an 8x8 sub macroblock basis if direct_8x8_inference_flag is set to 1 or on a 4x4 block basis if it is set to 0. For each sub macroblock or block, SubMbPredMode is derived, thus can any of the valid numbers. Motion vectors may also be different. In spatial direct mode, the motion vectors are subject to spatial neighbor macroblocks as well as co-located macroblock. The spatial prediction is based on the neighbor macroblocks, so the same spatial predicted motion vector applies to all sub macroblocks or blocks. However, under certain conditions, temporal predictor may replace (colZeroFlag) the spatial predictor for a given sub macroblock or block. Thus the motion vectors may differ.

In MbType definition for Inter Macroblock (and MbSkipflag = 0), the macroblock type names for major partitions nicely follow forms BP_MbPredMode_MbShape (like BP_L0_16x16) and B_MbPredMode0_MbPredMode1_MbShape (like B_L0_Bi_16x8). For minor partitions it is fixed as BP_MbShape as BP_8x8.

However, in Additional MbType definition with Direct/Skip for Inter Macroblock the macroblock types for Skip and Direct modes does not follow the same rule. The third field in P_Skip_16x16 or B_Direct_16x16_x indicates that “Skip” or “Direct” applies to the entire 16x16 macroblock, even though MbShape is 8x8 as that in BP_8x8. In order to distinguish the SubMbShape being 8x8 or 4x4 for B_Skip and B_Direct, the fourth field is added. 4MVPair indicates upto 4 MV pairs are presented with SubMbShape equals to 0; and 16MVPair indicates up to 16 MV pairs are presented with SubMbShape equals to FFh.Also note that P_8x8ref0 is not specified in PAK input interface, it is up to hardware to detect and choose its packing format based on number of reference indices and reference index for the given macroblock.

Macroblock Type Conversion Rules

For improved coding efficiency the PAK hardware has the capability to convert macroblock types to use more efficiency coding modes such as DIRECT and SKIP. For an inter macroblock or a sub macroblock coded as DIRECT, no motion vector is needed in the bitstream for the macroblock or sub macroblock. If a macroblock is coded as SKIP, it only consumes one SKIP bit (no motion vector, no coefficients are coded). And infomaton about the macroblock is ‘inferred’ according to the rules stated in the AVC Spec.

As the input to PAK, the following signals can convey the information regarding DIRECT and SKIP:

- MbSkipFlag
- Direct8x8Pattern
- CodecBlockPattern (CbpY, CbpCb, CbpCr)

Such conversion can be enabled or disabled through the SLICE_STATE fields DirectConvDisable and SkipConvDisable as well as the in line command field MbSkipConvDisable.

A P slice doesn’t support direct mode, it only supports P_Skip, which is equivalent to a 16_16_L0 prediction. Other prediction types cannot be converted to P_Skip. The following table describes the macroblock type conversion rules for a P slice. Here CBP = CbpY/CbpCb/CbpCr are the final computed results after quantization by the hardware. Note that hardware honors the input CbpY/CbpCb/CbpCr fields – if the value corresponding to a block is set to zero, the resulting CBP is also zero. The output mb_skip_flag and mb_type are the symbols coded in the bitstream as defined in the AVC spec. “DC” stands for “Don’t care”, “T” for “True”.

Note that the internal condition of MV==MVP is subject to the precise rules stated in the AVC Spec as quoted below. Note that there are exceptions for P_Skip from the normal motion vector prediction rules.

Derivation process for luma motion vectors for skipped macroblocks in P and SP slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0.

The reference index refIdxL0 for a skipped macroblock is derived as follows.

refIdxL0 = 0. (8-168)

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

– The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.

– The variable mvL0 is specified as follows.

– If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0.

– mbAddrA is not available

– mbAddrB is not available

– refIdxL0A is equal to 0 and both components of mvL0A are equal to 0

– refIdxL0B is equal to 0 and both components of mvL0B are equal to 0

– Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.

NOTE – The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

Macroblock type conversion rule for an inter macroblock in a P slice

Input		Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
P_Skip_16x16	DC	DC	DC	1	1	-	Forced to P_Skip; Hardware will force CBP to zero and also ignore SkipConvDisable control. Hardware doesn't check for MV==MVP error condition
P_Skip_16x16	DC	DC	DC	0	0	0	Reverse convert to P_L0_16x16; Hardware will force CBP to zero but reversely convert MbType as P_L0_16x16 once it determines that Skip is not allowed.
BP _16x16_L0	0	0	T	1	1	-	Converted to P_Skip. Even input doesn't provide skip hint, hardware can performance the optimization by detecting CBP and MV==MVP condition.
BP _16x16_L0	0	0	T	0	0	0	Reverse back to P_L0_16x16; Hardware will reverse back to P_L0_16x16 even Skip conditions are met once it determines that Skip is not allowed.
BP _16x16_L0	1	0	T	T	0	0	Still coded as P_L0_16x16 = 0.

A B slice supports both direct and skip modes. The following table describes the macroblock type conversion rules for a B slice. Hardware does not verify MV==MVP condition for a Skip/Direct macroblock in a B Slice as no DMV is performed by hardware.

Macroblock type conversion rule for an inter macroblock in a B slice

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
B_Skip_8x8 B_Skip_4x4	DC	DC	DC	n/a	1	1	-	Forced to B_Skip; Hardware will force CBP to zero and also ignore SkipConvDisable control.
B_Skip_8x8 B_Skip_4x4	DC	DC	DC	n/a	0	0	0	REVERSE convert to B_Direct_16x16; Hardware will force CBP to zero and also reverse convert to B_Direct_16x16 when it discovers Skip is not allowed.
B_Direct_16x16_4MVPair/16MVPair	0	0	0	n/a	1	1	-	Converted to B_Skip. Hardware first converts to B_Direct_16x16 and then further to B_Skip if CBP = 0.
B_Direct_16x16_4MVPair/16MVPair	0	0	0	n/a	0	0	0	Converted to B_Direct_16x16. Hardware

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
								first converts to B_Direct_16x16 and stop there as it discovers Skip is not allowed even CBP=0.
B_Direct_16x16_4MVPair/16MVPair	1	0	0	n/a	DC	0	0	Converted to B_Direct_16x16. Hardware converts to B_Direct_16x16 and stops there even though CBP = 0 as input disallows Skip conversion.
B_Direct_16x16_4MVPair/16MVPair	DC	0	NZ	n/a	DC	0	0	Converted to B_Direct_16x16. Hardware converts to B_Direct_16x16 and stops there because CBP != 0.
B_Direct_16x16_4MVPair/16MVPair	DC	1	DC	n/a	DC	0	16h	Stay as B_8x8. Hardware stays at

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
								B_8x8 and codes each sub macroblocks even all are direct.

The internal signal MbAffSkipAllowed is added to deal with a restriction on the frame/field flag (MbFieldFlag) which is unique to MBAFF. MbAffSkipAllowed is always set to 1 in non-MBAFF modes. In MBAFF mode, a macroblock pair may be both skipped only if its MbFieldFlag is the same as its available neighbor macroblock pair A or B if A or B is available (in that order), or is not 0 if A/B are both not available. Otherwise, one of the macroblocks in the pair must be coded.

To reduce the burden on software, PAK hardware handles the above restriction correctly. For the first MB in a pair, MbAffSkipAllowed is always set to 1. Therefore, hardware allows converting the first MB to Skip if skip conversion is enabled. For the second MB in a pair, hardware sets MbAffSkipAllowed to 0 if the following is true:

- The current MB Pair has different MbFieldFlag than its available neighbor A or B if A or B is available, or is not 0 if A/B are both not available
- And the first MB is coded as a SKIP (could be forced or converted)

Otherwise, it sets MbAffSkipAllowed to 1. As MbAffSkipAllowed is to 0 for the above condition, hardware will disallow Skip mode for the second MB. If the input signal forces it to Skip, hardware performs reverse-convertion to code it as P_L0_16x16 or B_Direct_16x16 with CBP = 0 for a macroblock in a P or B Slice. This means that hardware is able to correct the programming mistake by software. If the macroblock is not forced to skip, hardware simply disallows Skip conversion.

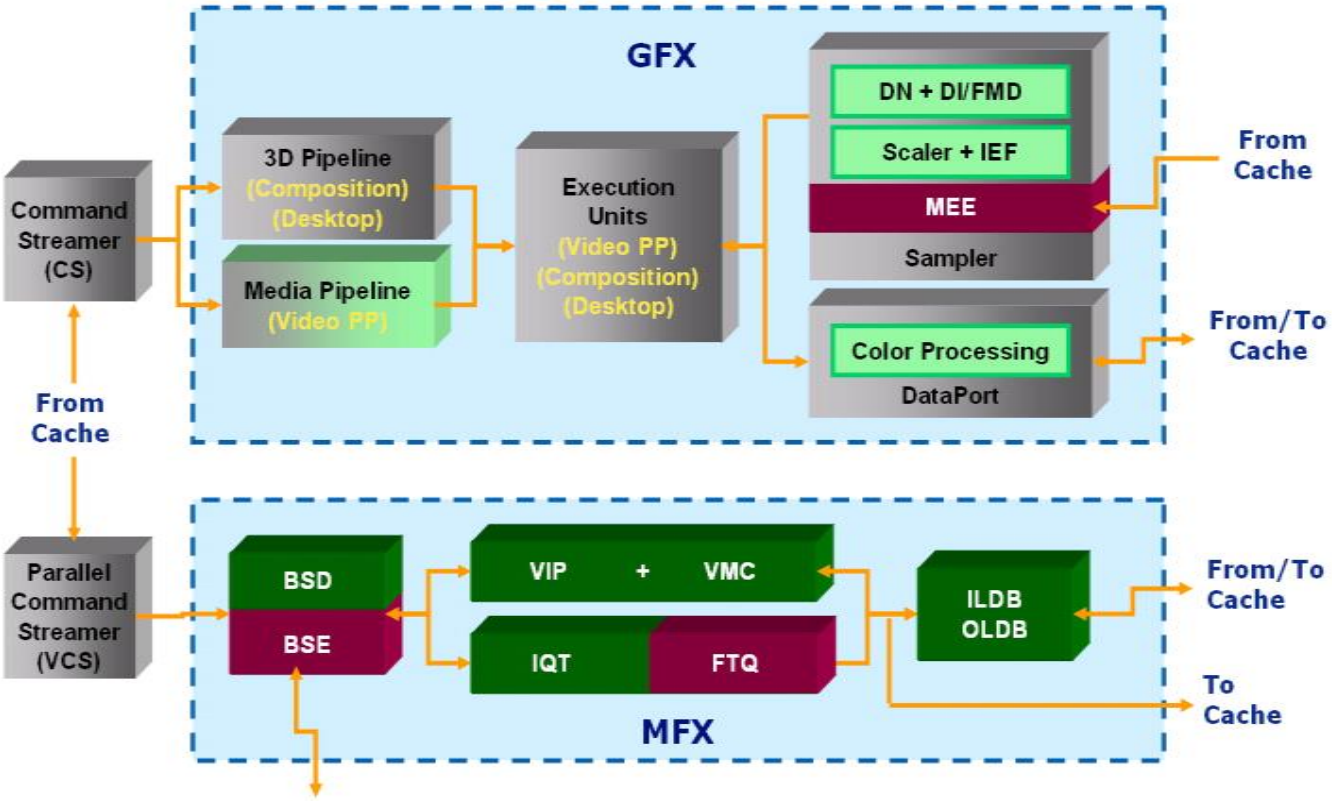
Software still has an option to disallow Skip Conversion on a per-MB basis using the MbSkipConvDisable control field in the inline command.

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).

Note: MFC only supports AVC (H.264).

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.

The **VME**(*Video Motion Estimation*) is located next to all image processing units, such as DN (*denoise*) 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

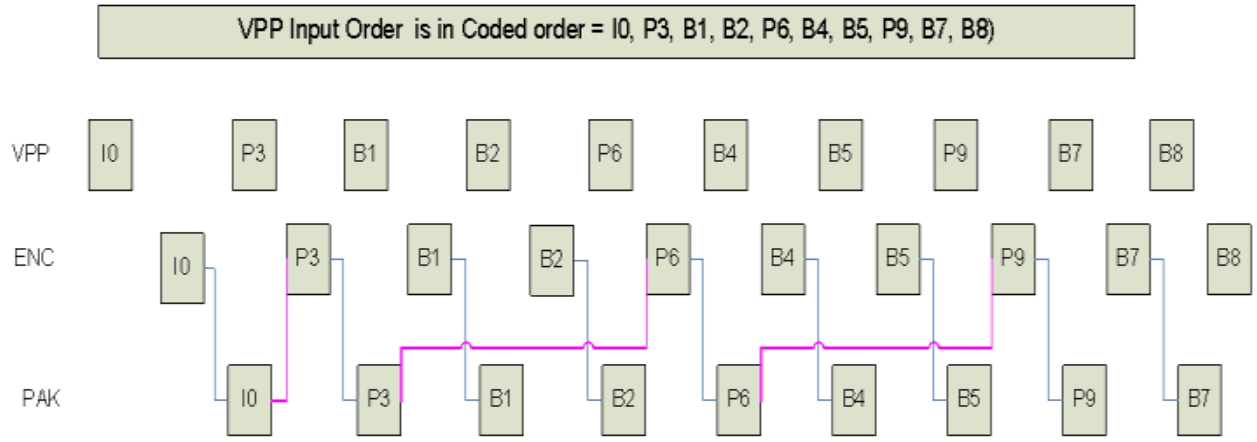
This section describes an architectural solution to map the first two steps in the GFX engine and the last step in the MFX engine. Since this mapping involves two OS-visible engines, managing them in parallel under one application is similar to the solution in earlier generations. Each engine has its 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
- 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.
    - 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: • 8-bit pixel precision of source images • loadable 2 AC and 2 DC Huffman tables • 3 loadable quantization matrix for Y, U, V • Interleaved and non-interleaved Scans • Single and multiple Scans	Extended DCT-based mode, Lossless, Hierarchical modes: 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 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) Chroma 4:1:1 Chroma 4:2:0 Chroma horizontal 4:2:2 Chroma vertical 4:2:2 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 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 usual.
        - 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 usual.
        - 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
    - 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

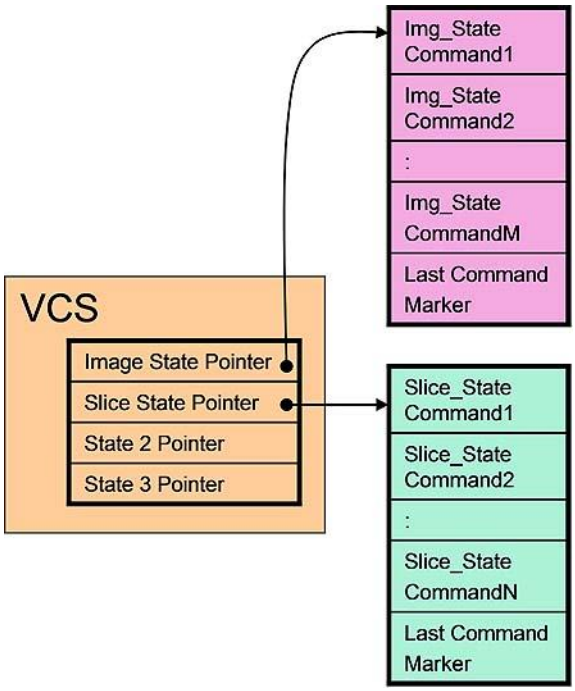
MFD hardware pipeline supports 3 different codec standards : AVC, VC1 and MPEG2. To make the interface flexible, each codec is designed with its own set of commands.

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

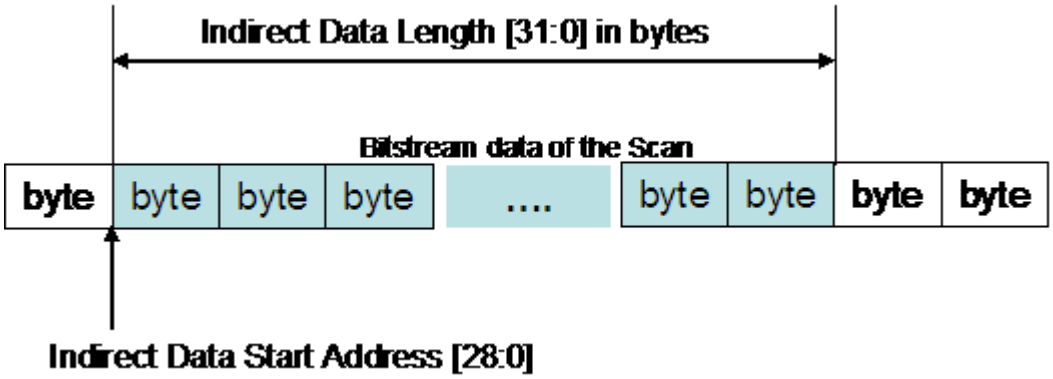
Bitstream buffer starts right at the very first MB data.

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 neighbor 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 GFXPIPE[28:27 = 1h, 26:24 = 1h, 23:16 = 04h] (Single DW, Non-pipelined)
	15:1	**Reserved:** MBZ
	0	**Pipeline Select** 0: 3D pipeline is selected 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
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)	Subop B (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
	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

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	Subop B (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable ?
2h	1h	0h	5h	MFX_AVC_WEIGHTOFFSET_STATE	MFX	SLICE	n/a
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)**					
2h	3h	0h	0h	MFX_MPEG2_PIC_STATE	MFX	IMAGE	n/a
2h	3h	0h	1-1Fh	Reserved	n/a	n/a	n/a

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
	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
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 Encoder Commands Sequence (Examples)

Example of AVC Encoder

Example of MVC Encoder

Example of MPEG2 Encoder

Example of VP8 Encoder

Example of JPEG Encoder

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_WAIT

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.

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).

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

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:

For JPEG encoder, 16-bit precision is used for each element 1/QM matrix. The 32 DWords are used for 64 QM elements with the following data structure:

	Bits 15:0	Bits 31:16
DWord1	1/QM[0][0]	1/QM[1][0]
DWord2	1/QM[2][0]	1/QM[3][0]
DWord3	1/QM[4][0]	1/QM[5][0]
DWord4	1/QM[6][0]	1/QM[7][0]
DWord5	1/QM[0][1]	1/QM[1][1]
DWord6	1/QM[2][1]	1/QM[3][1]
DWord7	1/QM[4][1]	1/QM[5][1]
DWord8	1/QM[6][1]	1/QM[7][1]
...		
DWord31	1/QM[4][7]	1/QM[5][7]
DWord32	1/QM[6][7]	1/QM[7][7]

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/VC1/MPEG2/JPEG/VP8 Decoder/Encoder:

[BSD is bitstream decoder rowstore; MPR is Motion Prediction rowstore; IP is Intra Prediction rowstore; VLF is loop filter rowstore; VDE is VDENC rowstore]

Codec	Mode	Frame Width	BSD	MPR	IP	VLF	VDENC	BSD Addr	MPR Addr	IP Addr	VLF ADDR	VDENC ADDR
VDENC AVC	Frame	<= 4k	N	Y	Y	Y	Y	0	256	512	768	0
	Field/Mbaff	<= 4k	N	Y	Y	N	Y	0	512	1024	N/A	0
VC1 Dec		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
MPEG2		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
JPEG		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
VP8		<= 4k	Y	N	Y	Y	Y	0	N/A	256	512	1536

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_MPEG_TS_CONTROL

MAX_QP_DELTA: Maximum QP delta is the Magnitude of QP delta between passes.

