Intel® OpenSource HD Graphics Programmer's Reference Manual (PRM)

Volume 4 Part 2: Shared Functions – Message Gateway, URB, Video Motion Estimation, Pixel Interpolator (Ivy Bridge)

For the 2012 Intel® Core™ Processor Family

May 2012

Revision 1.0

NOTICE:
This document contains information on products in the design phase of development, and Intel reserves the right to add or remove product features at any time, with or without changes to this open source documentation.

Doc Ref #: IHD-OS-V4 Pt 2 – 05 12

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work.

Contents

1. Shared Functions - Message Gateway 5
1.1 Messages 5
1.1.1 Message Descriptor 6
1.1.2 OpenGateway Message 7
1.1.3 CloseGateway Message 8
1.1.4 GetTimeStamp Message 11
1.1.5 BarrierMsg Message 13
1.1.6 MMIOReadWrite Message 15
2. Shared Functions - Unified Return Buffer (URB) 17
2.1 URB Size 17
2.2 URB Access 17
2.3 State 18
2.4 URB Messages 18
2.4.1 Execution Mask 19
2.4.2 Message Descriptor 19
2.4.3 URB_WRITE* and URB_READ* 20
2.4.4 URB_ATOMIC* 32
3. Shared Functions – Video Motion Estimation 34
3.1 Theory of Operation 34
3.1.1 Shape Decision 34
3.1.2 Integer Motion Estimation 39
3.1.3 Fractional Motion Estimation 44
3.1.4 BME and Weighted Prediction 45
3.1.5 Skip Check 46
3.1.6 Intra Prediction Estimation 48
3.1.7 Transform Adjusted SAD 48
3.1.8 Early Decisions 50
3.1.9 Performance Information 51
3.1.10 VME Changes 52
3.2 Surfaces 52
3.3 State 52
3.3.1 BINDING_TABLE_STATE 52
3.3.2 SURFACE_STATE 52
3.3.3 VME_STATE 53
3.4 Change Details 56
3.4.1 Record Stream-out and Stream-in 56
3.4.2 MV Definitions and Precision 57
3.4.3 Expanded MV Costs 58
3.4.4 Remove Skip MV Restriction 59
3.4.5 Bilinear Interpolation 59
3.4.6 AVC Intra Mode Mask 59
3.5 Messages 59
3.5.1 VME Motion Search Request 60
3.5.2 Message Descriptor 60
3.5.3 Input Message 61
3.5.4 Writeback Message 80
3.5.5 Stream-in\Stream-out Message 90
4. Shared Functions Pixel Interpolater 93

4.1 Messages 93
4.1.1 Initiating Message 93
4.1.2 Writeback Message 98

1. Shared Functions - Message Gateway

The Message Gateway shared function provides a mechanism for active thread-to-thread communication. Such thread-to-thread communication is based on direct register access. One thread, a **requester thread**, is capable of writing into the GRF register space of another thread, a **recipient thread**. Such direct register access between two threads in a multi-processor environment some time is referred to as **remote register access**. Remote register access may include read or write. The architecture supports **remote register write**, but not remote register read (natively). Message Gateway facilitates such remote register write via message passing. The requester thread sends a message to Message Gateway requesting a write to the recipient thread's GRF register space. Message Gateway sends a writeback message to the recipient thread to complete the register write on behave of the requester. The requester thread and the recipient thread may be on the same EU or on different EUs.

When Bypass Gateway Control is set to 1, commands OpenGateway and CloseGateway are no longer used, the gateway parameters are taking the default values as the following:

- **RegBase** = 0
- **Gateway Size** check and **Key** check are bypassed.
- **Gateway Open** (an internal signal that is used to be set by OpenGateway message) check is bypassed

A separate Gateway exists per half-slice in the architecture. For ForwardMsg this is handled transparently, but barriers can only be accessed by threads in the local half-slice. This means that all threads that access a shared barrier need to use the half-slice select in GPGPU_OBJECT and MEDIA_OBJECT to stay on a single half-slice. GPGPU_WALKER handles this automatically.

1.1 Messages

Message Gateway supports such thread-to-thread communication with the following three messages:

- **OpenGateway:** opens a gateway for a requester thread. Once a thread successfully opens its gateway, it can be a recipient thread to receive remote register write.
- **CloseGateway:** closes the gateway for a requester thread. Once a thread successfully closes its gateway, Message Gateway will block any future remote register writes to this thread.
- **ForwardMsg:** forwards a formatted message (remote register write) from a requester thread to a recipient thread.
- **GetTimeStamp:** reads absolute and relative timestamps for a requester thread.
- **BarrierMsg:** A set of threads sends this message to the Gateway. When all threads in a group have sent the message, a reply (both a register write and an N0 notification) is sent to each member of the group.
- **UpdateGatewayState:** updates the internal state of the Message Gateway.