MAX_QP_DELTA is selected such that cumulative QP over all possible passes shouldn't exceed 51.

Example Configurations:

MAX Number of Passes	MAX_QP_DELTA
4	0xc
5	0xa
6	0x8
7	0x7

Commands
MFX_AVC_DIRECTMODE_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	**CodedPatterDC** (for AVC only, **111b** for others) This field indicates whether DC coefficients are sent. 1 bit each for Y, U and V.
	16	**Reserved MBZ**
	15	**Transform8x8Flag** When it is set to 0, the current MB uses 4x4 transform. When it is set to 1, the current MB uses 8x8 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: This field is only allowed to be set to 1 for two conditions: It must be 1 if **IntraMbFlag** = INTRA and **IntraMbMode** = INTRA_8x8 It may be 1 if **IntraMbFlag** = INTER and there is no sub partition size less than 8x8 Otherwise, this field must be set to 0. 0: 4x4 integer transform 1: 8x8 integer transform
	14	**MbFieldFlagMbFieldFlag** This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode. This field is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode. Same as the mb_field_decoding_flag syntax element in AVC spec. >0 = Frame macroblock 1 = Field macroblock

DWord	Bit	Description
	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. I_PCM is considered as Intra MB. For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) 1: INTRA (intra macroblock)
	12-8	**MbType5Bits** 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** FieldMB Polarity - vctrl_vld_top_field AVC
	6	**Reserved MBZ**
	5:4	**IntraMbMode** This field is provided to carry information partially overlapped with MbType. This field is only valid if **IntraMbFlag** = INTRA, otherwise, it is ignored by hardware.
	3	**Reserved MBZ**
	2	**MbSkipFlag** Reserved MBZ (DXVA Encoder). HW (VDSunit) doesn't have skip MB info. 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.
	1:0	**InterMbMode** This field is provided to carry redundant information as that in MbType. It also carries additional information such as skip. This field is only valid if **IntraMbFlag** =INTER, otherwise, it is ignored by hardware.
1	31:16	**MbYCnt (Vertical Origin).** This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks. Format = U8 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. Format = U8 in unit of macroblock.
2	31	**Conceal MB Flag.** This field specifies if the current MB is a conceal MB, use in AVC/VC1/MPEG2 mode.
	30	**Last MB of the Slice Flag.** 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** 0 in a bit - indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all

DWord	Bit	Description
		coefficient values are zero) 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** 0 in a bit - indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 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** 0 in a bit - indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 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). Bit15=Y0Sub0, Bit0=Y3Sub3
3	31:28	**Skip8x8Pattern** /> 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. This field is only valid for a B slice. It is ignored by hardware for a P slice. Hardware also ignores this field for an intra macroblock. 0 in a bit - Corresponding MVs are sent in the bitstream 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. Range 0 to 384 (9 bits)
	15:8	MbClock16 - MB compute clocks in 16-clock unit.
	7	**mbz** (AVC) **/ QScaleType** (MPEG2)
	6:0	**QpPrimeY** (AVC) **/ QScaleCode** (MPEG2) 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 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 vcp_vds_mvdataR[162:159] VC1 vmd_vds_mt_vert_fld_selR[3:0] MPEG2
	19:12	**Reserved**
	11:10	**SubBlockCodeType V** (If 8x8, 8x4, 4x8, 4x4 type)

DWord	Bit	Description
	9:8	**SubBlockCodeType U** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	7:6	**SubBlockCodeType Y3** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	5:4	**SubBlockCodeType Y2** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	3:2	**SubBlockCodeType Y1** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	1:0	**SubBlockCodeType Y0** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
Inter Cases		
8	31:16	**MvFwd[0].y** - y-component of the forward motion vector of the 1 8x8 or 1 4x4 subblock
	15:0	**MvFwd[0].x** - x-component of the forward motion vector of the 1 8x8 or 1 4x4 subblock
9	31:0	**MvBck[0]** - the backward motion vector of the 1 8x8 or 1 4x4 subblock
10	31:0	**MvFwd[1]** - the forward motion vector of the 2 8x8 or 4 4x4 subblock
11	31:0	**MvBck[1]** - the backward motion vector of the 2 8x8 or 4 4x4 subblock
12	31:0	**MvFwd[2]** - the forward motion vector of the 3 8x8 or 8 4x4 subblock
13	31:0	**MvBck[2]** - the backward motion vector of the 3 8x8 or 8 4x4 subblock
14	31:0	**MvFwd[3]** - the forward motion vector of the 4 8x8 or 12 4x4 subblock
15	31:0	**MvBck[3]**> - the backward motion vector of the 4th 8x8 or 12 4x4 subblock
8 to 15	31:0	**Reserved MBZ**

Inline data subfields for an Intra Macroblock

DWord	Bit	Description
4	31:16	**LumaIndraPredModes[1]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
	15:0	**LumaIndraPredModes[0]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 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. 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.

DWord	Bit	Description
		AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
5 AVC INTRA	31:16	**LumaIndraPredModes[3]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
	15:0	**LumaIndraPredModes[2]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. AVC: See the bit assignment later in this section. VC1: MBZ. MPEG2: MBZ.
6	31:8	Reserved (Reserved for encoder turbo mode **IntraResidueDataSize**, when this is not 0, optional residue data are provided to the PAK; Reserved for decoder)
	7:0	**MbIntraStruct** 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). 1 - IntraPredAvailFlagX, indicates the values of samples of neighbor X can be used in intra prediction for the current MB. 0 - IntraPredAvailFlagX, indicates the values of samples of neighbor X is not available for intra prediction of the current MB. 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). IntraPredAvailFlag-F is used only if ○ it is in MBAFF mode, i.e. **MbaffFrameFlag** = 1, ○ the current macroblock is of frame type**,** i.e. **MbFieldFag** = 0, and ○ the current macroblock type is Intra8x8, that is, **IntraMbFlag** = INTRA, **IntraMbMode** = INTRA_8x8, and **Transform8x8Flag** = 1. In any other cases IntraPredAvailFlag-A shall be used instead.

DWord	Bit	Description
		<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** (Left neighbor top half)</td></tr><tr><td>5</td><td>**IntraPredAvailFlagE - E** (Left neighbor bottom half)</td></tr><tr><td>4</td><td>**IntraPredAvailFlagB - B** (Top neighbor)</td></tr><tr><td>3</td><td>**IntraPredAvailFlagC - C** (Top right neighbor)</td></tr><tr><td>2</td><td>**IntraPredAvailFlagD - D** (Top left corner neighbor)</td></tr><tr><td>1:0</td><td>**ChromaIntraPredMode -** 2 bits to specify 1 of 4 chroma intra prediction mode, see the table in later section.</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) 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. This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA) This field is derived from MbType for a non-BP_8x8 inter macroblock, and carries redundant information as MbType) Bits [1:0]: SubMbPredMode[0] - for 8x8 Block 0 Bits [3:2]: SubMbPredMode[1] - for 8x8 Block 1 Bits [5:4]: SubMbPredMode[2] - for 8x8 Block 2 Bits [7:6]: SubMbPredMode[3] - for 8x8 Block 3 Blocks of the MB is numbered as follows : 0 1 2 3 Each 2-bit value [1:0] is defined as : 00 - Pred_L0

DWord	Bit	Description
		01 - Pred_L1 10 - BiPred **For VC1:** Bits [1:0]: "00"= Y0 Forward only, "01"= Y0 Backward only, "10"= Y0 Bi direction Bits [3:2]: SubMbPredMode[1] - for 8x8 Block 1 Bits [5:4]: SubMbPredMode[2] - for 8x8 Block 2 Bits [7:6]: SubMbPredMode[3] - for 8x8 Block 3
	7:0	**SubMbShape**[bit 7:0] (Sub Macroblock Shape) 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. This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA) This field is forced to 0 for a non-BP_8x8 inter macroblock, and effectively carries redundant information as MbType). Bits [1:0]: SubMbShape[0] - for 8x8 Block 0 Bits [3:2]: SubMbShape[1] - for 8x8 Block 1 Bits [5:4]: SubMbShape[2] - for 8x8 Block 2 Bits [7:6]: SubMbShape[3] - for 8x8 Block 3 Blocks of the MB is numbered as follows: 0 1 2 3 Each 2-bit value [1:0] is defined as: 00 - SubMbPartWidth=8, SubMbPartHeight=8 01 - SubMbPartWidth=8, SubMbPartHeight=4 10 - SubMbPartWidth=4, SubMbPartHeight=8 11 - SubMbPartWidth=4, SubMbPartHeight=4 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]** 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. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are

DWord	Bit	Description
		generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ 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]** 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. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ 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]** 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. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ 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]** 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. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format.

DWord	Bit	Description
		Bit 6:5: reserved MBZ 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]** 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. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ 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]** 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. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ 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]** 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. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)

DWord	Bit	Description
	7:0	**Frame Store ID L1[0]** 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. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ 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

PAK Object Inline Data Description

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. CABAC/CAVLC encoding
6. Bit stream packing
7. Intra and inter-Prediction decoding loop
8. Internal error handling

These state/parameter values may subject to change on a per-MB basis, and must be provided in each MFC_AVC_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_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.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="4">3</td><td>30</td><td>Reserved: MBZ</td></tr>
<tr><td>19</td><td>CbpDcY. This field specifies if the Luma DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process.

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.

0 - no 4x4 DC-only Luma sub-block is present; either not in Intra16x16 MB mode or all DC coefficients are zero.

<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>PAK Object Inline Data Description - CbpDcY</td></tr><tr><td colspan="2">When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1 and svc slice state: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.</td></tr></table></td></tr>
<tr><td>18</td><td>CbpDcU. This field specifies if the Chroma Cb DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process.

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.

0 - no 2x2 DC-only Chroma Cb sub-block is present; all DC coefficients are zero.

<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>PAK Object Inline Data Description</td></tr><tr><td colspan="2">When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1 and svc slice state: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.</td></tr></table></td></tr>
<tr><td>17</td><td>CbpDcV. This field specifies if the Chroma Cb DC sub-block is coded. Setting it to 0 will force PAK to zero out the Luma sub-block. Otherwise, whether the sub-block is coded will be determined by the quantization process.

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.

0 - no 2x2 DC-only Chroma Cr sub-block is present; all DC coefficients are zero.</td></tr>
</table>

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th colspan="2">Programming Note</th></tr>
<tr><td>Context:</td><td>PAK Object Inline Data Description</td></tr>
<tr><td colspan="2">When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1 and svc slice state: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.</td></tr>
</table>
</td></tr>
<tr><td></td><td>16</td><td>Reserved: MBZ
(reserved for future use as ExternalResidBufFlag for turbo mode)</td></tr>
<tr><td></td><td>15</td><td>Transform8x8Flag
This field indicates that 8x8 transform is used for the macroblock.
When it is set to 0, the current MB uses 4x4 transform. When it is set to 1, the current MB uses 8x8 transform. The transform_size_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 other conditions.
This field is only allowed to be set to 1 for two conditions:
It must be 1 if IntraMbFlag = INTRA and IntraMbMode = INTRA_8x8
It may be 1 if IntraMbFlag = INTER and there is no sub partition size less than 8x8
Otherwise, this field must be set to 0.
<table>
<tr><th colspan="2">Programming Note</th></tr>
<tr><td>Context:</td><td>PAK Inline Object Data Description</td></tr>
<tr><td colspan="2">When SvcSliceState TcoeffLvlPredFlag=1, and AvcImgState EntropyCodingFlag is 1(CABAC), this field cannot be 1.</td></tr>
</table>
0: 4x4 integer transform
1: 8x8 integer transform</td></tr>
<tr><td></td><td>14</td><td>FieldMbFlag
This field specifies the field polarity of the current macroblock, as the mb_field_decoding_flag syntax element in AVC spec.
This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode. It is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode.
0 = Frame macroblock
1 = Field macroblock</td></tr>
<tr><td></td><td>13</td><td>IntraMbFlag
This field specifies whether the current macroblock is an Intra (I) macroblock. I_PCM is considered as Intra MB.</td></tr>
</table>

DWord	Bit	Description
		For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) 1: INTRA (intra macroblock)
	12:8	**MbType5Bits** This field is encoded to match with the best macroblock mode determined as described in the next section. It follows a unified encoding for inter and intra macroblocks according to AVC Spec.
	7	**FieldMbPolarityFlag** This field indicates the field polarity of the current macroblock. Within an 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. 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. This field is reserved and MBZ for a progressive frame picture. 0 = Current macroblock is a field macroblock from the **top** field 1 = Current macroblock is a field macroblock from the **bottom** field *Programming Note: Here bits [26:24] (MbAffFieldFlag and FiedlMbPolarityFlag) match with bits [10:8] of the Media Block Read message descriptor, simplifying the programming for message generation, as when MbAffFieldFlag is "1", kernels need to override the original "frame" surface state set for MBAFF frame picture.*
	6	MB Reserved: Inter MB converted to IPCM. This field is for HW purposes only. SW should not use it.
	5:4	**IntraMbMode** This field is provided to carry information partially overlapped with MbType. This field is only valid if **IntraMbFlag** = INTRA, otherwise, it is ignored by hardware.
	3	Reserved: MBZ
	2	**SkipMbFlag** 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, indicating that a macroblock is inferred as a P_Skip (or B_Skip) in a P Slice (or B Slice). Hardware honors input MVs for motion prediction and forces CBP to zero. 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

DWord	Bit	Description
		described in the later sub sections. This field can only be set to 1 for certain values of MbType. See details later. This field is only valid for an inter macroblock. For intra MB (bit 13 of this DW set to one), this bit must be set to zero. 0 = not a skipped macroblock 1 = is coded as a skipped macroblock
	1:0	**InterMbMode** This field is provided to carry redundant information as that encoded in MbType. This field is only valid if **IntraMbFlag** =0, otherwise, it is ignored by hardware.
4	31:16	Cbp4x4Y[bit 15:0] (Coded Block Pattern Y) For 4x4 sub-block (when Transform8x8flag = 0 or in intra16x16) : 16-bit cbp, one bit for each 4x4 Luma sub-block (not including the DC 4x4 Luma block in intra16x16) in a MB. The 4x4 Luma sub-blocks are numbered as blk0 1 4 5 bit15 14 11 10 lk2 3 6 7 bit13 12 9 8 blk8 9 12 13 bit7 6 3 2 blk10 11 14 15 bit5 4 1 0 The cbpY bit assignment is cbpY bit [15 - X] for sub-block_num X. For 8x8 block (when Transform8x8flag = 1) Only the lower 4 bits [3:0] are valid; the remaining upper bits [15:4] are ignored. The 8x8 Luma blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpY bit assignment is cbpY bit [3 - X] for block_num X. 0 in a bit - indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK. 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).

DWord	Bit	Description
		Programming Note **Context:** PAK Object Inline Data Description When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1 and svc slice state: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.
4	15:8	MbYCnt (Vertical Origin). This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
	7:0	MbXCnt (Horizontal Origin). This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
5	31:16	Cbp4x4V (Coded Block Pattern Cr) Only the lower 4 bits [3:0] are valid for 4:2:0. The 4x4 Cr sub-blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0 The cbpCr bit assignment is cbpCr bit [3 - X] for sub-block_num X. 0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK. 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). For monochrome, this field is ignored. For 4.2.2, [23:16] for U(Cb), and [31:24] ignored. For 4.4.4, the definition is the same as for luma component: 1bit per 4x4 block. **Programming Note** **Context:** PAK Object Inline Data Description When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1 and svc slice state: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.
5	15:0	Cbp4x4U (Coded Block Pattern Cb) Only the lower 4 bits [3:0] are valid for 4:2:0. The 4x4 Cb sub-blocks are numbered as blk0 1 bit3 2 blk2 3 bit1 0

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>The cbpCb bit assignment is cbpCb bit [3 - X] for sub-block_num X.

0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero), or force to zero for PAK.

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).

For monochrome, this field is ignored.

For 4.2.2, [7:0] for U(Cb), and [15:8] ignored.

For 4.4.4, the definition is the same as for luma component: 1bit per 4x4 block.

<table><tr><th colspan="2">Programming Note</th></tr><tr><td>Context:</td><td>PAK Object Inline Data Description</td></tr><tr><td colspan="2">When Reference Mb: IPCM or inferred IPCM, current mb: base mode flag = 1 and svc slice state: TcoeffLvlPredFlag = 1 ; all bits in CbpDcY, CbpDcU, CbpDcV, Cbp4x4Y[15:0], Cbp4x4V[15:0] and Cbp4x4U[15:0] must set to 1's.</td></tr></table></td></tr>
<tr><td>6</td><td>31:28</td><td>Skip8x8Pattern

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.

This field is only valid for a B slice. It is ignored by hardware for a P slice. Hardware also ignores this field for an intra macroblock.

0 in a bit - Corresponding MVs are sent in the bitstream

1 in a bit - Corresponding MVs are not sent in the bitstream</td></tr>
<tr><td></td><td>27</td><td>EnableCoeffClamp

1 = the magnitude of coefficients of the current MB will be clamped based on the clamping matrix after quantization

0 = no clamping</td></tr>
<tr><td></td><td>26</td><td>LastMbFlag

1 - the current MB is the last MB in the current Slice

0 - the current MB is not the last MB in the current Slice - Reserved MBZ.</td></tr>
<tr><td></td><td>25</td><td>SkipMbConvDisable

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 Section Macroblock Type Conversion Rules.</td></tr>
</table>

DWord	Bit	Description
		0 - Enable skip type conversion for the current macroblock 1 - Disable skip type conversion for the current macroblock
	24	Reserved MBZ.
	23:16	**Reserved. Ignored by HW, this field will be re-derived internally**. (was **QpPrimeV.** 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.)
	15:8	**Reserved. Ignored by HW, this field will be re-derived internally**. (Was **QpPrimeU.** 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.)
	7:0	**QpPrimeY** This is the per-MB QP value specified for the current MB. 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. **Programming Note** **Context:** PAK Object Inline Data Description This value may differ from the actual codes, when HW QRC is on
7 .. 9	31:0 Each	For intra macroblocks, definition of these fields are specified in Inline data subfields for an Intra Macroblock. For inter macroblocks, definition of these fields are specified in Inline data subfields for an Inter Macroblock.
10	31:24	**MaxSizeInWord** PAK should not exceed this budget accumulatively, otherwise it will trickle the PANIC mode.
	23:16	**TargetSizeInWord** PAK should use this budget accumulatively to decide if it needs to limit the number of non-zero coefficients.
	15:0	**Lambda_Or_RoundingOffset** When **TQEnb**=1, in MFX_AVC_IMG_STATE, this 16-bit unsigned value multiplied by 2 is used as a lambda for the rate-distortion cost estimation in Trellis quantization (TQ). If the upper 4 bits are all set to 1 (0xFXXX), TQ is disabled and the regular quantizer is used. Thus, the valid range is 0~0xEFFF. When TQ is enabled per MB, the TQR in MFC_AVC_IMG_STATE is used for rounding quantization coefficients. When **TQEnb**=0 or the upper 4 bits are all set to 1, the lower 4-bit value indicates the rounding precision (offset) for the regular quantizer. The values ranging 0001b ~ 0111 are reserved.