  One example usage is to allow a control thread to change Barrier Byte to convey dynamic state information. This may be used to support interrupt when persistent compute/worker threads are synchronized using Barrier.
- **MMIO Read/Write:** allows a message to read or write an MMIO register. The MEDIA_VFE_STATE command has a field which limits the accesses for security.

1.1.1 Message Descriptor

The following message descriptor applies to all messages supported by Message Gateway.

Bit	Description
19	**Header Present** This bit must be set to **zero** for all Message Gateway messages. (this bit is not part of the shared function specific message descriptor)
18:17	Ignored (these bits are not part of the shared function specific message descriptor)
16:15	**Notify.** Send Notification Signal. This is a two-bit field indicating which notify event is sent. 00: No notify 01: Increment recipient thread's N0 notification counter 10: Increment recepient thread's N2 notification counter 11: Reserved This field is only valid for a ForwardMsg message. It is ignored for other messages. The BarrierMsg message always increments the N0 notification counter.
14	**AckReq.** Acknowledgment Required. When this bit is set, an acknowledgment return message is required. Message Gateway will send a writeback message containing the error code to the requester thread using the post destination register address. When this bit is not set, no writeback message is sent to the requesting thread by Message Gateway, even if an error occurs. This field is valid for OpenGateway, CloseGateway, ForwardMsg and BarrierMsg messages. When this bit is set, post destination register must be valid and the response length must be 1. When this bit is not set, post destination register must be *null* and the response length must be 0. This bit cannot be set when EOT is set; otherwise, hardware behavior is undefined. 0 = No Acknowledgement is required. 1 = Acknowledgement is required.
13:3	Reserved: MBZ
2:0	**SubFuncID.** Identify the supported sub-functions by Message Gateway. Encodings are: 000 = **OpenGateway.** Open the gateway for the requester thread. 001 = **CloseGateway.** Close the gateway for the requester thread. 010 = **ForwardMsg.** Forward the formatted message to the recipient thread with the given offset from the recipient's register base. 011 = **GetTimeStamp**. Read absolute and relative timestamps. 100 = **BarrierMsg.** Record an additional thread reaching the barrier. 101 = **UpdateGatewayState.** Update the barrier byte for a barrier. 110 = **MMIO Read/Write** Others are reserved

1.1.2 OpenGateway Message

The OpenGateway message opens a communication channel between the requesting thread and other threads. It specifies a key for other threads to access its gateway, as well as the GRF register range allowed to be written. The message consists of a single 256-bit message payload.

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requesting thread after completion of the OpenGateway function. Only the least significant DWord in the post destination register is overwritten.

If the EOT is set for this message, Message Gateway will ignore this message; instead, it will close the gateway for the requesting thread regardless of the previous state of the gateway.

It is software's policy to determine how to generate the key.

1.1.2.1 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:29	**Reserved**: MBZ
	28:21	**RegBase:** The register base address to be stored in the Message Gateway. It is used to compute the destination GRF register address from the offset field in ForwardMsg. RegBase contains 256-bit GRF aligned register address. Note 1: This field aligns with bits [28:21] of the Offset field of the message payload for ForwardMsg. Note 2: the most significant bit of this field must be zero. Format = U8 Range = [0,127]
	20:11	**Reserved**: MBZ
	10:8	**Gateway Size:** The range limit for messages through the Message Gateway. The maximal allowed Gateway Size is 32 GRF registers. 000: 1 GRF Register 001: 2 GRF Registers 010: 4 GRF Registers 011: 8 GRF Registers 100: 16 GRF Registers 101: 32 GRF Registers
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.

DWord	Bit	Description
M0.4	31:16	**Reserved**: MBZ
	15:0	**Reserved:** MBZ
M0.3:0		Ignored

1.1.2.2 Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 101: *Opcode Error.* Attempt to send a message which is not either open/close/forward other codes: Reserved

1.1.3 CloseGateway Message

The CloseGateway message closes a communication channel for the requesting thread that was previously opened with OpenGateway. Each thread is allowed to have only one open gateway at a time, thus no additional information in the message payload is required to close the gateway. The message consists of a single 256-bit message payload.is

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requesting thread after completion of the CloseGateway function. Only the least significant DWord in the post destination register is overwritten.

1.1.3.1 Message Payload

DWord	Bit	Description
M0.7:6		**Ignored**
M0.5	31:8	**Ignored**
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4:0		**Ignored**

1.1.3.2 Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 101: *Opcode Error.* Attempt to send a message which is not either open/close/forward other codes: Reserved

1.1.3.3 ForwardMsg Message

The ForwardMsg message gives the ability for a requester thread to write a **data segment** in the form of a byte, a dword, 2 dwords, or 4 dwords to a GRF register in a recipient thread. The message consists of a single 256-bit message payload, which contains the specially formatted data segment.