DWord	Bit	Description
		<table><tr><th>Value</th><th>Name</th></tr><tr><td>0000b</td><td>**RoundInterEnable, RoundInter, RoundIntraEnable,** and **RoundIntra** defined in MFC_AVC_SLICE_STATE are used as rounding precision.</td></tr><tr><td>1000b</td><td>+1/16</td></tr><tr><td>1001b</td><td>+2/16</td></tr><tr><td>1010b</td><td>+3/16</td></tr><tr><td>1011b</td><td>+4/16</td></tr><tr><td>1100b</td><td>+5/16</td></tr><tr><td>1101b</td><td>+6/16</td></tr><tr><td>1110b</td><td>+7/16</td></tr><tr><td>1111b</td><td>+8/16</td></tr></table>**Format: U16**

VDEnc Mode Inline data (For PAK Standalone validation)

12	31:16	**MV Y** The value of the y component of this motion vector for FWD block 0.
	15:0	**MV X** The value of the x component of this motion vector for FWD block 0.
13	31:16	**MV Y** The value of the y component of this motion vector for FWD block 1.
	15:0	**MV X** The value of the x component of this motion vector for FWD block 1.
14	31:16	**MV Y** The value of the y component of this motion vector for FWD block 2.
	15:0	**MV X** The value of the x component of this motion vector for FWD block 2.
15	31:16	**MV Y** The value of the y component of this motion vector for FWD block 3.
	15:0	**MV X** The value of the x component of this motion vector for FWD block 3.

16	31:16	**MV Y** The value of the y component of this motion vector for BWD block 0.
	15:0	**MV X** The value of the x component of this motion vector for BWD block 0.
17	31:16	**MV Y** The value of the y component of this motion vector for BWD block 1.
	15:0	**MV X** The value of the x component of this motion vector for BWD block 1.
18	31:16	**MV Y** The value of the y component of this motion vector for BWD block 2.
	15:0	**MV X** The value of the x component of this motion vector for BWD block 2.
19	31:16	**MV Y** The value of the y component of this motion vector for BWD block 3.
	15:0	**MV X** The value of the x component of this motion vector for BWD block 3.
20	31:16	**LumaIntraMode[1]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.
	15:0	**LumaIntraMode[0]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 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. 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.
21	31:16	**LumaIntraMode[3]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.
	15:0	**LumaIntraMode[2]**

		Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.
22	31:16	**Minimal Distortion** This field contains the overall distortion for the source block associated with the winning MbType, which could be one of intra or inter modes.
	15:0	**SkipRawDistortion** This field contains Skip Raw Distortion which may be used by software to further refine the skip decision.
23	31:16	**InterRawDistortion** This field provides the Inter Raw Distortion (Sad/Haar) for the current macroblock.
	15:0	**BestIntraRawDistortion** This field contains the best IntraRawDistortion (Sad/Haar) for the current macroblock. The IntraMBMode will indicate if this is a16x16/8x8/4x4 distortion.

Inline data for LumaIntraMode

	0 or 1	0	0	1	1
ExtendedForm	Intra4x4	Intra8x8	Intra16x16	Intra8x8	Intra16x16
DW4 - 31:28	Block 7	-	-	-	Block 0
DW4 - 27:24	Block 6	-	-	-	Block 0
DW4 - 23:20	Block 5	-	-	-	Block 0
DW4 - 19:16	Block 4	-	-	-	Block 0
DW4- 15:12	Block 3	-	-	-	Block 0
DW4 - 11:8	Block 2	-	-	-	Block 0
DW4 - 7:4	Block 1	-	-	-	Block 0
DW4 - 3:0	Block 0	-	-	-	Block 0
DW5 - 31:28	Block 15	-	-	-	Block 0
DW5 - 27:24	Block 14	-	-	-	Block 0
DW5 - 23:20	Block 13	-	-	-	Block 0
DW5 - 19:16	Block 12	-	-	-	Block 0
DW5 - 15:12	Block 11	-	-	-	Block 0
DW5- 11:8	Block 10	-	-	-	Block 0
DW5 - 7:4	Block 9	-	-	-	Block 0
DW5 - 3:0	Block 8	-	-	-	Block 0

vctrl_pred_mode[63:0]	(vctrl_it_lumaintrapredmode3[15:0] & vctrl_it_lumaintrapredmode2[15:0] & vctrl_it_lumaintrapredmode1[15:0] & vctrl_it_lumaintrapredmode0[15:0]) : vctrl_pred_mode_noextend[63:0]
vctrl_pred_mode_noextend[63:0]	(vctrl_INTRA_vld_16x16mode & vctrl_it_Transform8x8Flag) ? vctrl_pred_mode_noextend_4x4[63:0] : vctrl_pred_mode_noextend_16x16[63:0] : vctrl_pred_mode_noextend_8x8[63:0] : vctrl_pred_mode_noextend_4x4[63:0]
vctrl_pred_mode_noextend_16x16[63:0]	vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0] & vctrl_it_lumaintrapredmode0[3:0]
vctrl_pred_mode_noextend_8x8[63:0]	"h000" & vctrl_it_lumaintrapredmode0[15:12] & "h000" & vctrl_it_lumaintrapredmode0[11:8] & "h000" & vctrl_it_lumaintrapredmode0[7:4] & "h000" & vctrl_it_lumaintrapredmode0[3:0]
vctrl_pred_mode_noextend_4x4[63:0]	vctrl_it_lumaintrapredmode3[15:0] & vctrl_it_lumaintrapredmode2[15:0] & vctrl_it_lumaintrapredmode1[15:0] & vctrl_it_lumaintrapredmode0[15:0]

Inline data for RefPicSelect

	0	0	0	0 or 1	1	1	1
ExtendedForm	16x16	16x8	8x16	8x8	16x16	16x8	8x16
DW8 - 31:24	-	-	-	L0 blk3	L0 blk0	-	L0 blk1
DW8 - 23:16	-	-	-	L0 blk2	L0 blk0	-	L0 blk0
DW8 - 15:8	-	L0 blk1	L0 blk1	L0 blk1	L0 blk0	-	L0 blk1
DW8 - 7:0	L0 blk0	L0 blk0	L0 blk0	L0 blk0	L0 blk0	-	L0 blk0
DW9 - 31:24	-	-	-	L1 blk3	L1 blk0	-	L1 blk1
DW9 - 23:16	-	-	-	L1 blk2	L1 blk0	-	L1 blk0
DW9 - 15:8	-	L1 blk1	L1 blk1	L1 blk1	L1 blk0	-	L1 blk1
DW9 - 7:0	L1 blk0	L1 blk0	L1 blk0	L1 blk0	L1 blk0	-	L1 blk0

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
7	31:16	**LumaIntraMode[1]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. See the bit assignment table later in this section.
	15:0	**LumaIntraMode[0]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 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. 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. See the bit assignment table later in this section.
8	31:16	**LumaIntraMode[3]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. See the bit assignment table later in this section.
	15:0	**LumaIntraMode[2]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. See the bit assignment later in this section.

<table>
<tr><th>Dword</th><th>Bit</th><th>Description</th></tr>
<tr><td>9</td><td>31:8</td><td>Reserved: MBZ
(Reserved for encocder turbo mode IntraResidueDataSize, when this is not 0, optional residue data are provided to the PAK; Reserved for decoder)</td></tr>
<tr><td></td><td>7:0</td><td>IntraStruct

This field contains 6 bits for IntraPredAvailFlags[5:0] and 2 bits for ChromaIntraPredMode. The IntraPredAvailFlags[4:0] (the lower 5 bits) have already included the effect of the constrained_intra_pred_flag. See the diagram later for the definition of neighbor position around the current MB or MB pair (in MBAFF mode).

1 - IntraPredAvailFlagY, indicates the values of samples of neighbor Y can be used in intra prediction for the current MB.

0 - IntraPredAvailFlagY, indicates the values of samples of neighbor Y is not available for intra prediction of the current MB.

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).

IntraPredAvailFlag-F is used only if

• It is in MBAFF mode, that is, MbaffFrameFlag = 1
• The current macroblock is of frame type, that is, MbFieldFag = 0
• The current macroblock type is Intra8x8, that is, IntraMbFlag = INTRA, IntraMbMode = INTRA_8x8, and Transform8x8Flag = 1

In any other cases IntraPredAvailFlag-A shall be used instead.

<table>
<tr><th>Bits</th><th>IntraPredAvailFlags 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 (Left neighbor top half)</td></tr>
<tr><td>5</td><td>IntraPredAvailFlagE - E (Left neighbor bottom half)</td></tr>
<tr><td>4</td><td>IntraPredAvailFlagB - B (Top neighbor)</td></tr>
<tr><td>3</td><td>IntraPredAvailFlagC - C (Top right neighbor)</td></tr>
<tr><td>2</td><td>IntraPredAvailFlagD - D (Top left corner neighbor)</td></tr>
<tr><td>1:0</td><td>ChromaIntraPredMode - 2 bits to specify 1 of 4 chroma intra prediction modes, see the table in later section.</td></tr>
</table>
</td></tr>
</table>

Inline data subfields for an Inter Macroblock

DWord	Bit	Description
7	31:16	**Reserved: MBZ**
	15:8	**SubMbPredMode (Sub-Macroblock Prediction Mode):** If **InterMbMode** is INTER8x8, this field describes the prediction mode of the sub-partitions in the four 8x8 sub-macroblock. It contains four subfields each with 2-bits, corresponding to the four 8x8 sub-macroblocks in sequential order. This field is derived from sub_mb_type for a BP_8x8 macroblock. This field is derived from **MbType** for a non-BP_8x8 inter macroblock, and carries redundant information as **MbType**). If **InterMbMode** is INTER16x16, INTER16x8 or INTER8x16, this field carries the prediction modes of the sub macroblock (one 16x16, two 16x8 or two 8x16). The unused bits are set to zero. Bits [1:0]: SubMbPredMode[0] Bits [3:2]: SubMbPredMode[1] Bits [5:4]: SubMbPredMode[2] Bits [7:6]: SubMbPredMode[3]
	7:0	**SubMbShape** (Sub Macroblock Shape) 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. This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA). Otherwise, this field is ignored by hardware Bits [1:0]: SubMbShape[0] - for 8x8 Block 0 Bits [3:2]: SubMbShape[1] - for 8x8 Block 1 Bits [5:4]: SubMbShape[2] - for 8x8 Block 2 Bits [7:6]: SubMbShape[3] - for 8x8 Block 3 Blocks of the MB is numbered as follows : 01 23 Each 2-bit value [1:0] is defined as : 00 - SubMbPartWidth=8, SubMbPartHeight=8 01 - SubMbPartWidth=8, SubMbPartHeight=4 10 - SubMbPartWidth=4, SubMbPartHeight=8 11 - SubMbPartWidth=4, SubMbPartHeight=4
8	31:24	**RefPicSelect[0][3]**

DWord	Bit	Description
		Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	23:16	**RefPicSelect[0][2]** Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	15:8	**RefPicSelect[0][1]** Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
	7:0	**RefPicSelect[0][0]** Support up to 4 reference pictures per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List0 Table.
9	31:24	**RefPicSelect[1] [3]** Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table. For P- picture these bits must be set to zero.
	23:16	**RefPicSelect[1][2]** Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table. For P- picture these bits must be set to zero.
	15:8	**RefPicSelect[1][1]** Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table. For P- picture these bits must be set to zero.
	7:0	**RefPicSelect[1][0]** Support up to 4 reference pictures per L1 direction, one per MB partition, if exists. See details in later section. This field specifies the reference index into the Reference Picture List1 Table. For P- picture these bits must be set to zero.

Luma Intra Prediction Modes

Luma Intra Prediction Modes (LumaIntraPredModes) is defined in Definition of LumaIntraPredModes. It is further categorized as Intra16x16PredMode, Intra8x8PredMode and Intra4x4PredMode, operating on 16x16, 8x8 and 4x4 block sizes, respectively. illustrates the intra prediction directions geometrically for the Intra4x4 prediction. When a macroblock is subdivided, the intra prediction is performed for the subdivision in a predetermined order. For example, Numbers of Block4x4 in a 16x16 region shows the block order for Intra4x4 prediction, and Numbers of Block4x4 in an 8x8 region or numbers of Block8x8 in a 16x16 region shows the block order of Block8x8 in a 16x16 region or Block4x4 in an 8x8 region.

Definition of LumaIntraPredModes

LumaIntraPredModes [index]		Intra16x16PredMode	Intra8x8PredMode	Intra4x4PredMode
Index	Bit	MbType = [1...24] Transform8x8Flag = 0	MbType = 0 Transform8x8Flag = 1	MbType = 0 Transform8x8Flag = 0
0	15:12	MBZ	**Block8x8 3**	**Block4x4 3 (0_0)**
	11:8	MBZ	**Block8x8 2**	**Block4x4 2 (0_1)**
	7:4	MBZ	**Block8x8 1**	**Block4x4 1 (0_2)**
	3:0	**Block16x16**	**Block8x8 0**	**Block4x4 0 (0_3)**
1	15:12	MBZ	MBZ	**Block4x4 7 (1_0)**
	11:8	MBZ	MBZ	**Block4x4 6 (1_1)**
	7:4	MBZ	MBZ	**Block4x4 5 (1_2)**
	3:0	MBZ	MBZ	**Block4x4 4 (1_3)**
2	15:12	MBZ	MBZ	**Block4x4 11 (2_0)**
	11:8	MBZ	MBZ	**Block4x4 10 (2_1)**
	7:4	MBZ	MBZ	**Block4x4 9 (2 2)**
	3:0	MBZ	MBZ	**Block4x4 8 (2_3)**
3	15:12	MBZ	MBZ	**Block4x4 15 (3_0)**
	11:8	MBZ	MBZ	**Block4x4 14 (3_1)**

LumaIntraPredModes [index]		Intra16x16PredMode	Intra8x8PredMode	Intra4x4PredMode
Index	Bit	MbType = [1...24] Transform8x8Flag = 0	MbType = 0 Transform8x8Flag = 1	MbType = 0 Transform8x8Flag = 0
	7:4	MBZ	MBZ	**Block4x4 13 (3_2)**
	3:0	MBZ	MBZ	**Block4x4 12 (3_3)**

Definition of Intra16x16PredMode

Intra16x16PredMode	Description
0	**Intra_16x16_Vertical**
1	**Intra_16x16_Horizontal**
2	**Intra_16x16_DC**
3	**Intra_16x16_Plane**
4 - 15	Reserved

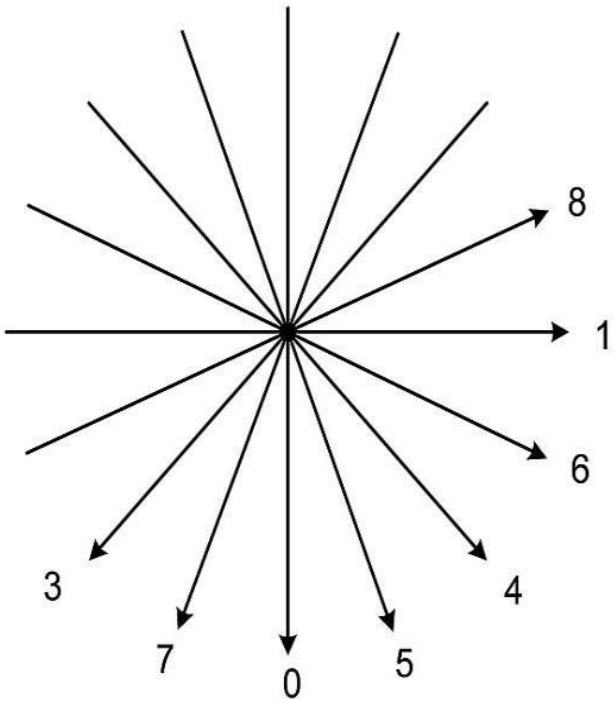
Definition of Intra8x8PredMode

Intra8x8PredMode	Description
0	**Intra_8x8_Vertical**
1	**Intra_8x8_Horizontal**
2	**Intra_8x8_DC**
3	**Intra_8x8_Diagonal_Down_Left**
4	**Intra_8x8_Diagonal_Down_Right**
5	**Intra_8x8_Vertical_Right**
6	**Intra_8x8_Horizontal_Down**
7	**Intra_8x8_Vertical_Left**
8	**Intra_8x8_Horizontal_Up**
9 - 15	Reserved

Definition of Intra4x4PredMode

Intra4x4PredMode	Description
0	**Intra_4x4_Vertical**
1	**Intra_4x4_Horizontal**
2	**Intra_4x4_DC**
3	**Intra_4x4_Diagonal_Down_Left**
4	**Intra_4x4_Diagonal_Down_Right**
5	**Intra_4x4_Vertical_Right**
6	**Intra_4x4_Horizontal_Down**
7	**Intra_4x4_Vertical_Left**
8	**Intra_4x4_Horizontal_Up**
9 - 15	Reserved

Intra_4x4 prediction mode directions



Numbers of Block4x4 in a 16x16 region

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Numbers of Block4x4 in an 8x8 region or numbers of Block8x8 in a 16x16 region

0	1
2	3

Definition of Chroma Intra Prediction Mode

ChromaIntraPredMode (intra_chroma_pred_mode)	Name of intra_chroma_pred_mode
0	Intra_Chroma_DC (prediction mode)
1	Intra_Chroma_Horizontal (prediction mode)
2	Intra_Chroma_Vertical (prediction mode)
3	Intra_Chroma_Plane (prediction mode)

Reference Indices Defined for Each MB Partition Type and Bit Assignment

	frame/field MB/Picture				
MB partitioning	16x16	16x8	8x16	8x8	
RefIdxL0/1[0]	blk0	blk0	blk0	blk0	Bit 7:0
RefIdxL0/1[1]	x	blk1	blk1	blk1	Bit 15:8
RefIdxL0/1[2]	x	x	x	blk2	Bit 23:16
RefIdxL0/1[3]	x	x	x	blk3	Bit 31:24

MB Neighbor Availability in Intra-Prediction Modes (IntraPredAvailFlags)

Current MB is labelled as X. For non-MBAFF mode, 4 neighbors, A, B, C, D, are depicted in the following picture and are defined as the following.

- MB D: top left neighbor of current MB X
- MB C: top right neighbor of current MB X
- MB B: top neighbor of current MB X
- MB A: left neighbor of the current MB X

mbAddrD D (top-left)	**mbAddrB** B (top)	**mbAddrC** C (top-right)
mbAddrA A (left)	**CurrMbAddrX** X	N/A
N/A	N/A	N/A

For MBAFF mode, the current MB is labelled as X0 or X1, 4 neighbor pairs, A0/A1, B0/B1, C0/C1, D0/D1, are depicted in the following picture and are defined as the following.

- MB D0: first MB of top left neighbor MB pair of current MB pair X0/X1

- MB D1: second MB of top left neighbor MB pair of current MB pair X0/X1
- MB C0: first MB of top right neighbor MB pair of current MB pair X0/X1
- MB C1: second MB of top right neighbor MB pair of current MB pair X0/X1
- MB B0: first MB of top neighbor MB pair of current MB pari X0/X1
- MB B1: second MB of top neighbor MB pair of current MB pari X0/X1
- MB A0: first MB of left neighbor MB pair of the current MB pair X0/X1
- MB A1: second MB of left neighbor MB pair of the current MB pair X0/X1

mbAddrD D0	**mbAddrB** B0	**mbAddrC** C0
mbAddrD+1 D1	**mbAddrB+1** B1	**mbAddrC+1** C1
mbAddrA A0	**CurrMbAddrX** X0 or	N/A
mbAddrA+1 A1	**CurrMbAddrX** X1	N/A

For a given macroblock X (or X0/X1), the 6 neighbor availability signals, namely, A, B, C, D, E, F, are defined as the following.

- IntraPredAvailFlagF - F (Single neighbor pixel at the left 8th row (-1,7)
- IntraPredAvailFlagA - A (Left neighbor top half pixel group)
- IntraPredAvailFlagE - E (Left neighbor bottom half pixel group)
- IntraPredAvailFlagB - B (Top neighbor pixel group)
- IntraPredAvailFlagC - C (Top right neighbor pixel group)
- IntraPredAvailFlagD - D (Top left corner neighbor pixel)

The following table depicts the generation of IntraPredAvailFlags[5:0] signals in a condensed form. It should note that for most cases only one input neighbor signal is assigned for each condition. The exception is in the four places for deriving left neighbor A and E where the neighbor is only available if left neighbors (A0 and A1) are both available (A0&A1). Also note that F takes output value very similar to that for A except the two "AND" conditions, where F is assigned to A1 instead of (A0&A1).

Table: Definition of intra-prediction neighbor availability calculation in MBAFF mode

Output =>		D		B		C		A		E		F	
Current X \ Neighbor Y		Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field	Y-Frame	Y-Field
X_0 (Top)	X-Frame	D_1	D_1	B_1	B_1	C_1	C_1	A_0	A_0 & A_1	A_0	A_0 & A_1	A_0	A_1
	X-Field	D_1	D_0	B_1	B_0	C_1	C_0	A_0	A_0	A_1	A_0	A_0	A_0
X_1 (Bottom)	X-Frame	A_0	A_1	X_0	N/A	0	0	A_1	A_0 & A_1	A_1	A_0 & A_1	A_1	A_1
	X-Field	D_1	D_1	B_1	B_1	C_1	C_1	A_0	A_1	A_1	A_1	A_0	A_1

In the table below, Definition of intra-prediction neighbor availability calculation in MBAFF mode, *X-Frame* or *X-Field* indicates the frame/field mode of the current MB; and *Y-Frame* or *Y-Field* indicates the corresponding neighbor MB for the given neighbor location, being upper left (D) or left (A) for example. Therefore, "Y-" takes the selected neighbor MB name as in the output cell in the same column. For example, for output D, if X1 is a frame MB, Y = A, if X1 is a field MB, Y = D.

For non-MBAFF mode, as A0=A1, B0=B1, C0=C1 and D0=D1, the neighbor assignment is degenerated into the following simple table. Here, E is assigned to the same as A and F is forced to 0.

Table: Definition of intra-prediction neighbor availability calculation in non-MBAFF mode

Output =>	D	B	C	A	E	F
X	D0	B0	C0	A0	A0	0

To further explain the neighbor assignment rules in Definition of intra-prediction neighbor availability calculation in MBAFF mode, the following table provides description for each condition. Please note that this table is **informative** as it provides redundant information as in Definition of intra-prediction neighbor availability calculation in MBAFF mode.

Table: Detailed explanation of intra-prediction neighbor availability calculation in MBAFF mode

Current MB	Current MB Field	Neighbor MB Field	Neighbor MB Select (Y=?)	Neighbor Avail Result (OUTPUT)	Description
			D		
X0 (Top)	X-Frame	Y-Frame	D	D1	Top Frame MB uses [-1,-1] = D_31, thus D1 only, regardless D frame or field pair
	X-Frame	Y-Field	D	D1	
	X-Field	Y-Frame	D	D1	Top Field MB uses [-1,-2] = D_30, thus if D is frame pair, takes D1 (D1_14 pixel), and if D is field pair, takes D0 (D0_15 pixel)
	X-Field	Y-Field	D	D0	
X1 (Bottom)	X-Frame	Y-Frame	A	A0	Bottom Frame MB uses [-1,15] = A_15, thus A0 (A0_15 pixel) if A is a frame pair, or A1 (A1_7 pixel), if A is a field pair
	X-Frame	Y-Field	A	A1	

Current MB	Current MB Field	Neighbor MB Field	Neighbor MB Select (Y=?)	Neighbor Avail Result (OUTPUT)	Description
			D		
	X-Field	Y-Frame	D	D1	Bottom Field MB uses [-1,-1] = D_31, thus D1 only, regardless D frame or field pair
	X-Field	Y-Field	D	D1	
			B		
X0 (Top)	X-Frame	Y-Frame	B	B1	Top Frame MB uses [0...15,-1] = B_31, thus B1 only, regardless B frame or field pair
	X-Frame	Y-Field	B	B1	
	X-Field	Y-Frame	B	B1	Top Field MB uses [0...15,-2] = B_30, thus if B is frame pair, takes B1 (B1_14 row), and if B is field pair, takes B0 (B0_15 row)
	X-Field	Y-Field	B	B0	
X1 (Bottom)	X-Frame	Y-Frame	X	X0	Bottom Frame MB uses [0...15,15], thus X0 (X0_15 row)
	X-Frame	Y-Field	X	n/a	**Note:** X0 and X1 must have the same field type, this row is n/a.
	X-Field	Y-Frame	B	B1	Bottom Field MB uses [0...15,-1] = B_31, thus B1 only, regardless B frame or field pair
	X-Field	Y-Field	B	B1	
			C		
X0 (Top)	X-Frame	Y-Frame	C	C1	Top Frame MB uses [16...23,-1] = C_31, thus C1 only, regardless C frame or field pair
	X-Frame	Y-Field	C	C1	
	X-Field	Y-Frame	C	C1	Top Field MB uses [16...23,-2] = C_30, thus if C is frame pair, takes C1 (C1_14 row), and if C is field pair, takes C0 (C0_15 row)
	X-Field	Y-Field	C	C0	
X1 (Bottom)	X-Frame	Y-Frame	n/a	0	Bottom Frame MB doesn't have left-top neighbor by definition, thus forced to 0
	X-Frame	Y-Field	n/a	0	
	X-Field	Y-Frame	C	C1	Bottom Field MB uses [16...23,-1] = C_31, thus C1 only, regardless C frame or field pair
	X-Field	Y-Field	C	C1	
			A		
X0 (Top)	X-Frame	Y-Frame	A	A0	First Half of Top Frame MB uses [-1,0...7], thus A0 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A0	First Half of Top Field MB uses [-1,0..2..4..14], thus take A0 (if A is frame pair, takes A0 even lines, and if A is field pair, takes A0 first half)
	X-Field	Y-Field	A	A0	
X1 (Bottom)	X-Frame	Y-Frame	A	A1	First Half of Bottom Frame MB uses [-1,16...23], thus A1 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A0	First Half of Bottom Field MB uses [-

Current MB	Current MB Field	Neighbor MB Field	Neighbor MB Select (Y=?)	Neighbor Avail Result (OUTPUT)	Description
			D		
	X-Field	Y-Field	A	A1	1,1..3..15], thus take A0 (if A is frame pair, takes A0 odd lines, and if A is field pair, takes A1 first half)
			E		
X0 (Top)	X-Frame	Y-Frame	A	A0	Second Half of Top Frame MB uses [-1,8...15], thus A0 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A1	Second Half of Top Field MB uses [-1,16..18..30], thus take A1 (if A is frame pair, takes A1 even lines, and if A is field pair, takes A0 second half)
	X-Field	Y-Field	A	A0	
X1 (Bottom)	X-Frame	Y-Frame	A	A1	Second Half of Bottom Frame MB uses [-1,24...31], thus A1 if A is a frame pair; but is only avail if both A0 and A1 are avail if A is a field pair due to the mix
	X-Frame	Y-Field	A	A0&A1	
	X-Field	Y-Frame	A	A1	Second Half of Bottom Field MB uses [-1,17..19..31], thus takes A1 (if A is frame pair, takes A1 odd lines, and if A is field pair, takes A1 second half)
	X-Field	Y-Field	A	A1	
			F		
X0 (Top)	X-Frame	Y-Frame	A	A0	Top Frame MB uses [-1,7] = A_7 (odd location), thus A0 if A is frame pair and A1 if field pair
	X-Frame	Y-Field	A	A1	
	X-Field	Y-Frame	A	A0	Top Field MB uses [-1,14] = A_14 (even location), thus A0 regardless A frame or field pair
	X-Field	Y-Field	A	A0	
X1 (Bottom)	X-Frame	Y-Frame	A	A1	Bottom Frame MB uses [-1,23] = A_23 (odd location), thus A1 regardless A frame or field pair
	X-Frame	Y-Field	A	A1	
	X-Field	Y-Frame	A	A0	Bottom Field MB uses [-1,15] = A_15 (odd location), thus A0 if A is frame pair and A1 if A is field pair
	X-Field	Y-Field	A	A1	

Macroblock Type for Intra Cases

MbType follows two different tables according to whether the macroblock is an inter or intra macroblock according to IntraMbFlag.

For an intra macroblock, MbType, as defined in MbType definition for Intra Macroblock, carries redundant information as IntraMbMode. The notation I_16x16_x_y_z used in the table, 'x' is

Intra16x16LumaPredMode, 'y' is ChromaCbpInd, and 'z' is LumaCbpInd, as defined in Sub field definition used by MbType for a macroblock with Intra16x16 prediction.

MbType definition for Intra Macroblock

Macroblock Type	MbType
I_4x4	0
I_8x8	0
I_16x16_0_0_0	1
I_16x16_1_0_0	2
I_16x16_2_0_0	3
I_16x16_3_0_0	4
I_16x16_0_1_0	5
I_16x16_1_1_0	6
I_16x16_2_1_0	7
I_16x16_3_1_0	8
I_16x16_0_2_0	9
I_16x16_1_2_0	Ah
I_16x16_2_2_0	Bh
I_16x16_3_2_0	Ch
I_16x16_0_0_1	Dh
I_16x16_1_0_1	Eh
I_16x16_2_0_1	Fh
I_16x16_3_0_1	10h
I_16x16_0_1_1	11h
I_16x16_1_1_1	12h
I_16x16_2_1_1	13h

Macroblock Type	MbType
I_16x16_3_1_1	14h
I_16x16_0_2_1	15h
I_16x16_1_2_1	16h
I_16x16_2_2_1	17h
I_16x16_3_2_1	18h
I_PCM	19h (used by HW)

Note: MbType here is identical as specified in DXVA 2.0.

For Intra_16x16 modes, the 5 bits of value (MbType - 1) have the following meanings.

Sub field definition used by MbType for a macroblock with Intra16x16 prediction

Bits	Description
4	**LumaCbpInd** - Luma Coded Block Pattern Indicator 0 means none of the luma blocks are coded. 1 means that at least one luma block is coded. 0 = SUBMODE_I16_L_0 1 = SUBMODE_I16_L_NZ In VME output, this field is forced to be 1 before adding 1 in Intra_16x16 mode.
3:2	**ChromaCbpInd** - Chroma Coded Block Pattern Indicator 00 means none of chroma blocks are coded. 01 means that only the chroma DC block is coded, but all AC blocks are not coded. 10 means that at least one AC chroma block is coded. 00 = SUBMODE_I16_C_0 01 = SUBMODE_I16_C_DC 10 = SUBMODE_I16_C_NZ 11 = Reserved In VME output, this field is forced to be 10 before adding 1 in Intra_16x16 mode. *Programming Note: Adding 1 to MbType by VME hardware may have carry in to this field. But as '11' is reserved, the carry-in doesn't propagate into bit 4 or higher. This allows software to update MbType, if desired, using the redundant LumaIntraPredModes information.*
1:0	**Intra16x16PredMode** - Intra16x16 Prediction Mode

Bits	Description
	These two bits carries redundant (identical) information as that in LumaIntraPredModes[0][0]. 0 = SUBMODE_I16_VER 1 = SUBMODE_I16_HOR 2 = SUBMODE_I16_DC 3 = SUBMODE_I16_PLANE

IntraMbMode definition

IntraMbMode [1:0]	Description	Supported by VME?	Used by PAK?
0	**INTRA_16x16 (redundant with MbType)**	Yes	Ignored
1	**INTRA_8x8**	Yes	Yes
2	**INTRA_4x4**	Yes	Yes
3	**IPCM (redundant with MbType)**	No	Ignored

As an alternative representation, MbType is logically the same as the following, except the I_PCM and I_NxN (i.e. I_4x4 and I_8x8) cases:

- 24 types of 16x16 intra modes: **A**+**B**+**C**+**D**: (1h - 18h)

  MBTYPE_INTRA_16x16 1h A

  - 4 Intra16x16 modes:

    SUBMODE_I16_VER 0 B
    SUBMODE_I16_HOR 1 B
    SUBMODE_I16_DC 2 B
    SUBMODE_I16_PLN 3 B

  - 3 Chroma Cbp indices:

    SUBMODE_I16_C_0 0 C
    SUBMODE_I16_C_DC 4 C
    SUBMODE_I16_C_NZ 8 C

  - 2 Luma Cbp indices:

    SUBMODE_I16_L_0 0 D
    SUBMODE_I16_L_NZ Ch D

Macroblock Type for Inter Cases

Sub-Macroblock Prediction Mode, SubMbPredMode, indicates the prediction mode for the sub-partitions. Prediction mode specifies prediction direction being forward (from L0), backward (from L1) or bi-directional (from both L0 and L1). Its meaning depends on InterMbMode. Definition of SubMbPredMode[i] provides the definition of the field.

- If InterMbMode is INTER16x16, only SubMbPredMode[0] is valid, it describes the prediction mode of the 16x16 macroblock. The other entries are set to zero by hardware.
    - For AVC, SubMbPredMode[0] contains redundant information as encoded in MbType parameter.
    - *Note: SubMbPredMode[1]-[3] are intentionally set to zero to allow a simple LUT to derive MbType as described later.*
- If InterMbMode is INTER16x8, and INTER8x16, only the first two entries SubMbPredMode[0] and SubMbPredMode[1] are valid, describing the sub-macroblock prediction mode.
    - For AVC, SubMbPredMode[0]/[1] contains redundant information as encoded in MbType parameter.
    - *Note: SubMbPredMode[2]-[3] are intentionally set to zero to allow a simple LUT to derive MbType as described later.*
- If InterMbMode is INTER8x8, each entry of SubMbPredMode describes the prediction mode of the sub-partition of an 8x8 sub-macroblock.
    - For AVC, SubMbPredMode can be derived from sub_mb_type field for BP_8x8 macroblocks as defined in AVC spec.
    - *Note on Direct Sub-macroblock Prediction Mode: Direct prediction is not conveyed through SubMbPredMode, instead, it is carried through Direct8x8Pattern.*

InterMbMode definition

MbSkipFlag	InterMbMode	Description
0	0	**INTER16x16**
0	1	**INTER16x8**
0	2	**INTER8x16**
0	3	**INTER8x8**
1	0	**PSKIP/BSKIP16x16***
1	3	**BSKIP**
1	1, 2	Reserved
Used by PAK	Ignored by PAK	

* BSKIP16x16 is an optional non-standard but equivalent optimization.

Definition of SubMbPredMode based on InterMbMode

SubMbPredMode	INTER16x16	INTER16x8	INTER8x16	INTER8x8
Bit	MbType = [1...3]	MbType = [16h]	MbType = [4...15h]	MbType = [16h]
7:6	MBZ	MBZ	MBZ	**Block8x8 3**
5:4	MBZ	MBZ	MBZ	**Block8x8 2**
3:2	MBZ	**Block16x8 1**	**Block8x16 1**	**Block8x8 1**
1:0	**Block16x16**	**Block16x8 0**	**Block8x16 0**	**Block8x8 0**
	Ignored by PAK	**Ignored by PAK**	**Ignored by PAK**	**Used by PAK**

Definition of SubMbPredMode[i]

SubMbPredMode	Description	InterMbMode	VME Output	MvCountPred	Notes
0	Pred_L0	All	Yes	1	P or B Slice
1	Pred_L1	All	Yes	1	B Slice Only
2	BiPred	All	Yes	2	B Slice Only
3	Reserved	Reserved	Reserved	Reserved	Reserved

Sub-Macroblock Shape, SubMbShape[i], for i = 0...3, describes the shape of the sub partitions of the 8x8 sub-macroblock of a BP_8x8 macroblock. This field is only valid if InterMBMode is INTER8x8. They are defined in Definition of SubMbShape for an 8x8 region of a BP_8x8 macroblock (including BSKIP, BDIRECT). The parameters can be derived from *sub_mb_type* field as defined in AVC spec.

Note: These fields must be correctly set even for **Direct** or **Skip** 8x8 cases, the individual B_Direct_8x8 block is flagged by the **Direct8x8Pattern** variable.

Definition of SubMbShape for an 8x8 region of a BP_8x8 macroblock (including BSKIP, BDIRECT)

	Description			
SubMbShape	NumSubMbPart	SubMbPartWidth	SubMbPartHeight	MvCountShape
0	1	8	8	1
1	2	8	4	2
2	2	4	8	2
3	4	4	4	4

For an inter macroblock, MbType, carries redundant information as InterMbMode and SubMbPredMode. MbType definition for Inter Macroblock (and MbSkipflag = 0) provides the typical inter macroblock types and Additional MbType definition with Direct/Skip for Inter Macroblock provides that with skip and direct modes. The definition of MbType for both P slice and B slice is the same and is equivalent to that

for mb_type of a B slice in the AVC spec. As direct mode is indicated using a separate field Direct8x8Pattern, 0 is reserved for MbType.