The ForwardMsg message utilizes a communication channel previously opened by the recipient thread. The recipient thread has communicated its EUID, TID, and key to the requester thread previously via some other mechanism. Generally, this is done through the thread spawn message from parent to child thread, allowing each child (requester) to then communicate with its parent through a gateway opened by the parent (recipient). The child could then use ForwardMsg message to communicate its own EUID, TID, and key back to the parent to enable bi-directional communication after opening its own gateway.

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requester thread after completion of the ForwardMsg function. Only the least significant DWord in the post destination register is overwritten.

If the Notify bit in the message descriptor is set, a 'notification' is sent to the recipient thread in order to increment the recipient thread's notification counter. This allows multiple messages to be sent to the recipient without waking up the recipient thread. The last message, having this bit set, will then wake up the recipient thread.

1.1.3.4 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:29	Reserved: MBZ
	28:16	**Offset:** It provides the destination register position in the recipient thread GRF register space as the offset from the RegBase stored in the recipient thread's gateway entry. The offset is in unit of byte, such that bits [28:21] is the 256-bit aligned register offset and bits [4:0] is the sub-register offset. The sub-register offset must be aligned to the Length field in bits [10:8]. The subfields of Offset are further

DWord	Bit	Description	
		illustrated as the following. Offset[28:21]: Register offset from the gateway base (Range [0, 127]: bit 12 MBZ) Offset[20:18]: DW offset Offset[17:16]: Byte offset (must be 00 for all DW length cases) **Programming restriction:** R0 can not be used as destination GRF register for ForwardMsg. NULL register is also not allowed as destination.	
	15:11	Reserved: MBZ	
	10:8	**Length:** The length of the data segment. 000: 1 byte 001: 1 word 010: 1 dword 011: 2 dwords 100: 4 dwords 101-111: Reserved	
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.	
M0.4	31:30	Ignored	
	29		
	28		
	31:30		
	29:28		
	27:24	**EUID:** The Execution Unit ID as part of the Recipient field is used to identify the recipient thread to whom the message is forwarded.	
	23:19	Ignored	
	18:16	**TID:** The Thread ID as part of the Recipient field is used to identify the recipient thread to whom the message is forwarded.	
	15:0	**Key** The key to match with the one stored in the recipient thread's entry in Message Gateway. Ignored	
M0.3	31:0	**Data Segment DWord 3:** valid only for the 4-DWord data segment length	
M0.2	31:0	**Data Segment DWord 2:** valid only for the 4-DWord data segment length	
M0.1	31:0	**Data Segment Dword 1:** valid only for the 2- and 4-DWord data segment lengths	
M0.0	31:24	**Data Segment Byte 0:** the same byte must be copied to all four positions within this DWord. Valid only for the 1-Byte	**Data Segment Dword 0:** valid only for the 1-, 2- and 4-Dword data segment

DWord	Bit	Description	
		data segment length.	lengths
	23:16	**Data Segment Byte 0**	
	15:8	**Data Segment Byte 0**	
	7:0	**Data Segment Byte 0**	

1.1.3.5 Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 001*:* Reserved 010: *Gateway Closed*. Attempt to send a message through a closed gateway 101: *Opcode Error.* Attempt to send a message which is not either open/close/forward 110: *Invalid Message Size.* Attempt to forward a message with length greater than 4 DW 111: Reserved

1.1.3.6 Writeback Message to Recipient Thread

This message contains the byte or dwords data segment indicated in the message written to the GRF register offset indicated. Only the byte/dword(s) will be enabled, all other data in the GRF register is untouched.

1.1.4 GetTimeStamp Message

The GetTimeStamp message gives the ability for a requester thread to read the timestamps back from the message gateway. The message consists of a single 256-bit message payload.

AbsoluteTimeLap is based on an absolute wall clock in unit of nSec/uSec that is independent of context switch or GPU frequency adjustment. Message Gateway shares the same GPU timestamp. Details can be found in the TIMESTAMP register section in *vol1c Memory Interface and Command Stream*.

RelativeTimeLap is based on a relative time count that is counting the GPU clocks for the context. The relative time count is saved/restored during context switch.

1.1.4.1 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31	**Return to High GRF**: 0: the return 128-bit data goes to the first half of the destination GRF register 1: the return 128-bit data goes to the second half of the destination GRF register
	30:8	Reserved : MBZ
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:0	Ignored
M0.1	31:0	Ignored
M0.0	31:0	Ignored

1.1.4.2 Writeback Message to Requester Thread

As the writeback message is only sent if the **AckReq** bit in the message descriptor is set, **AckReq** bit must be set for this message.

Only half of the destination GRF register is updated (via write-enables). The other half of the register is not changed. This is determined by the **Return to High GRF** control field.

Writeback Message if Return to High GRF is set to 0:

DWord	Bit	Description
W0.7:4		Reserved (not overwritten)
W0.3	31:0	**RelativeTimeLapHigh:** This field returns the MSBs of time lap for the relative clock since the previous reset. This field represents 1.024 uSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
W0.2	31:20	**RelativeTimeLapLow:** This field returns the LSBs of time lap for the relative clock since the previous reset. This field represents 1/4 nSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
	19:0	Reserved : MBZ
W0.1	31:0	**AbsoluteTimeLapHigh:** This field returns the MSBs of time lap for the absolute clock since the previous reset. This field represents 1.024 uSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12

DWord	Bit	Description
W0.0	31:20	**AbsoluteTimeLapLow:** This field returns the LSBs of time lap for the absolute clock since the previous reset. This field represents 1/4 nSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
	19:0	Reserved : MBZ

Writeback Message if Return to High GRF is set to 1:

DWord	Bit	Description
W0.7	31:0	**RelativeTimeLapHigh**
W0.6	31:20	**RelativeTimeLapLow**
	19:0	Reserved : MBZ
W0.5	31:0	**AbsoluteTimeLapHigh**
W0.4	31:20	**AbsoluteTimeLapLow**
	19:0	Reserved : MBZ
W0.3:0		Reserved : MBZ

1.1.5 BarrierMsg Message

The BarrierMsg message gives the ability for multiple threads to synchronize their progress. This is useful when there are data shared between threads. The message consists of a single 256-bit message payload.

Upon receiving one such message, Message Gateway increments the Barrier counter and mark the Barrier requester thread. There is no immediate response from the Message Gateway. When the counter value equates **Barrier Thread Count**, Message Gateway will send response back to all the Barrier requesters.

1.1.5.1 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:0	Ignored
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:28	Ignored
	27:24	**BarrierID**. This field indicates which one from the 16 Barrier States is updated. Format: U4 Note: this field location matches with that of R0 header.
	23:16	**Barrier.Offset.** This is the offset for the Barrier to indicate the offset from the requester's RegBase. Regbase is 0 if Bypass Gateway Control is 1. Barrier.Offset + RegBase must be in the valid GRF range, but not point to r0. Otherwise, hardware behavior is undefined.

DWord	Bit	Description
		It is in unit of 256-bit GRF register. The most significant bit of this field must be zero. Format = U8 Range = [0,127]
	15:0	Ignored
M0.1	31:0	Ignored
M0.0	31:4	Ignored

1.1.5.2 Message Payload

DWord	Bit	Description
M0.7	31:0	**Ignored**
M0.6	31:0	**Ignored**
M0.5	31:0	**Ignored**
M0.4	31:0	**Ignored**
M0.3	31:0	**Ignored**
M0.2	31	**Ignored**
	30:28	**Ignored**
	27:24	**BarrierID**. This field indicates which one from the 16 Barrier States is updated. Format: U4 Note: this field location matches with that of R0 header.
	23:16	**Ignored**
	15	**Barrier Count Enable**: Allows the message to reprogram the barrier count. If set, the current value of the barrier state is compared to the Barrier Count field (below). If these values are equal, the barrier is considered satisfied, barrier responses are sent to the waiting thread(s) including the sending thread, and the barrier state is reset to 0. If these values are not equal, the barrier state is incremented and the sending thread is added to the list of threads waiting on this barrier. If clear, the Message Gateway increments the Barrier counter and marks the Barrier requester thread. There is no immediate response from the Gateway. When the counter value equates Barrier Thread Count, Gateway will send response back to all the Barrier requesters. Format: Enable
	14:9	**Barrier Count**: If Barrier Count Enable is set, this field specifies the terminating barrier count. Otherwise this field is ignored. All threads that belong to a single barrier must deliver the same value for this field for a particular barrier iteration.
	8:0	**Ignored**
M0.1	31:0	**Ignored**
M0.0	31:4	**Ignored**

1.1.5.3 Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 001: *Error (Barrier is inactive).* Other encodings are reserved.