Here, MVCount is the number of motion vectors actually encoded in the bitstream. It is informative. For a BP_8x8 or equivalent Skip/Direct macroblock, MVCount is the sum of the following term for the four 8x8 sub macroblock (with i = 0...3):

MvCountShape[i] * MvCountPred[i] * MvCountDirect[i]

where MvCountShape[i] is block count for sub macroblock [i], MvCountPred[i] is the motion vector count for each block of sub macroblock[i], and MvCountDirect[i] is the multipler for direct mode for B Slice, indicating whether motion vectors are coded or not. It must be set to 1 for P slice. For B Slice, MvCountDirect[i] = !Direct8x8Pattern[i], which is 0 for a sub macroblock coded as direct mode and 1 otherwise.

In the tables, "DC" stands for "Don't Care" as PAK hardware ignores these fields.

MbType definition for Inter Macroblock (and MbSkipflag = 0)

Macroblock Type	MbType	MbSkipFlag	Direct8x8Pattern	SubMbShape	SubMbPredMode	MVCount
Reserved	0	-	-	-	-	-
BP_L0_16x16	1	0	0	DC	DC	1
B_L1_16x16	2	0	0	DC	DC	1
B_Bi_16x16	3	0	0	DC	DC	2
BP_L0_L0_16x8	4	0	0	DC	DC	2
BP_L0_L0_8x16	5	0	0	DC	DC	2
B_L1_L1_16x8	6	0	0	DC	DC	2
B_L1_L1_8x16	7	0	0	DC	DC	2
B_L0_L1_16x8	8	0	0	DC	DC	2
B_L0_L1_8x16	9	0	0	DC	DC	2
B_L1_L0_16x8	0Ah	0	0	DC	DC	2
B_L1_L0_8x16	0Bh	0	0	DC	DC	2
B_L0_Bi_16x8	0Ch	0	0	DC	DC	3
B_L0_Bi_8x16	0Dh	0	0	DC	DC	3
B_L1_Bi_16x8	0Eh	0	0	DC	DC	3
B_L1_Bi_8x16	0Fh	0	0	DC	DC	3
B_Bi_L0_16x8	10h	0	0	DC	DC	3
B_Bi_L0_8x16	11h	0	0	DC	DC	3
B_Bi_L1_16x8	12h	0	0	DC	DC	3
B_Bi_L1_8x16	13h	0	0	DC	DC	3
B_Bi_Bi_16x8	14h	0	0	DC	DC	4
B_Bi_Bi_8x16	15h	0	0	DC	DC	4
BP_8x8	16h	0	!= Fh	vary	vary	Sum

Macroblock Type	MbType	MbSkipFlag	Direct8x8Pattern	SubMbShape	SubMbPredMode	MVCount
Reserved	17h-1Fh	-	-	-	-	-

Additional MbType definition with Direct/Skip for Inter Macroblock

Macroblock Type	MbType	Xfrm 8x8	MbSkip Flag	Direct8x8 Pattern	SubMb Shape	SubMb PredMode	MvCount	Notes
P_Skip_16x16	1	-	1	DC	DC	DC	0	Skipped macroblock. Motion compensation like P_L0_16x16
B_Skip_16x16_4MVPair	16h	vary	1	Fh	0	vary	0	Skipped macroblock. Motion compensation like B_8x8 with 8x8 subblocks, when **direct_8x8_inference_flag** is set to 1
B_Skip_16x16_16MVPair	16h	0	1	Fh	FFh	vary	0	Skipped macroblock. Motion compensation like B_8x8 with 4x4 subblocks, when **direct_8x8_inference_flag** is set to 0
B_Direct_16x16_4MVPair	16h	vary	0	Fh	0	vary	0	MbType coded as B_Direct_16x16. Motion compensation like B_8x8 with 8x8 subblocks, when **direct_8x8_inference_flag** is set to 1
B_Direct_16x16_16MVPair	16h	0	0	Fh	FFh	vary	0	MbType coded as B_Direct_16x16. Motion compensation like B_8x8 with 4x4 subblocks, when **direct_8x8_inference_flag** is set to 0

People might notice that B_DIRECT_16x16 and B_SKIP are mapped on BP_8x8 for AVC decoding interface in IT mode as the motion compensation operation for both modes are the same as BP_8x8. According to AVC Spec, motion vectors for B_DIRECT_16x16 and B_SKIP are derived from temporally co-located macroblock on an 8x8 sub macroblock basis if direct_8x8_inference_flag is set to 1 or on a 4x4 block basis if it is set to 0. For each sub macroblock or block, SubMbPredMode is derived, thus can any of the valid numbers. Motion vectors may also be different. In spatial direct mode, the motion vectors are subject to spatial neighbor macroblocks as well as co-located macroblock. The spatial prediction is based on the neighbor macroblocks, so the same spatial predicted motion vector applies to all sub

macroblocks or blocks. However, under certain conditions, temporal predictor may replace (colZeroFlag) the spatial predictor for a given sub macroblock or block. Thus the motion vectors may differ.

In MbType definition for Inter Macroblock (and MbSkipflag = 0), the macroblock type names for major partitions nicely follow forms *BP_MbPredMode_MbShape* (like BP_L0_16x16) and *B_MbPredMode0_MbPredMode1_MbShape* (like B_L0_Bi_16x8). For minor partitions it is fixed as *BP_MbShape* as BP_8x8.

However, in Additional MbType definition with Direct/Skip for Inter Macroblock the macroblock types for Skip and Direct modes does not follow the same rule. The third field in P_Skip_16x16 or B_Direct_16x16_x indicates that "Skip" or "Direct" applies to the entire 16x16 macroblock, even though MbShape is 8x8 as that in BP_8x8. In order to distinguish the SubMbShape being 8x8 or 4x4 for B_Skip and B_Direct, the fourth field is added. 4MVP**air** indicates upto 4 MV pairs are presented with SubMbShape equals to 0; and 16MVP**air** indicates up to 16 MV pairs are presented with SubMbShape equals to FFh.Also note that P_8x8ref0 is not specified in PAK input interface, it is up to hardware to detect and choose its packing format based on number of reference indices and reference index for the given macroblock.

Macroblock Type Conversion Rules

For improved coding efficiency the PAK hardware has the capability to convert macroblock types to use more efficiency coding modes such as DIRECT and SKIP. For an inter macroblock or a sub macroblock coded as DIRECT, no motion vector is needed in the bitstream for the macroblock or sub macroblock. If a macroblock is coded as SKIP, it only consumes one SKIP bit (no motion vector, no coefficients are coded). And infomaton about the macroblock is 'inferred' according to the rules stated in the AVC Spec.

As the input to PAK, the following signals can convey the information regarding DIRECT and SKIP:

- MbSkipFlag
- Direct8x8Pattern
- CodecBlockPattern (CbpY, CbpCb, CbpCr)

Such conversion can be enabled or disabled through the SLICE_STATE fields DirectConvDisable and SkipConvDisable as well as the in line command field MbSkipConvDisable.

A P slice doesn't support direct mode, it only supports P_Skip, which is equivalent to a 16_16_L0 prediction. Other prediction types cannot be converted to P_Skip. The following table describes the macroblock type conversion rules for a P slice. Here CBP = CbpY/CbpCb/CbpCr are the final computed results after quantization by the hardware. Note that hardware honors the input CbpY/CbpCb/CbpCr fields - if the value corresponding to a block is set to zero, the resulting CBP is also zero. The output mb_skip_flag and mb_type are the symbols coded in the bitstream as defined in the AVC spec. *DC* stands for *Don't care*, *T* for *True*.

Note that the internal condition of MV==MVP is subject to the precise rules stated in the AVC Spec as quoted below. Note that there are exceptions for P_Skip from the normal motion vector prediction rules.

Derivation process for luma motion vectors for skipped macroblocks in P and SP slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0.

The reference index refIdxL0 for a skipped macroblock is derived as follows.

refIdxL0 = 0. (8-168)

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

- The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.

- The variable mvL0 is specified as follows.

- If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0.

- mbAddrA is not available

- mbAddrB is not available

- refIdxL0A is equal to 0 and both components of mvL0A are equal to 0

- refIdxL0B is equal to 0 and both components of mvL0B are equal to 0

- Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.

NOTE - The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

Macroblock type conversion rule for an inter macroblock in a P slice

Input		Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
P_Skip_16x16	DC	DC	DC	1	1	-	Forced to P_Skip; Hardware will force CBP to zero and also ignore SkipConvDisable control. Hardware doesn't check for MV==MVP error condition
P_Skip_16x16	DC	DC	DC	0	0	0	Reverse convert to P_L0_16x16; Hardware will force CBP to zero but reversely convert MbType as P_L0_16x16 once it determines that Skip is not allowed.

Input		Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
BP _16x16_L0	0	0	T	1	1	-	Converted to P_Skip. Even input doesn't provide skip hint, hardware can performance the optimization by detecting CBP and MV==MVP condition.
BP _16x16_L0	0	0	T	0	0	0	Reverse back to P_L0_16x16; Hardware will reverse back to P_L0_16x16 even Skip conditions are met once it determines that Skip is not allowed.
BP _16x16_L0	1	0	T	T	0	0	Still coded as P_L0_16x16 = 0.

A B slice supports both direct and skip modes. The following table describes the macroblock type conversion rules for a B slice. Hardware does not verify MV==MVP condition for a Skip/Direct macroblock in a B Slice as no DMV is performed by hardware.

Macroblock type conversion rule for an inter macroblock in a B slice

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
B_Skip_8x8 B_Skip_4x4	DC	DC	DC	n/a	1	1	-	Forced to B_Skip; Hardware will force CBP to zero and also ignore SkipConvDisable control.

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
B_Skip_8x8 B_Skip_4x4	DC	DC	DC	n/a	0	0	0	REVERSE convert to B_Direct_16x16; Hardware will force CBP to zero and also reverse convert to B_Direct_16x16 when it discovers Skip is not allowed.
B_Direct_16x16_4MVPair/ 16MVPair	0	0	0	n/a	1	1	-	Converted to B_Skip. Hardware first converts to B_Direct_16x16 and then further to B_Skip if CBP = 0.
B_Direct_16x16_4MVPair/ 16MVPair	0	0	0	n/a	0	0	0	Converted to B_Direct_16x16. Hardware first converts to B_Direct_16x16 and stop there as it discovers Skip is not allowed even CBP=0.
B_Direct_16x16_4MVPair/	1	0	0	n/a	DC	0	0	Converted

Input			Internal			Output		Notes
Macroblock Type	SkipConvDisable || SkipConvDisable	DirectConvDisable	CBP	MV == MVP	MbAffSkipAllowed	mb_skip_flag	mb_type	
16MVPair								to B_Direct_16x16. Hardware converts to B_Direct_16x16 and stops there even though CBP = 0 as input disallows Skip conversion.
B_Direct_16x16_4MVPair/ 16MVPair	DC	0	NZ	n/a	DC	0	0	Converted to B_Direct_16x16. Hardware converts to B_Direct_16x16 and stops there because CBP != 0.
B_Direct_16x16_4MVPair/ 16MVPair	DC	1	DC	n/a	DC	0	16h	Stay as B_8x8. Hardware stays at B_8x8 and codes each sub macroblocks even all are direct.

The internal signal MbAffSkipAllowed is added to deal with a restriction on the frame/field flag (MbFieldFlag) which is unique to MBAFF. MbAffSkipAllowed is always set to 1 in non-MBAFF modes. In MBAFF mode, a macroblock pair may be both skipped only if its MbFieldFlag is the same as its available neighbor macroblock pair A or B if A or B is available (in that order), or is not 0 if A/B are both not available. Otherwise, one of the macroblocks in the pair must be coded.

To reduce the burden on software, PAK hardware handles the above restriction correctly. For the first MB in a pair, MbAffSkipAllowed is always set to 1. Therefore, hardware allows converting the first MB to Skip if skip conversion is enabled. For the second MB in a pair, hardware sets MbAffSkipAllowed to 0 if the following is true:

- The current MB Pair has different MbFieldFlag than its available neighbor A or B if A or B is available, or is not 0 if A/B are both not available
- And the first MB is coded as a SKIP (could be forced or converted)

Otherwise, it sets MbAffSkipAllowed to 1. As MbAffSkipAllowed is to 0 for the above condition, hardware will disallow Skip mode for the second MB. If the input signal forces it to Skip, hardware performs reverse-convertion to code it as P_L0_16x16 or B_Direct_16x16 with CBP = 0 for a macroblock in a P or B Slice. This means that hardware is able to correct the programming mistake by software. If the macroblock is not forced to skip, hardware simply disallows Skip conversion.

Software still has an option to disallow Skip Conversion on a per-MB basis using the MbSkipConvDisable control field in the inline command.

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

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**

	DWord	Bit	MvSize	
			8	32
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**
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**

	DWord	Bit	MvSize	
			8	32
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**
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 Motion Vector duplication by sub-macroblocks for a 16x16 macroblock, whereas the 8x8 column is for 4x(8x8) partition without minor shape and 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).

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**

DWord	Bit	16x16	16x8	8x16	8x8
	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.

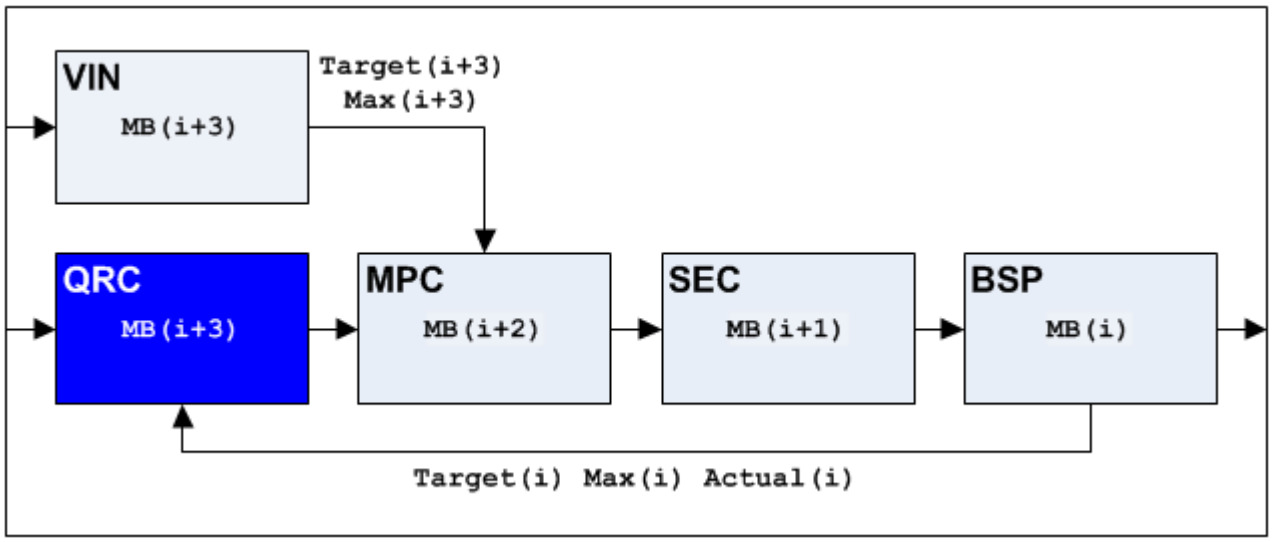
MbSkipFlag	MbType	Description	Mv Quantity	MvSize	(Minimal MvSize)
1	1	P_Skip_16x16	0	8	1
0	1	BP_L0_16x16	1	8	1
0	2	B_L1_16x16	1	8	1
0	3	B_Bi_16x16	2	8	2
0	4	BP_L0_L0_16x8	2	8	4
0	5	BP_L0_L0_8x16	2	8	4
0	6	B_L1_L1_16x8	2	8	8
0	7	B_L1_L1_8x16	2	8	8
0	8	B_L0_L1_16x8	2	8	8
0	9	B_L0_L1_8x16	2	8	8
0	0Ah	B_L1_L0_16x8	2	8	8
0	0Bh	B_L1_L0_8x16	2	8	8
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	^34	8 or 32	8 or 32

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

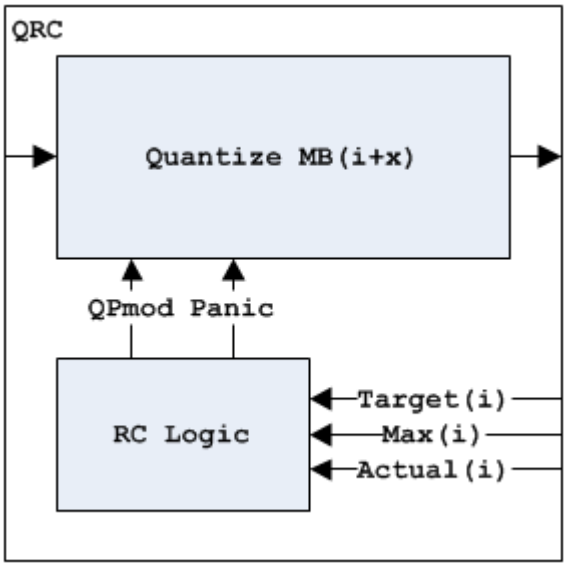
Direct8x8Pattern	SubMb Shape	SubMb PredMode	Description	Mv Quantity	Mv Size	(Min MvSize)
OR	OR	OR		ADD	ADD	ADD
1	0	0	P_Skip_8x8 B_Direct_L0_8x8 (B-Skip_ L0_8x8)	0	2	1
1	0	1	B_Direct_L1_8x8 (B-Skip_ L1_8x8)	0	2	1
1	0	2	B_Direct_Bi_8x8 (B-Skip_ Bi_8x8)	0	2	2
1	3	0	P_Skip_4x4 B_Direct_L0_4x4 (B-Skip_ L0_4x4)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

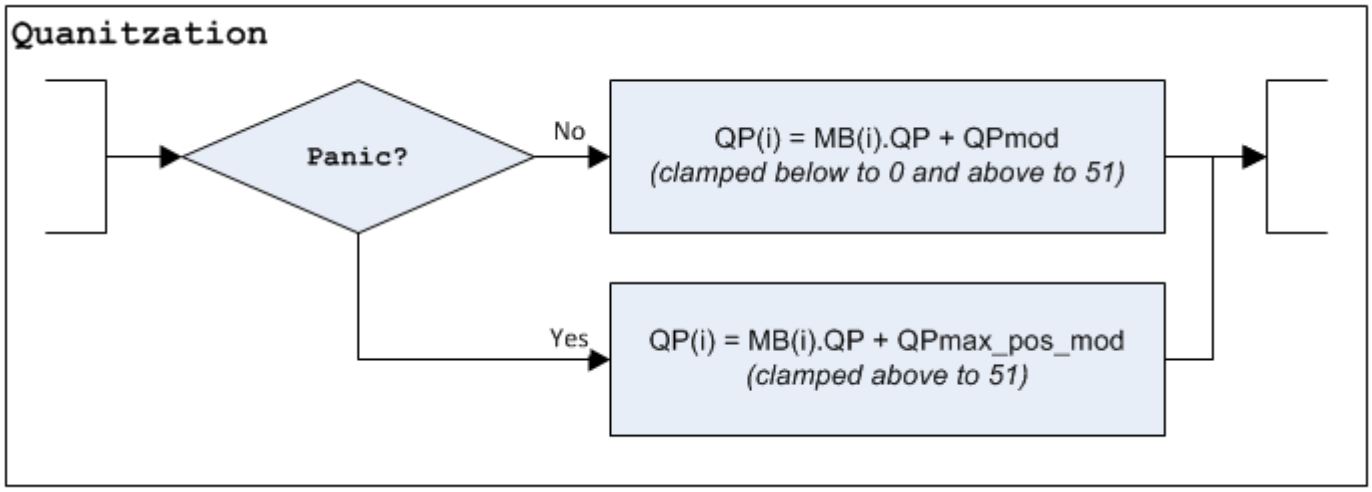
The QRC (Quantization Rate Control) unit receives data from BSP (Bit Serial Packer) and VIN (Video In) and generates adjustments to QP values across macroblocks.