1.1.5.4 Broadcast Writeback Message

When the count for a Barrier reaches Barrier.Count, the Message Gateway sends the notification bit N0 to each EU/Thread that reached the barrier. A Barrier Return Byte is not sent.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:16	Reserved (not overwritten)
	15:8	**Reserved** (not overwritten)
	7:0	Reserved (not overwritten)

1.1.6 MMIOReadWrite Message

1.1.6.1 Message Payload

DWord	Bit	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:0	Ignored
M0.4	31:0	Ignored
M0.3	31:1	Ignored
	0	**MMIO R/W:** 0 – MMIO Read – a response will be sent to the EU with read data 1 – MMIO Write – no response is sent to EU (unless acknowledge requested in sideband)
M0.2	31:28	Ignored
	22:2	**MMIO Address:** The MMIO DWord address to be accessed.
	1:0	Ignored
M0.1	31:0	Ignored
M0.0	31:0	MMIO Write Data (Only if MMIO R/W = 1, otherwise ignored).

1.1.6.2 Writeback Message to Requester Thread (MMIO Read Only)

DWord	Bit	Description
R0.7	31:0	Ignored
R0.6	31:0	Ignored
R0.5	31:0	Ignored
R0.4	31:0	Ignored
R0.3	31:0	Ignored
R0.2	31:0	Ignored
R0.1	31:0	Ignored
R0.0	31:0	**MMIO Read Data**

2. Shared Functions - Unified Return Buffer (URB)

The Unified Return Buffer (URB) is a general-purpose buffer used for sending data between different threads, and, in some cases, between threads and fixed-function units (or vice-versa). A thread accesses the URB by sending messages.

2.1 URB Size

A URB entry is a logical entity within the URB, referenced by an entry handle and comprised of some number of consecutive rows. A row corresponds in size to a 256-bit EU GRF register. Read/write access to the URB is generally supported on a row-granular basis.

URB Size	URB Rows	URB Rows when SLM Enabled
128k	4096	2048
256k	8096	4096

2.2 URB Access

The URB can be written by the following agents:

- Command Stream (CS) can write constant data into Constant URB Entries (CURBEs) as a result of processing CONSTANT_BUFFER commands.
- The Video Front End (VFE) fixed-function unit of the Media pipeline can write thread payload data in to its URB entries.
- The Vertex Fetch (VF) fixed-function unit of the 3D pipeline can write vertex data into its URB entries
- Threads can write data into URB entries via URB_WRITE messages sent to the URB shared function.

The URB can be read by the following agents:

- The Thread Dispatcher (TD) is the main source of URB reads. As a part of spawning a thread, pipeline fixed-functions provide the TD with a number of URB handles, read offsets, and lengths. The TD reads the specified data from the URB and provide that data in the thread payload pre-loaded into GRF registers.
- The Geometry Shader (GS) and Clipper (CLIP) fixed-function units of the 3D pipeline can read selected parts of URB entries to extract vertex data required by the pipeline.
- The Windower (WM) FF unit reads back depth coefficients from URB entries written by the Strip/Fan unit.

Note that the CPU can not read the URB directly.

2.3 State

The URB function is stateless, with all information required to perform a function being passed in the write message.

See URB Allocation (*Graphics Processing Engine*) for a discussion of how the URB is divided amongst the various fixed functions.

2.4 URB Messages

This section documents the global aspects of the URB messages. The actual data stored in URB entries differs for each fixed function – refer to *3D Pipeline* and the fixed-function chapters or details on 3D URB data formats and *Media* for media-specific URB data formats.

URB Handles: Unlike prior products where the URB handle contents was not specified for software use, URB handles are now specified as offsets into the URB partition in the L3 cache, in 512-bit units. Thus, kernels are now allowed to perform math operations on URB handles.

- The **End of Thread** bit in the message descriptor may be set on URB messages only in threads dispatched by the vertex shader (VS), hull shader (HS), domain shader (DS), and geometry shader (GS). The **End of Thread** bit cannot be set on URB_READ* or URB_ATOMIC* messages.

Execution Mask. The low 8 bits of the execution mask on the send instruction determines which DWords from each write data phase are written or which DWords from each read phase are written to the destination GRF register. The execution mask is ignored on URB_ATOMIC* messages, since this is a scalar operation that is always enabled.

Out-of-Bounds Accesses. Reads to addresses outside of the URB region allocated in the L3 cache return 0. Writes to addresses outside of the URB region are dropped and will not modify any URB data.

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
URB Read HWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Write HWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Read OWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Write OWORD	yes	N/A	N/A	handle + URBoffset	1, 2

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
				or handle + URBoffset + offset	
URB Atomic MOV	yes	N/A	N/A	handle + URBoffset	1
URB Atomic INC	yes	N/A	N/A	handle + URBoffset	1

"offset" is in the message payload, and is per-slot.
"handle" is the handle address in the message header.
"URBoffset" is the **Global Offset** field in the URB message descriptor.

2.4.1 Execution Mask

The Execution Mask specified in the 'send' instruction determines which DWords within each message register are read/written to the URB.

2.4.2 Message Descriptor

Bit	Description
19	Header Present This bit must be set to **one** for all URB messages.
18:17	Ignored
16	**Per Slot offset:** If clear, the slot offset fields in the header are ignored. If set the slot offset fields are added to the global offset to obtain the overall offset. **Programming Notes:** • This bit must be 0 for URB_ATOMIC_* messages.
15	**Complete** **For URB_WRITE*, URB_SIMD8_WRITE and URB_ATOMIC*:** This bit is ignored. **For URB_READ* and URB_SIMD8_READ:** If set, this signals that the thread is finished reading from the URB entry(s) referenced by the handles(s), causing the entry(s) to be deallocated. This bit is strictly control information passed to snooping FF units. The URB shared function itself does not use this bit for any purpose.
14	**Swizzle Control.** This field is used to specify which "swizzle" operation is to be performed on the write data. It indirectly specifies whether one or two handles are valid. 0: URB_NOSWIZZLE The message accesses a single URB entry (using **URB Handle 0**). 1: URB_INTERLEAVED The message accesses two URB entries. The data is interleaved such that the upper DWords (7:4) of each 256-bit unit contain data associated with **URB Handle 1**, and the lower DWords (3:0) contain data

Bit	Description
	associated with **URB Handle 0**.
13:3	**Global Offset.** This field specifies a destination offset (in 256-bit units) from the start of the URB entry(s), as referenced by **URB Handle *n***, at which the data (if any) will be written to or read from. When URB_INTERLEAVED is used, this field provides a 256-bit granular offset applied to both URB entries. If the **Per Slot Offset** bit is set, this offset is added to the per-slot offsets in the header to obtain the overall offset. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. For the URB_ATOMIC* messages, this offset is in 32-bit units instead of 256-bit units. Format = U11 Range = [0, 1023] for URB_*_HWORD messages. Range = [0, 2047] for URB_*_OWORD messages. Range = [0, 2047] for URB_ATOMIC* messages.
2:0	**URB Opcode** 0: URB_WRITE_HWORD 1: URB_WRITE_OWORD 2: URB_READ_HWORD 3: URB_READ_OWORD 4: URB_ATOMIC_MOV 5: URB_ATOMIC_INC 6-7: Reserved

2.4.3 URB_WRITE* and URB_READ*

The URB_WRITE* and URB_READ* messages share the same header definition. URB_WRITE has additional payload containing the write data, but has no writeback message. URB_READ has no payload beyond the header (message length is always one), but always has a writeback message. URB_WRITE_SIMD4x2 has a single-phase payload with the per-slot offsets followed by the write data, and has no writeback message. URB_READ_SIMD4x2 has a single phase payload containing the per-slot offsets.

2.4.3.1 Message Header

M0.5[7:0] bits in message header are used for enabling DWs in cull test, at HDC unit by HS kernel, while writing TF data using URB write messages. Cull test is performed on outside TF and HS kernel set the appropriate DW enable, which carry the TF for different domain types. When DW is enabled and if cull test is positive, HS stage will be informed by HDC unit, to cull the HS handle early at HS stage itself.

DWord	Bits	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:17	Ignored
	16	**High OWORD Enable** For URB_READ_OWORD and URB_WRITE_OWORD with NOSWIZZLE indicates whether the 128 bits of the GRF register is used. 0: 1 OWord, read into or written from the low 128 bits of the GRF register 1: 1 OWord, read into or written from the high 128 bits of the GRF register
	15	Vertex 1 DATA [3] / Vertex 0 DATA[7] Channel Mask When **Swizzle Control** = URB_INTERLEAVED this bit controls Vertex 1 DATA[3], when **Swizzle Control** = URB_NOSWIZZLE this bit controls Vertex 0 DATA[7]. This bit is ANDed with the corresponding channel enable to determine the final channel enable. For the URB_READ_OWORD & URB_READ_HWORD messages, when final channel enable is 1 it indicates that Vertex 1 DATA [3] will be included in the writeback message. For the URB_WRITE_OWORD & URB_WRITE_HWORD messages, when final channel enable is 1 it indicates that Vertex 1 DATA [3] will be written to the surface. 0: Vertex 1 DATA [3] / Vertex 0 DATA[7] channel not included 1: Vertex DATA [3] / Vertex 0 DATA[7] channel included
	14	**Vertex 1 DATA [2] Channel Mask**
	13	**Vertex 1 DATA [1] Channel Mask**
	12	**Vertex 1 DATA [0] Channel Mask**
	11	**Vertex 0 DATA [3] Channel Mask**
	10	**Vertex 0 DATA [2] Channel Mask**
	9	**Vertex 0 DATA [1] Channel Mask**