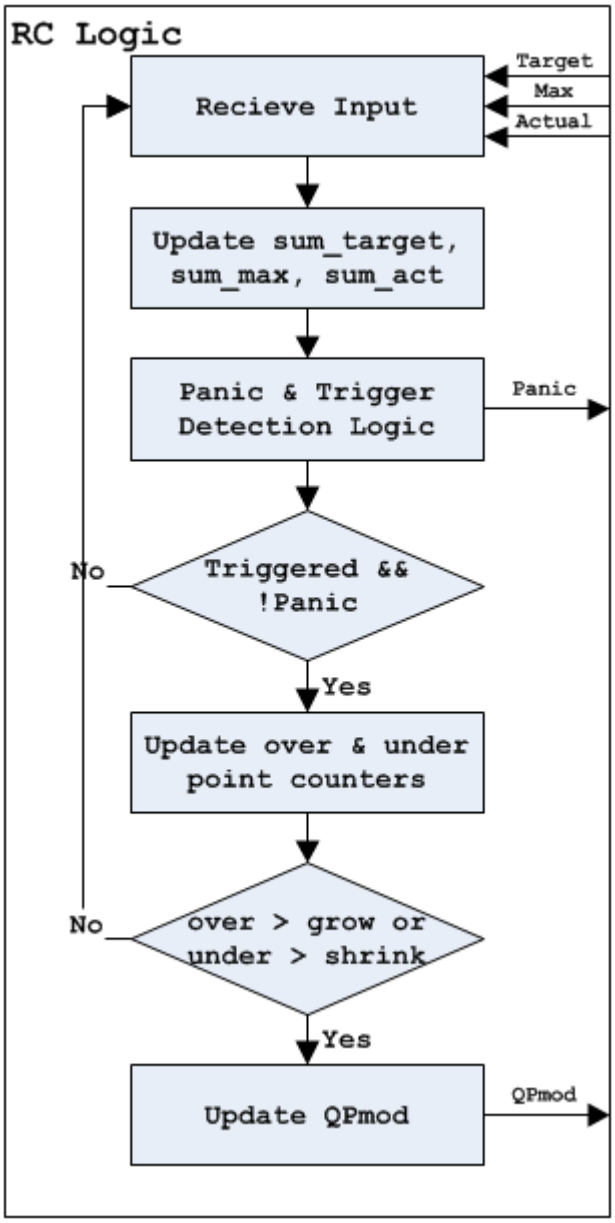
QRC can be logically partitioned into two units as shown below.



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



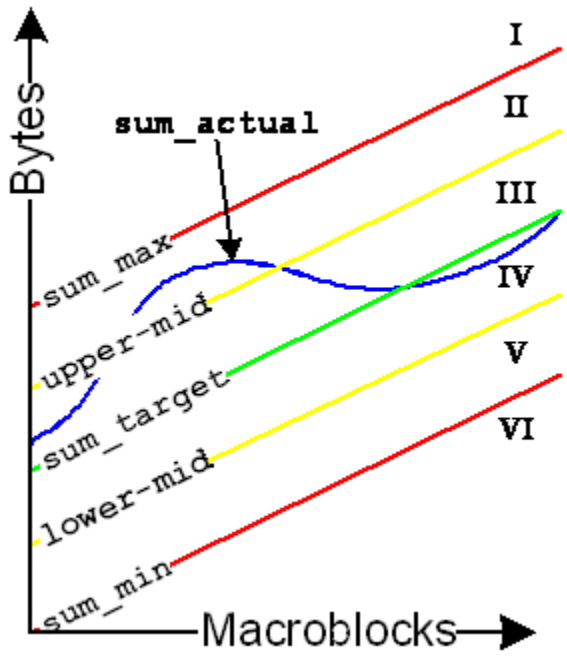
The signals QPmod and panic are generated by the RC logic based on data feedback from BSP. A flowchart of the RC logic is given below.



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

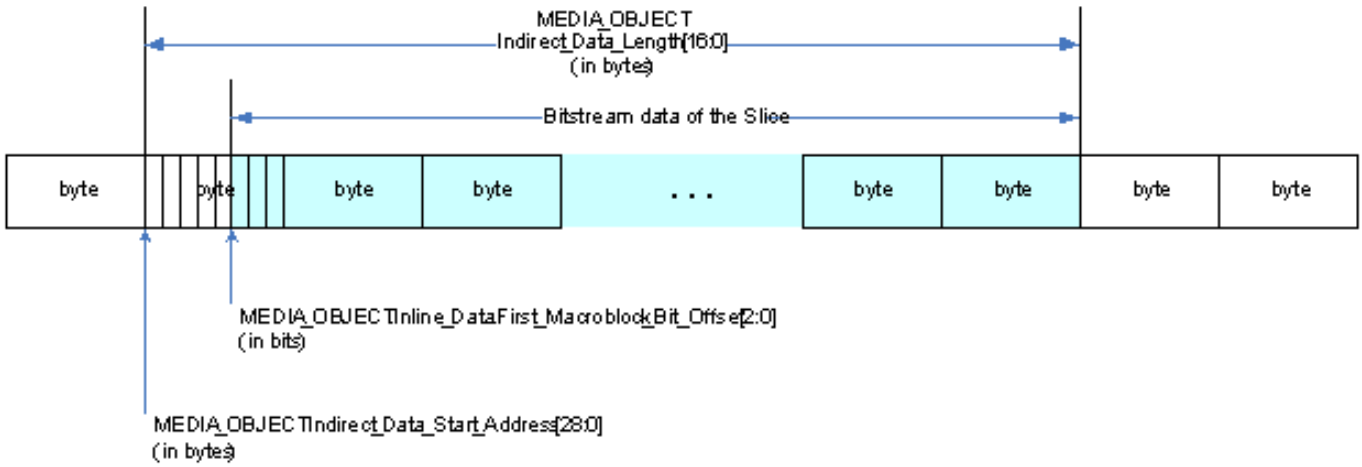
MFD_MPE2_BSD_OJBECT Inline Data Description

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. The image below, Indirect data buffer for a slice 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. This field is reserved (MBZ) when the **IntraMbFlag** = 1. The valid encodings are: **011 = Unpacked: Two motion vector pairs** Others are reserved. *(The following encodings are intended for other formats:* *001 = 1MV: one 16x16 motion vector*

DWord	Bit	Description
		010 = 2MV: One 16x16 motion vector pair *011 = 4MV: Four 8x8 motion vectors, or Two 16x8 motion vector pairs* *100 = 8MV: Four 8x8 motion vector pairs* *101 = 16MV: 16 4x4 motion vectors* *110 = 32MV: 16 4x4 motion vector pairs* *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** Used to indicate transformation type for MPEG-2. 0 = Frame DCT transformation 1 = Field DCT transformation
	14	**FieldMbFlag** For MPEG-2, this flag is set to 1 if either the picture is in field type or the MB is INTER of field type, i.e. split into two 16x8 field blocks.
	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) 1: INTRA (intra macroblock)
	12:8	**MbType** 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

DWord	Bit	Description
	2	**SkipMbFlag** 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. 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. This field can only be set to 1 for certain values of MbType. See details later. This field is only valid for an inter macroblock. Hardware ignores this field for an intra macroblock. 0 = not a skipped macroblock 1 = is coded as a skipped macroblock **Note:** When this flag is set to 1, the correct MVs are assumed for HW decoder to generate decoded reconstruction frame.
	1:0	**InterMbMode** This field is provided to carry redundant information as that encoded in MbType. 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. 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. Format = U16 in unit of macroblock.
3	31:24	**MaxSizeInWord** PAK should not exceed this budget accumulatively, otherwise it will trickle the PANIC mode.
	23:16	**TargetSizeInWord** 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. Format = 6-bit mask (or 8-bit, & 12-bit, for 422 and 444). Bit 11: Y0Bit 10: Y1Bit 9: Y2Bit 8: Y3

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>Bit 7: Cb4Bit 6: Cr5Bits 0-5: MBZ</td></tr>
<tr><td rowspan="9">4</td><td>31</td><td>LastMbInSlice - the last MB in a slice.</td></tr>
<tr><td>30</td><td>FirstMbInSlice - the first slice in a slice, it requires slice header insertion.</td></tr>
<tr><td>29:28</td><td>MBZ</td></tr>
<tr><td>27</td><td>EnableCoeffClamp
1 = the magnitude of coefficients of the current MB will be clamped based on the clamping matrix after quantization
0 = no clamping</td></tr>
<tr><td>26</td><td>LastMbInSG
1 - the current MB is the last MB in the current slice group.</td></tr>
<tr><td>25</td><td>MbSkipConvDisable
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.
0 - Enable skip type conversion for the current macroblock
1 - Disable skip type conversion for the current macroblock</td></tr>
<tr><td>24</td><td>FirstMbInSG
1 - the current MB is the last MB in the current slice group.</td></tr>
<tr><td>23:20</td><td>MBZ</td></tr>
<tr><td>19:16</td><td>MvFieldSelect - Motion Vertical Field Select. A bit-wise representation of a long [2][2] array as defined in Section 6.3.17.2 of the ISO/IEC 13818-2 (see also Section 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.
0 = The prediction is taken from the top reference field.
1 = The prediction is taken from the bottom reference field.</td></tr>
</table>

DWord	Bit	Description
	15:5	**MBZ Reserved**
	4:0	**QpScaleCode**
5	31:16	**MV[0][0].y** - the y coordinate of the first forward MV if Mv[0][0] n/a: if Mv[1][0] available, it MUST be set to the same value as Mv[1][0]. else it MUST be set to the value 0
	15:0	**MV[0][0].x** - the x coordinate of the first forward MV if Mv[0][0] n/a: if Mv[1][0] available, it MUST be set to the same value as Mv[1][0]. else it MUST be set to the value 0
6	31:0	**MV[1][0]** - the first backward MV if Mv[1][0] n/a: it MUST be set to the same value as Mv[0][0]
7	31:0	**MV[0][1]** - the second forward MV if Mv[0][1] n/a: if Mv[1][1] available, it MUST be set to the same value as Mv[1][1]. else it MUST be set to the same value as Mv[0][0]
8	31:0	**MV[1][1]** - the second backward MV if Mv[1][1] n/a: it MUST be set to the same value as Mv[1][0]

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_bit0	motion_type_bit1	motion_forward	motion_backward
IPB	Intra	1	0	1Ah	0/1	-	1	-	-	-	-
P **B** B	Skip	0	1	01h 02h 03h	0/1	0	0	-	-	1 0 1	0 1 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
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 02h 03h	0	0	0	0	1	1 0 1	0 1 1
B Fram	Field type	0	0	04h 06h	1	1	0	1	0	1 0	0 1

		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_bit0	motion_type_bit1	motion_forward	motion_backward
e				14h						1	1
B Field	One 16x16	0	0	01h 02h 03h	1	0	0	1	0	1 0 1	0 1 1
B Field	Two 16x8	0	0	04h 06h 14h	1	1	0	0	1	1 0 1	0 1 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 seem 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 BSpec

Location		HW	
		BSPEC	
BYTE	Dword	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**	
0.4-0.5	0[4-5]	**IntraMbMode**	IntraMacroblock
0.6	0[6]	**mbz**	
0.7	0[7]	**FieldMbPolarity**	derived
1		**MbType**	
1.0-1.4	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
2.4-2.6	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]
8.0-8.5	2[0-5]	**mbz**	
8.6-8.7	2[6-7]	**CbpAcUV**	PAK control
9	2[8-11]	**CbpAcY**	PAK control
	2[12-15]	**mbz**	
10	2[16-23]	**TargetedSzInWord**	

Location		HW	
		BSPEC	
BYTE	Dword	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 Bspec to DXVA