DWord	Bits	Description
	8	**Vertex 0 DATA [0] Channel Mask**
	7:0	**Reserved**
M0.4	31:0	**Slot 1 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 256-bit units) from the start of the URB entry, as referenced by **URB Handle 1**, at which the data will be accessed. This field is ignored unless **Swizzle Control** is set to URB_INTERLEAVED. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. Format = U32 Range = [0, 1023] for URB_*_HWORD messages. The range of the calculated offset must fall within the range [0, 1023] or behavior is undefined. Range = [0, 2047] for URB_*_OWORD messages. The range of the calculated offset must fall within the range [0, 2047] or behavior is undefined.
M0.3	31:0	**Slot 0 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 256-bit units) from the start of the URB entry, as referenced by **URB Handle 0**, at which the data will be accessed. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. Format = U32 Range = [0, 1023] for URB_*_HWORD messages. The range of the calculated offset must fall within the range [0, 1023] or behavior is undefined. Range = [0, 2047] for URB_*_OWORD messages. The range of the calculated offset must fall within the range [0, 2047] or behavior is undefined.
M0.2	31:16	: **GS Number of Output Vertices for Slot 1.** Indicates the number of vertices output for geometry shader slot 1 primitive. This field is only defined if end-of-thread is set on the message. It is ignored for all messages from non-GS threads. Format = U16
	15:0	: **GS Number of Output Vertices for Slot 0.** Indicates the number of vertices output for geometry shader slot 0 primitive. This field is only defined if end-of-thread is set on the message. It is ignored for all messages from non-GS threads. Format = U16

DWord	Bits	Description
M0.1	31:16	: **Handle ID 1.** This ID is assigned by the fixed function unit and links the work in channel 1 to a specific entry within the fixed function unit. This field is ignored unless **Swizzle Control** indicates Interleave mode.
	15:0	**URB Handle 1.** This is the URB handle where channel 1's results are to be written or read. This field is ignored unless **Swizzle Control** indicates interleave mode.
M0.0	31:16	**Handle ID 0.** This ID is assigned by the fixed function unit and links the work in channel 0 to a specific entry within the fixed function unit.
	15:0	**URB Handle 0.** This is the URB handle where channel 0's results are to be written or read.

2.4.3.2 URB_WRITE_HWORD Write Data Payload

For the URB_WRITE_HWORD messages, the message payload will be written to the URB entries indicated by the URB return handles in the message header.

Payload	Usage
URB_NOSWIZZLE	The message payload contains data to be written to a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The message payload contains data to be written to two separate URB entries. The payload data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

2.4.3.2.1 URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply write data into consecutive URB locations (no data swizzling applied).

Programming Notes:

- The URB function will use (not ignore) the Channel Enables associated with this message.

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing *n* pairs of 4-DWord vertex elements (where for the example, *n* is >2).

DWord	Bit	Description
M1.7	31:0	**Vertex Data [7]**
M1.6	31:0	**Vertex Data [6]**
M1.5	31:0	**Vertex Data [5]**

DWord	Bit	Description
M1.4	31:0	**Vertex Data [4]**
M1.3	31:0	**Vertex Data [3]**
M1.2	31:0	**Vertex Data [2]**
M1.1	31:0	**Vertex Data [1]**
M1.0	31:0	**Vertex Data [0]**
M2.7	31:0	**Vertex Data [15]**
M2.6	31:0	**Vertex Data [14]**
M2.5	31:0	**Vertex Data [13]**
M2.4	31:0	**Vertex Data [12]**
M2.3	31:0	**Vertex Data [11]**
M2.2	31:0	**Vertex Data [10]**
M2.1	31:0	**Vertex Data [9]**
M2.0	31:0	**Vertex Data [8]**
…		**…**
Mn.7	31:0	**Vertex Data [8(n-1)+7]**
Mn.6	31:0	**Vertex Data [8(n-1)+6]**
Mn.5	31:0	**Vertex Data [8(n-1)+5]**
Mn.4	31:0	**Vertex Data [8(n-1)+4]**
Mn.3	31:0	**Vertex Data [8(n-1)+3]**
Mn.2	31:0	**Vertex Data [8(n-1)+2]**
Mn.1	31:0	**Vertex Data [8(n-1)+1]**
Mn.0	31:0	**Vertex Data [8(n-1)+0]**

2.4.3.2.2 URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing *n* 4-DWord vertex elements (n>1).

Programming Restrictions:

- The URB function will use (not ignore) the Channel Enables associated with this message.
- Writes to overlapping addresses of vertex0 and vertex1 will have undefined write ordering.