Location	BSPEC	
BYTE	**DXVA**	**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**
2.6-2.7	MBscanMethod	**= MbScanMethod**
3.0-3.1	MotionType	see (B)
3.2	HostResDiff	**= ResidDataFlag**
3.3	Reserved	
3.4-3.7	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 Bspec 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);
        }
    }
```

Video Codec VC-1

This section describes support for the open video compression standard VC-1, which is the common name for SMPTE 421M approved on April 3, 2006.

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 position 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.

JPEG and MJPEG

JPEG Decoder Commands

Following are JPEG Decoder Commands:

MFD_JPEG_BSD_OBJECT

MFX_JPEG_PIC_STATE command is used for both encoding and decoding. Note the duplicate bits and the "Exists If" rows that specify what the bits represent for Encoder and Decoder.

MFX_JPEG_PIC_STATE

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).

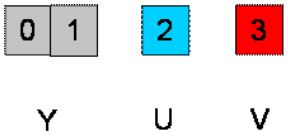
Chroma type is determined by the values of horizontal and vertical sampling factors of the components (*Hi* and Vi where *i* is a component id) in the Frame header as shown in the following table.

	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

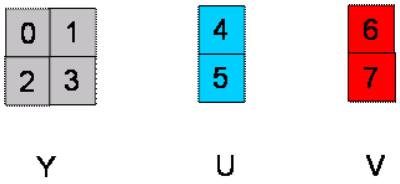
For YUV400, the value of *V1* can be 1, 2, or 3 and will be same as the value of *H1*, and the Minimum coded unit (MCU) is one 8x8 block. For the other chroma formats, if non-interleaved data, the MCU is one 8x8 block. For interleaved data, the MCU is the sequence of block units defined by the sampling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

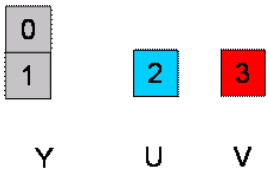
422H_2Y



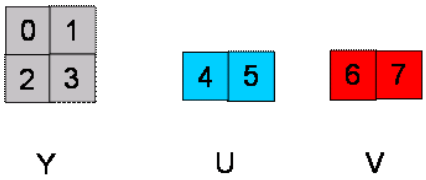
422H_4Y



422V_2Y

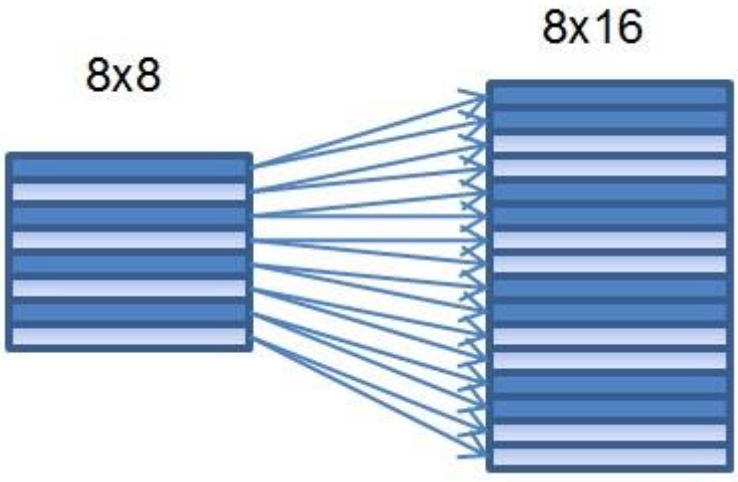


422V_4Y

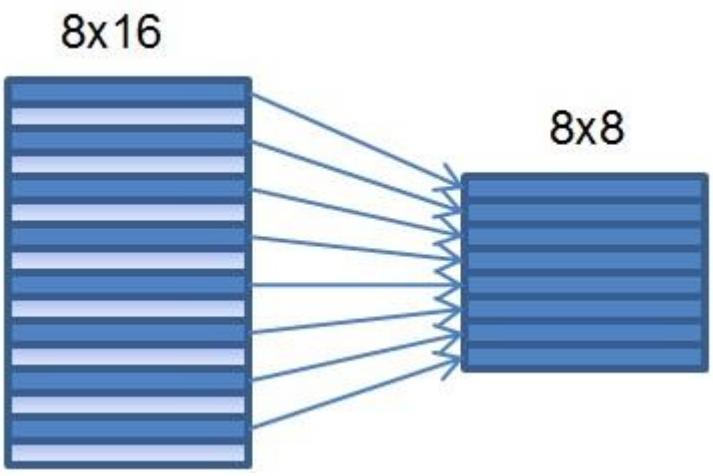


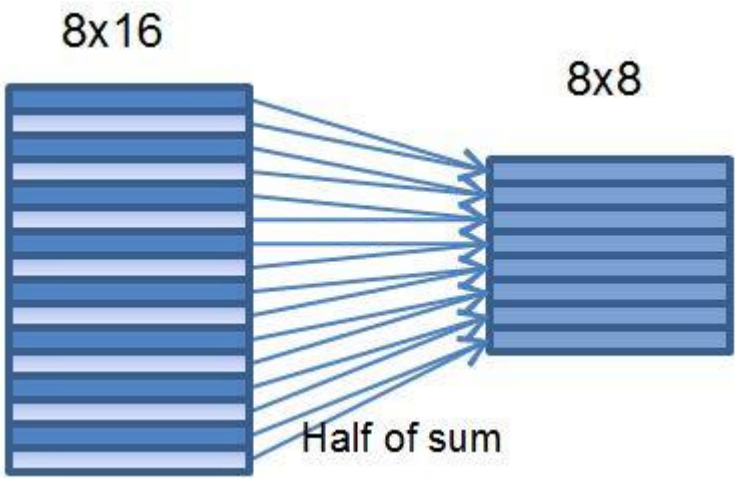
If picture width *X* in the Frame header is not a multiple of 8, the decoding process needs to extend the number of columns to complete the right-most sample blocks. If the component is to be interleaved, the decoding process needs to extend the number of samples by one or more additional blocks so that 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.

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: 
1	1	Average vertically neighboring two pixels:

VertDownSamplingEnb	AvgDownSampling	Down-Sampling Methods
		

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

- InputFormat is the same, and should be programmed the same.
- If the InputFormat is YUV400 or YUV444 or YUV411, then output cannot be NV12, YUY2 or UYVY, it has to be planar (like legacy). But for 420 and 422 InputFormat, there's a choice of having Planar, NV12, YUY2 or UYVY OutputFormat (new addition). 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:

Commands
MFX_PIPE_MODE_SELECT
MFX_SURFACE_STATE
MFX_PIPE_BUF_ADDR_STATE
MFX_IND_OBJ_BASE_ADDR_STATE
MFX_JPEG_PIC_STATE
MFX_FQM_STATE (One each for Luma, CB and CR)
MFC_JPEG_ HUFF_TABLE_STATE(Huffman table 0 and 1 need two commands to be issued).
MFC_JPEG_SCAN_OBJECT
MFX_PAK_INSERT_OBJECT (Multiple commands can be given based on the need)

Following are JPEG Encoder Commands:

MFX_JPEG_PIC_STATE command is used for both encoding and decoding. Note the duplicate bits and the "Exists If" rows that specify what the bits represent for Encoder and Decoder.

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

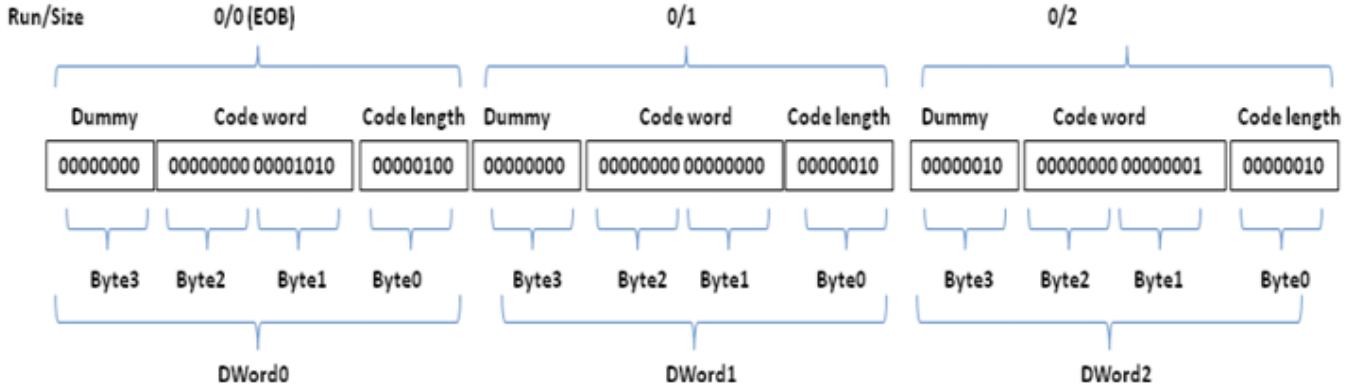
MFC_JPEG_SCAN_OBJECT

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

MFC_JPEG_ HUFF_TABLE_STATE

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.

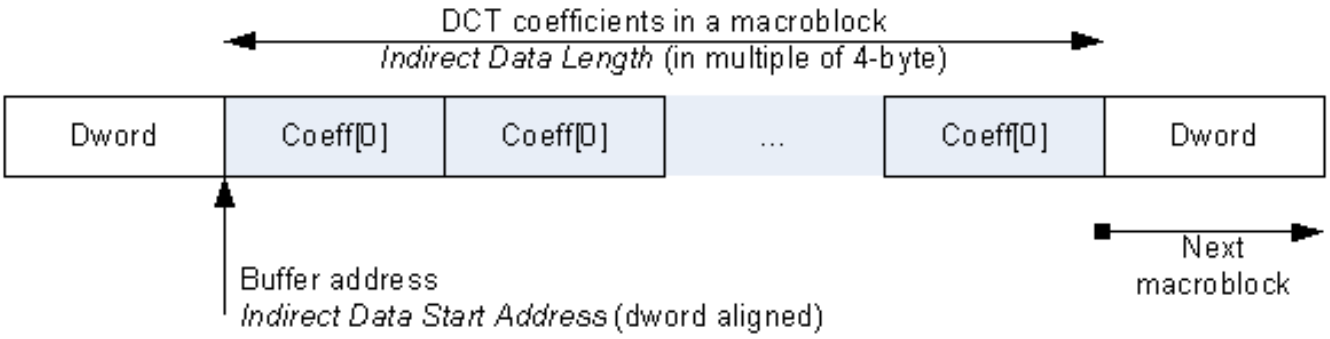
MFD_IT_OBJECT

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 Structure of the IDCT Compressed Data Buffer. 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 Structure of a transform-domain residue unit, *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. The valid range of this field depends on the size of the transform block. Format = U6 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** Specify the number of MVs (in unit of motion vector, 4 bytes each) to be fetched for motion compensation operation. Decoder IT mode only supports packed MV format (DXVA). This field specifies the exact number of MVs present for the current MB. 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. For an Intra-MB, MvQuantity is set to 0. MvQuantity = 0, signifies there is no MV indirect data for the current MB. This field must be set in consistent with **Indirect MV Data Length**, so as not to exceed its bound Unsigned.
	23:20	Reserved MBZ (DXVA)

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

DWord	Bit	Description
		1 - is an intra MB I_PCM is considered as Intra MB. For I-picture MB (IntraPicFlag =1), this field must set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes).
	12:8	**MbType** This field carries the Macroblock Type. The meaning depends on IntraMbFlag. If IntraMbFlag is 1, this field is the intra macroblock type as defined in MbType definition for Intra Macroblock_. 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. Programming note: It is exactly matched with that of DXVA 2.0.
	7	**FieldMbPolarityFlag** This field indicates the field polarity of the current macroblock. 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. 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. 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. 0 = Current macroblock is a field macroblock from the **top** field (first in a MBAFF pair) 1 = Current macroblock is a field macroblock from the **bottom** field (second in a MBAFF pair)
	6	**IsLastMB** 1 - the current MB is the last MB in the current Slice 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)
1	31:16	**CbpY[bit 15:0] (Coded Block Pattern Y)** For 4x4 sub-block (when **Transform8x8flag** = 0 or in intra16x16) : 16-bit cbp, one bit for each 4x4 Luma sub-block (not including the DC 4x4 Luma block in intra16x16) in a MB. The 4x4 Luma sub-blocks are numbered as

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<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.
For 8x8 block (when Transform8x8flag = 1)
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.
0 in a bit - indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero)
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.
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.
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.
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.
Format = U8 in unit of macroblock.
</td></tr>
<tr><td>2</td><td>31:16</td><td>
CbpCr (Coded Block Pattern Cr 4:2:0-only)
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
<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>
</td></tr>
</table>

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>The cbpCr bit assignment is cbpCr bit [3 - X] for sub-block_num X.

0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero)

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).

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)

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
<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.

0 in a bit - indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero)

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).

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

Driver is responsible for deriving the QpPrimeCr from QpPrimeY.

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

Driver is responsible for deriving the QpPrimeCb from QpPrimeY.

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>
<tr><td></td><td>7:0</td><td>QpPrimeY

This is the per-MB QP value specified for the current MB.

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.</td></tr>
<tr><td>4 to 6</td><td>31:0</td><td>For intra macroblocks, definition of these fields are specified in Inline data subfields for an Intra Macroblock</td></tr>
</table>

DWord	Bit	Description
	Each	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 MV record size 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

Macroblock Type	MVQuant
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 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)
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

DWord	Bit	Description
	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
	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)

DWord	Bit	Description
	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)
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**

DWord	Bit	Description
	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**
	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

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td>+0</td><td>31:28</td><td>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.

This field is valid for non-intra macroblock only.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>28</td><td>Forward predict of current frame/field or TOP field of interlace frame, or block 0 in 4MV mode.</td></tr>
<tr><td>29</td><td>Backward predict of current frame/field or TOP field of interlace frame, or forward predict for block 1 in 4MV mode.</td></tr>
<tr><td>30</td><td>Forward predict of BOTTOM field of interlace frame, or block 2 in 4MV mode.</td></tr>
<tr><td>31</td><td>Backward predict of BOTTOM field of interlace frame, or forward predict for block 3 in 4MV mode.</td></tr>
</table>
Each corresponding bit has the following indication.

0 = The prediction is taken from the top reference field.

1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>27</td><td>Reserved. MBZ</td></tr>
<tr><td></td><td>26</td><td>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.

Non-intra macroblock only. This field is derived from MvFieldSelect.

0 = The prediction is taken from the top reference field.

1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>25:24</td><td>MotionType - Motion Type

For frame picture, a macroblock may only be either 00 or 10.

For interlace picture, a macroblock may be of any motion types. It can be 01 if and only if DctType is 1.

This field is 00 if and only if IntraMacroblock is 1.

00 = Intra

01 = Field Motion.

10 = Frame Motion or no motion.

Others = Reserved.</td></tr>
</table>

DWord	Bits	Description
	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. 0 = No directional prediction switch from top field to bottom field 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). For progressive picture, this field must be set to '0', i.e. all macrobalcoks are frame macroblock. 0 = Frame residual type. 1 = Field residual type.
	20	**OverlapTransform.** This field indicates whether overlap smoothing filter should be performed on I-block boundaries. 0 = No overlap smoothing filter. 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. 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. 0 = 1MV-mode. 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). 0 = No backward motion vector. 1 = Use backward motion vector(s).
	17	**MotionForward.** This field specifies whether the forward motion vector is active for P and B pictures. 0 = No forward motion vector. 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).

DWord	Bits	Description
		For field motion, this field indicates whether the top field of the macroblock is coded as intra. 0 = Non-intra macroblock. 1 = Intra macroblock.
	15:12	**LumaIntra8x8Flag - Luma Intra 8x8 Flag** This field specifies whether each of the four 8x8 luminance blocks are intra or inter coded when Motion4MV is set to 4MV-Mode. Each bit corresponds to one block. "0" indicates the block is inter coded and '1' indicates the block is intra coded. When Motion4MV is not 4MV-Mode, this field is reserved and MBZ. Bit 15: Y0 Bit 14: Y1 Bit 13: Y2 Bit 12: Y3
	11:6	**CBP - Coded Block Pattern** This field specifies whether the 8x8 residue blocks in the macroblock are present or not. Each bit corresponds to one block. "0" indicates residue block isn't present, "1" indicates residue block is present. 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). Bit 11: Y0 Bit 10: Y1 Bit 9: Y2 Bit 8: Y3 Bit 7: Cb4 Bit 6: Cr5
	5	**ChromaIntraFlag - Derived Chroma Intra Flag** This field specifies whether the chroma blocks should be treated as intra blocks based on motion vector derivation process in 4MV mode. 0 = Chroma blocks are not coded as intra. 1 = Chroma blocks are coded as intra

DWord	Bits	Description
	4	**LastRowFlag - Last Row Flag** This field indicates that the current macroblock belongs to the last row of the picture. This field may be used by the kernel to manage pixel output when overlap transform is on. 0 = Not in the last row 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** This field contains the overscan edge mask for the Chroma blocks. The left edge masks are hardware and the top edge masks are used by the kernel software. Bit 24: Top edge of block Cb/Cr Bit 25: Left edge of block Cb/Cr
	23:16	**OSEdgeMaskLuma** This field contains the overscan edge mask for the Luma blocks. The left edge masks are hardware and the top edge masks are used by the kernel software. Bit 16: Top edge of block Y0 Bit 17: Top edge of block Y1 Bit 18: Top edge of block Y2 Bit 19: Top edge of block Y3 Bit 20: Left edge of block Y0 Bit 21: Left edge of block Y1 Bit 22: Left edge of block Y2 Bit 23: Left edge of block Y3 *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**

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>In unit of macroblocks relative to the current picture (frame or field).</td></tr>
<tr><td></td><td>7:0</td><td>HorzOrigin - Horizontal Origin
In unit of macroblocks.</td></tr>
<tr><td>+2</td><td>31:16</td><td>MotionVector[0].Vert</td></tr>
<tr><td></td><td>15:0</td><td>MotionVector[0].Horz</td></tr>
<tr><td>+3</td><td>31:0</td><td>MotionVector[1]</td></tr>
<tr><td>+4</td><td>31:0</td><td>MotionVector[2]</td></tr>
<tr><td>+5</td><td>31:0</td><td>MotionVector[3]</td></tr>
<tr><td>+6</td><td>31:0</td><td>MotionVectorChroma
This field is not valid for a field motion in an interlaced frame picture where 4 MVs for chroma blocks.
Notes: This field is derived from MotionVector[3:0] as described in the following section.</td></tr>
<tr><td>+7</td><td>31:24</td><td>Subblock Code for Y3
The following subblock coding definition applies to all 6 subblock coding bytes. Bits 7:6 are reserved.
<table>
<tr><th colspan="2">Subblock Partitioning (Bits [1:0]) Specify Transform uses for an 8x8 block</th><th colspan="4">Subblock Present (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>
</table>

DWord	Bits	Description
	7:0	**Subblock Code for Y0**
+8	31:16	**Reserved.** MBZ
	15:8	**Subblock Code for Cr**
	7:0	**Subblock Code for Cb**
+9	31:24	**ILDB control data for block Y3**
	23:16	**ILDB control data for block Y2**
	15:8	**ILDB control data for block Y1**
	7:0	**ILDB control data for block Y0**
+10	31:16	**Reserved**
	15:8	**ILDB control data for Cr block**
	7:0	**ILDB control data for Cb block**

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 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 Section 6.3.17.2 of the ISO/IEC 13818-2 (see also Section 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.
0 = The prediction is taken from the top reference field.
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 Section 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.
Format = MC_MotionType
<table>
<tr><th>Value</th><th>Destination = Frame
Picture_Structure = 11</th><th>Destination = Field
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></td></tr>
<tr><td></td><td>23:22</td><td>Reserved. (Scan method)</td></tr>
<tr><td></td><td>21</td><td>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 Section 6.3.17.1. This field is zero if Coded Block Pattern is also zero (no coded blocks present).
0 = MC_FRAME_DCT (Macroblock is frame DCT coded).
1 = MC_FIELD_DCT (Macroblock is field DCT coded).</td></tr>
<tr><td></td><td>20</td><td>Reserved (was Overlap Transform - H261 Loop Filter).</td></tr>
<tr><td></td><td>19</td><td>Reserved (was 4MV Mode - H263/WMV)</td></tr>
<tr><td></td><td>18</td><td>Macroblock Motion Backward. This field specifies if the backward motion vector is active. See</td></tr>
</table>

DWord	Bit	Description
		ISO/IEC 13818-2 Tables B-2 through B-4. 0 = No backward motion vector. 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. 0 = No forward motion vector. 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. 0 = Non-intra macroblock. 1 = Intra macroblock.
	15:12	**Reserved : MBZ**
	11:6	**Coded Block Pattern.** This field specifies whether blocks are present or not. Format = 6-bit mask. Bit 11: Y0 Bit 10: Y1 Bit 9: Y2 Bit 8: Y3 Bit 7: Cb4 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** In unit of macroblocks relative to the current picture (frame or field).
	7:0	**HorzOrigin - Horizontal Origin** In unit of macroblocks.
+2	31:16	**Motion Vectors - Field 0, Forward, Vertical Component.** Each vector component is a 16-bit

DWord	Bit	Description
		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

MFX Error Handling

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	**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) The same value as **RateControlCounterEnable** of MFX_AVC_SLICE_STATE Command
	1	**MbInterConfFlag:** MB level InterMB conformance flag to trigger mutli-pass 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 1- if total Bit Count of an intra macroblock is more than Intra Conformance Max size limit in the 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) 1 - Indicates if clamping of any coefficient is done on the macroblock for Rate Control
	15:0	Reserved (future QRC stat output)

PAK Frame Statistics StreamOut

The following frame statistics are written to memory at the conclusion of a frame. If Multipass occurs, these values are overwritten by the end of any subsequent passes of the current frame (hence it contains only the final pass statistics).

The streamout is done to the MB streamout surface, starting at the next CL boundary. If MB streamout is disabled, Frame level streamout starts with 0 offset.

MFX_PAK_FRAME_STATISTICS		
Source:	**VideoCS**	
Length Bias:	**2**	
DWord	**Bit**	**Description**
0	31:16	Reserved : MBZ
	15:0	SumSliceHeader - Report the total size (in bits) of all slice headers inserted into the bitstream for this frame.
1	31:0	SumMBHeader - Report the total size (in bits) of all MB headers (non coeff bits) inserted into the bitstream for this frame.
2	31:0	SumNZC - Report the total number of nonzero coefficients after quantization.
3	31:0	Reserved: MBZ
4	31:16	IntraMB16x16 - Count of # of MB's that were of type Intra 16x16
	15:0	IntraMB8x8 - Count of # of MB's that were of type Intra 8x8
5	31:16	IntraMB4x4 - Count of # of MB's that were of type Intra 4x4
	15:0	InterMB16x16 - Count of # of MB's that were of type Inter 16x16
6	31:16	InterMB16x8 - Count of # of MB's that were of type Inter 16x8
	15:0	InterMB8x16 - Count of # of MB's that were of type Inter 8x16
7	31:16	InterMB8x8 - Count of # of MB's that were of type Inter 8x8
	15:0	InterSkip16x16 - Count of # of MB's that were of type Inter 16x16 skip
8:49	31:0	RhoDomainStats - Each DW contains 1 of the 42 registers containing the raw Rho Domain coefficient metrics. DW 8 is QP 10 and DW 49 is QP51.
50	31:0	Reserved: MBZ

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

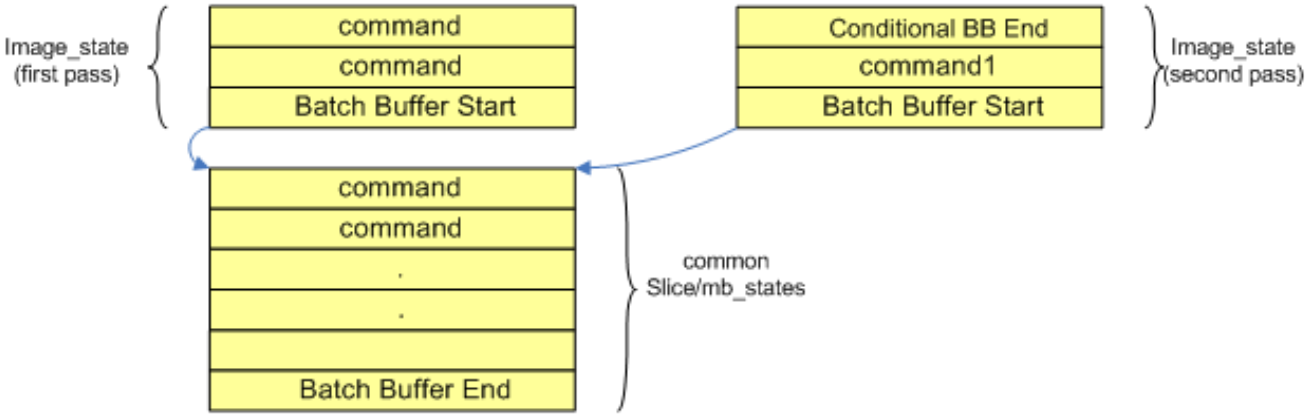
Following 2 MI packets are used inside VCS without any change to support Multipass-PAK behaviour.

- 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 cannot 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:

Registers
MFD_ERROR_STATUS - MFD Error Status
AVC CAVLC
AVC CABAC
VC1
MPEG2

Registers
JPEG
MFD_PICTURE_PARAM - MFD Picture Parameter
MFX_STATUS_FLAGS - MFX Pipeline Status Flags
MFX_MB_COUNT - MFX Frame Macroblock Count
MFX_SE-BIN_CT - MFX Frame BitStream SE/BIN Count

Encoder Registers

Following are the Encoder Registers:

Register
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
MFX_VP8_CNTRL_MASK - Reported BitRateControl parameter Mask
MFX_VP8_CNTRL_STATUS - Reported BitRateControl parameter Status
MFX_VP8_FRM_BYTE_CNT - Reported Final Bitstream Byte Count
MFX_VP8_FRM_ZERO_PAD - Reported Frame Zero Padding Byte Count
MFX_VP8_BRC_DQindex - Reported BitRateControl DeltaQindex
MFX_VP8_BRC_DLoopFilter - Reported BitRateControl DeltaLoopFilter
MFX_VP8_BRC_CumulativeDQindex01 - Reported BitRateControl CumulativeDeltaQindex and Qindex 01
MFX_VP8_BRC_CumulativeDQindex23 - Reported BitRateControl CumulativeDeltaQindex and Qindex 23
MFX_VP8_BRC_CumulativeDLoopFilter01 - Reported BitRateControl CumulativeDeltaLoopFilter and LoopFilter 01
MFX_VP8_BRC_CumulativeDLoopFilter23 - Reported BitRateControl CumulativeDeltaLoopFilter and LoopFilter 23
MFX_VP8_BRC_Convergence_Status - Reported BitRateControl Convergence Status

MMIO Interface

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

- Use as Status register for Bit Rate Control
- Use for Context Switch in Multipass

Register Name	Description	Register Type	Address Offset	Dec/Enc
MFX_VP8_CNTRL_MASK	BitRateControl parameter Mask register	RO	12900	Enc
MFX_VP8_CNTRL_STATUS	BitRateControl parameter Status register	RO	12904	Enc
MFX_VP8_FRM_BYTE_CNT	Final Bitstream Byte count	RO	12908	Enc
MFX_VP8_FRM_ZERO_PAD	Final Bitstream Zero Padding Byte count	RO	1290B	Enc
MFX_VP8_BRC_DQindex	BitRateControl Delta Qindex	RO	12910	Enc
MFX_VP8_BRC_DLoopFilter	BitRateControl Delta LoopFilter	RO	12914	Enc
MFX_VP8_BRC_CumulativeDQindex01	BitRateControl Cumulative Delta Qindex for Seg0/1	RW	12918	Enc
MFX_VP8_BRC_CumulativeDQindex23	BitRateControl Cumulative Delta Qindex for Seg2/3	RW	1291C	Enc
MFX_VP8_BRC_CumulativeDLoopFilter01	BitRateControl Cumulative Delta LoopFilter for Seg0/1	RW	12920	Enc
MFX_VP8_BRC_CumulativeDLoopFilter23	BitRateControl Cumulative Delta LoopFilter for Seg2/3	RW	12924	Enc
MFX_VP8_BRC_Convergence_Status	BitRateControl Convergence Status	RW	12928	Enc
MFX_VP8_DEBUG_CPBAC0_Bottom	CPBAC0 engine Bottom State	RO	1292C	Enc
MFX_VP8_DEBUG_CPBAC0_RangeCount	CPBAC0 engine Range/Count State	RO	12930	Enc
MFX_VP8_DEBUG_CPBAC0_Misc	CPBAC0 engine Misc. States	RO	12934	Enc
MFX_VP8_DEBUG_CPBAC1_Bottom	CPBAC1 engine Bottom State	RO	12938	Enc
MFX_VP8_DEBUG_CPBAC1_RangeCount	CPBAC1 engine Range/Count State	RO	1293C	Enc
MFX_VP8_DEBUG_CPBAC1_Misc	CPBAC1 engine Misc. States	RO	12940	Enc

The following registers are the same as above except they have a different Address Offset. They are used if the second VDbox (VP8 Encoder) exists.

Register Name	Description	Register Type	Address Offset	Dec/Enc
MFX_VP8_CNTRL_MASK	BitRateControl parameter Mask register	RO	1C900	Enc
MFX_VP8_CNTRL_STATUS	BitRateControl parameter Status register	RO	1C904	Enc
MFX_VP8_FRM_BYTE_CNT	Final Bitstream Byte count	RO	1C908	Enc
MFX_VP8_FRM_ZERO_PAD	Final Bitstream Zero Padding Byte count	RO	1C90B	Enc
MFX_VP8_BRC_DQindex	BitRateControl Delta Qindex	RO	1C910	Enc
MFX_VP8_BRC_DLoopFilter	BitRateControl Delta LoopFilter	RO	1C914	Enc
MFX_VP8_BRC_CumulativeDQindex01	BitRateControl Cumulative Delta Qindex for Seg0/1	RW	1C918	Enc
MFX_VP8_BRC_CumulativeDQindex23	BitRateControl Cumulative Delta Qindex for Seg2/3	RW	1C91C	Enc
MFX_VP8_BRC_CumulativeDLoopFilter01	BitRateControl Cumulative Delta LoopFilter for Seg0/1	RW	1C920	Enc
MFX_VP8_BRC_CumulativeDLoopFilter23	BitRateControl Cumulative Delta LoopFilter for Seg2/3	RW	1C924	Enc
MFX_VP8_BRC_Convergence_Status	BitRateControl Convergence Status	RW	1C928	Enc
MFX_VP8_DEBUG_CPBAC0_Bottom	CPBAC0 engine Bottom State	RO	1C92C	Enc
MFX_VP8_DEBUG_CPBAC0_RangeCount	CPBAC0 engine Range/Count State	RO	1C930	Enc
MFX_VP8_DEBUG_CPBAC0_Misc	CPBAC0 engine Misc. States	RO	1C934	Enc
MFX_VP8_DEBUG_CPBAC1_Bottom	CPBAC1 engine Bottom State	RO	1C938	Enc
MFX_VP8_DEBUG_CPBAC1_RangeCount	CPBAC1 engine Range/Count State	RO	1C93C	Enc
MFX_VP8_DEBUG_CPBAC1_Misc	CPBAC1 engine Misc. States	RO	1C940	Enc

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					

	Write		Surf Size
	Read		

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

VDBOX Registers

This section describes the VDBOX Command Memory Interface registers.

MMIO Ranges

MMIO ranges for media are described in this section. The base address of MFX(x), HuC(x), VCS(x), VECS(x), HEVC(x) are modified.

HEVC MMIO is split into two ranges as HEVC is split into frontend and Backend. x value can range from 0 through 7.

The address offset is defined in hierarchical manner. Each VDBOX has 16KB of MMIO address range and is allocated as shown in the table below. Unallocated address with-in 16KB space would be claimed by HEVCFE for writes and read zeros.

Offset to Scalable Engines:

UNIT	Address	Size
VCS (range 0)	0x0000 - 0x07FF	2 KB
MFX Pipe (VIN)	0x0800 - 0x0FFF	2 KB
VCS (range 1)	0x1000 - 0x1FFF	4 KB
HEVC Pipe (HWM)	0x2800 - 0x2AFF	750 B
AVP Pipe (AWM)	0x2B00 - 0x2CFF	500 B
VDENC	0x2D00-0x2DFF	1KB
Reserved	0x2E00-0x3EFF	4096B
CFCFG	0x3F00-0x3FFF	128B
SCR (no mmio space but MsgCh endpoints)		
Total allocation:		12.5 KB

VDBOX and VEBOX Offset Table:

Media Boxes	Base Address	Offset Range	Size	Media sliceid[2:0]	Media subsliceid[1:0]
VDBOX0	0x1C_0000	0x0000 - 0x3FFF	16KB	000	00
VDBOX1	0x1C_4000	0x0000 - 0x3FFF	16KB	000	01
VEBOX0	0x1C_8000	0x0000 - 0x3FFF	16KB	000	00
VDBOX2	0x1D_0000	0x0000 - 0x3FFF	16KB	001	00
VDBOX3	0x1D_4000	0x0000 - 0x3FFF	16KB	001	01
VEBOX1	0x1D_8000	0x0000 - 0x3FFF	16KB	001	00
VDBOX4	0x1E_0000	0x0000 - 0x3FFF	16KB	010	00
VDBOX5	0x1E_4000	0x0000 - 0x3FFF	16KB	010	01
VEBOX2	0x1E_8000	0x0000 - 0x3FFF	16KB	010	00
VDBOX6	0x1F_0000	0x0000 - 0x3FFF	16KB	011	00

Media Boxes	Base Address	Offset Range	Size	Media sliceid[2:0]	Media subsliceid[1:0]
VDBOX7	0x1F_4000	0x0000 - 0x3FFF	16KB	011	01
VEBOX3	0x1F_8000	0x0000 - 0x3FFF	16KB	011	00

Media VEBOX

This chapter describes the VEBOX Media Engine.

Media VEBOX Introduction

The VEBOX is an independent pipe with a variety of image enhancement functions.

The following sections are contained in Media VEBOX:

Feature
Denoise
Deinterlacer
Image Enhancement/Color Processing (IECP)
Capture Pipe
VEBOX State
VEBOX Surface State
VEB DI IECP Commands
Command Stream Backend - Video
Video Enhancement Engine Functions

The IECP consists of these functions:

Feature
STD - Skin Tone Detection detects colors which might represent skin. STE - Skin Tone Enhancement modifies colors marked by STD. GCC - Gamut Compression ACE - Automatic Contrast Enhancement changes luma values to enhance contrast.
LACE - Local Automatic Contrast Enhancement.
TCC - Total Color Control allows UV values to be modified to adjust color saturation. ProcAmp - implements the ProcAmp DDI functions to modify the brightness, contrast, hue, and saturation. CSC - Color Space Conversion GEE - Gamut Expansion and Color Correction in Linear RGB Space

Programming Note
The input and output dimensions are restricted to 16K for VEBOX DN/DI/IECP/Capture Pipe.

VEBOX State and Primitive Commands

Every engine can have internal state that can be common and reused across the data entities it processes instead of reloading for every data entity.

There are two kinds of state information:

1. Surface state or state of the input and output data containers.
2. Engine state or the architectural state of the processing unit.

For example in the case of DN/DI, architectural state information such as denoise filter strength can be the same across frames. This section gives the details of both the surface state and engine state.

Each frame should have these commands, in this order:

1. VEBOX_State
2. VEBOX_Surface_state for input & output
3. VEB_DI_IECP

Alternatively, VEBOX_Tiling_Convert can be used instead of VEB_DI_IECP.

VEBOX State

This chapter discusses various commands that control the internal functions of the VEBOX. The following commands are covered:

Command
DN/DI State Table Contents
VEBOX_IECP_STATE
VEBOX_FORWARD_GAMMA_CORRECTION_STATE
VEBOX_STATE **VEBOX_Ch_Dir_Filter_Coefficient**

DN-DI State Table Contents

This section contains tables that describe the state commands that are used by the Denoise and Deinterlacer functions.

VEBOX_DNDI_STATE

VEBOX_IECP_STATE

For all piecewise linear functions in the following table, the control points must be monotonically increasing (increasing continuously) from the lowest control point to the highest. Functions which have bias/correction values associated with each control point have the additional restriction that any control points which have the same value must also have the same bias/correction value. The piecewise linear functions include:

- For Skin Tone Detection:
    - Y_point_4 to Y_point_0
    - P3L to P0L
    - P3U to P0U
    - SATP3 to SATP1
    - HUEP3 to HUEP1
    - SATP3_DARK to SATP1_DARK
    - HUEP3_DARK to HUEP1_DARK
- For ACE/LACE:
    - Ymax, Y10 to Y1 and Ymin
    - There is no state variable to set the bias for Ymin and Ymax. The biases for these two points are equal to the control point values: B0 = Ymin and B11 = Ymax. That means that if control points adjacent to Ymin and Ymax have the same value as Ymin/Ymax then the biases must also be equal to the Ymin/Ymax control points based on the restriction mentioned above.
    - LACE gamma curve PWL (16 points & bias values)
- Forward Gamma correction
    - Gamma correction table (1K points & correction values)
- HDR
    - HDR Inverse gamma (4K correction values)
    - HDR Forward gamma (256 points & correction values)
    - Tone Mapping LUT (16 points & correction values)
- Gamut Expansion:
    - Gamma Correction (256 points & correction values)

Inverse Gamma Correction (256 points & correction values)

Command
VEBOX_STD_STE_STATE
VEBOX_ACE_LACE_STATE
VEBOX_TCC_STATE
VEBOX_PROCAMP_STATE
VEBOX_CSC_STATE
VEBOX_ALPHA_AOI_STATE
VEBOX_CCM_STATE
VEBOX_FRONT_END_CSC_STATE
VEBOX_GAMUT_CONTROL_STATE
Gamut_Expansion_Gamma_Correction
VEBOX_VERTEX_TABLE
VEBOX_CAPTURE_PIPE_STATE
VEBOX_FORWARD_GAMMA_CORRECTION_STATE
VEBOX_RGB_TO_GAMMA_CORRECTION

VEBOX Surface State

VEBOX_SURFACE_STATE

Surface Format Restrictions

The surface formats and tiling allowed are restricted, depending on which function is consuming or producing the surface.

Surface Format Restrictions

FourCC Code	Format	DN/DI Input	DN/DI Output	IECP Input	IECP Output	Capture Output	Scalar Input/Output
YUYV	YCRCB_NORMAL (4:2:2)	X	X	X	X	X	X
VYUY	YCRCB_SwapUVY (4:2:2)	X	X	X	X	X	X
YVYU	YCRCB_SwapUV (4:2:2)	X	X	X	X	X	X
UYVY	YCRCB_SwapY (4:2:2)	X	X	X	X	X	X
Y8	Y8 Monochrome	X	X	X	X	X	X
NV12	NV12 (4:2:0 with interleaved U/V)	X	X	X	X	X	X
AYUV	4:4:4 with Alpha (8-bit per channel)			X	X	X	X
Y216	4:2:2 packed 16-bit			X	X	X	X
Y416	4:4:4 packed 16-bit			X	X	X	X
Y410	4:4:4 packed 10-bit				X	X	X
P216	4:2:2 planar 16-bit			X	X	X	X
P016	4:2:0 planar 16-bit			X	X	X	X
Y16	Y16 Monochrome	X	X	X	X	X	X
	RGBA 10:10:10:2				X	X	
	RGBA 8:8:8:8	Spatial DN		X	X	X	
	RGBA 16:16:16:16	Spatial DN		X	X	X	
	BGRA 8:8:8:8				X	X	
Tiling							
	Tile Y	X	X	X	X	X	X
	Tile X	X	X	X	X	X	X
	Linear	X	X	X	X	X	X

Surface Format Restrictions

FourCC Code	Format	DN Input/Output	DI Input/Output	IECP Input	IECP Output	Capture Output	Scalar Input/Output
YUYV	YCRCB_NORMAL (4:2:2)	X	X	X	X	X	X
VYUY	YCRCB_SwapUVY (4:2:2)	X	X	X	X	X	X
YVYU	YCRCB_SwapUV (4:2:2)	X	X	X	X	X	X
UYVY	YCRCB_SwapY (4:2:2)	X	X	X	X	X	X
Y8	Y8 Monochrome	X	X	X	X	X	X
NV12	NV12 (4:2:0 with interleaved U/V)	X	X	X	X	X	X
AYUV	4:4:4 with Alpha (8-bit per channel)	X	Output only	X	X	X	X
Y216	4:2:2 packed 16-bit	X	X	X	X	X	X
Y416	4:4:4 packed 16-bit	X	Output only	X	X	X	X
Y410	4:4:4 packed 10-bit	X	Output only	X	X	X	X
P216	4:2:2 planar 16-bit	X	X	X	X	X	X
P016	4:2:0 planar 16-bit	X	X	X	X	X	X
Y16	Y16 Monochrome	X	X	X	X	X	X
	RGBA 10:10:10:2				X	X	
	RGBA 8:8:8:8	Spatial DN		X	X	X	
	RGBA 16:16:16:16	Spatial DN		X	X	X	
	BGRA 8:8:8:8				X	X	
Tiling							
	Tile Y	X	X	X	X	X	X
	Tile X	X	X	X	X	X	X
	Linear	X	X	X	X	X	X

Surface Formats - Feature Notes

Feature
Surfaces are 4 kb aligned, chroma X offset is cache line aligned (16 byte).
If Y8/Y16 is used as the input format, it must also be used for the output format (chroma is not created by VEBOX).
If IECP and either DN or DI are enabled at the same time, it is possible to select any input that is legal for DN/DI and any output which is legal for IECP. The only exception is that if DN or DI are enabled, the IECP is not able to output P216 and P016.
16-bit data from IECP or DN is rounded when converting to 8-bit output formats.
High Speed Bypass has the same format limitations as IECP Input/Output, but the surface formats for the input and output must be the same.
Capture Input is only linear Bayer Surface Format.
Input formats for Demosaic, White Balance, Vignette and Black Level Correction must be linear Bayer.
For capture pipe, we **can** support the combination of DN and P216 and P016. For capture pipe with a P016 output the U/V output is not an average of the 4 component pixels, but the U/V for pixel 4 (the lower right pixel of the 4).
Output format Y410 is supported for DN, DI and DM modes only with IECP enabled. In non IECP cases, default of "0xFFFF" is sent if output format requires alpha.
For DN 444 input formats interlaced input content is not supported
If IECP and either DN or DI are enabled at the same time, it is possible to select any input that is legal for DN/DI and any output which is legal for IECP.

SFC

This chapter describes the SFC Media Engine.

SFC Overview

Scaler & Format Converter (SFC) pipeline is introduced on Skylake as a multi-format scaling engine to accelerate several media usages and achieve ultra-low-power video playback.

SFC Commands Definition

This section contains definitions for commands used with the scaler and format converter (SFC). These commands are sent from the VDBOX/VEBOX to the SFC pipeline.

SFC_AVS_LUMA_Coeff_Table

SFC_AVS_CHROMA_Coeff_Table

SFC_AVS_STATE

SFC_FRAME_START

SFC_LOCK

SFC_STATE