DWord	Bit	Description
M1.7	31:0	**Vertex 1 Data [3]**
M1.6	31:0	**Vertex 1 Data [2]**
M1.5	31:0	**Vertex 1 Data [1]**
M1.4	31:0	**Vertex 1 Data [0]**
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**
M2.7	31:0	**Vertex 1 Data [7]**
M2.6	31:0	**Vertex 1 Data [6]**
M2.5	31:0	**Vertex 1 Data [5]**
M2.4	31:0	**Vertex 1 Data [4]**
M2.3	31:0	**Vertex 0 Data [7]**
M2.2	31:0	**Vertex 0 Data [6]**
M2.1	31:0	**Vertex 0 Data [5]**
M2.0	31:0	**Vertex 0 Data [4]**
…		**…**
Mn.7	31:0	**Vertex 1 Data [4(n-1)+3]**
Mn.6	31:0	**Vertex 1 Data [4(n-1)+2]**
Mn.5	31:0	**Vertex 1 Data [4(n-1)+1]**
Mn.4	31:0	**Vertex 1 Data [4(n-1)+0]**

DWord	Bit	Description
Mn.3	31:0	**Vertex 0 Data [4(n-1)+3]**
Mn.2	31:0	**Vertex 0 Data [4(n-1)+2]**
Mn.1	31:0	**Vertex 0 Data [4(n-1)+1]**
Mn.0	31:0	**Vertex 0 Data [4(n-1)+0]**

2.4.3.3 URB_READ_HWORD Writeback Message

For the URB_READ_HWORD messages, the URB entries indicated by the URB handles in the message header are read and returned in the writeback message. The amount of read data returned is determined by the **Response Length** field.

While GS threads will read one vertex at a time to the URB, the VS will read two interleaved vertices. The description of the URB read messages will refer to the per-vertex DWords described in the Vertex URB Entry Formats section of the *3D Overview* chapter.

Payload	Usage
URB_NOSWIZZLE	The writeback message contains data read from a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The writeback message contains data read from two separate URB entries. The data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

2.4.3.3.1 URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply read data into consecutive URB locations (no data interleaving applied).

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing *n* pairs of 4-DWord vertex elements (where for the example, *n* is >2).

DWord	Bit	Description
W0.7	31:0	**Vertex Data [7]**
W0.6	31:0	**Vertex Data [6]**
W0.5	31:0	**Vertex Data [5]**
W0.4	31:0	**Vertex Data [4]**
W0.3	31:0	**Vertex Data [3]**
W0.2	31:0	**Vertex Data [2]**
W0.1	31:0	**Vertex Data [1]**

DWord	Bit	Description
W0.0	31:0	**Vertex Data [0]**
W1.7	31:0	**Vertex Data [15]**
W1.6	31:0	**Vertex Data [14]**
W1.5	31:0	**Vertex Data [13]**
W1.4	31:0	**Vertex Data [12]**
W1.3	31:0	**Vertex Data [11]**
W1.2	31:0	**Vertex Data [10]**
W1.1	31:0	**Vertex Data [9]**
W1.0	31:0	**Vertex Data [8]**
…		**…**
Wn.7	31:0	**Vertex Data [8n+7]**
Wn.6	31:0	**Vertex Data [8n+6]**
Wn.5	31:0	**Vertex Data [8n+5]**
Wn.4	31:0	**Vertex Data [8n+4]**
Wn.3	31:0	**Vertex Data [8n+3]**
Wn.2	31:0	**Vertex Data [8n+2]**
Wn.1	31:0	**Vertex Data [8n+1]**
Wn.0	31:0	**Vertex Data [8n+0]**

2.4.3.3.2 URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing *n* 4-DWord vertex elements (n>1).

DWord	Bit	Description
W0.7	31:0	**Vertex 1 Data [3]**
W0.6	31:0	**Vertex 1 Data [2]**
W0.5	31:0	**Vertex 1 Data [1]**
W0.4	31:0	**Vertex 1 Data [0]**
W0.3	31:0	**Vertex 0 Data [3]**

DWord	Bit	Description
W0.2	31:0	**Vertex 0 Data [2]**
W0.1	31:0	**Vertex 0 Data [1]**
W0.0	31:0	**Vertex 0 Data [0]**
W1.7	31:0	**Vertex 1 Data [7]**
W1.6	31:0	**Vertex 1 Data [6]**
W1.5	31:0	**Vertex 1 Data [5]**
W1.4	31:0	**Vertex 1 Data [4]**
W1.3	31:0	**Vertex 0 Data [7]**
W1.2	31:0	**Vertex 0 Data [6]**
W1.1	31:0	**Vertex 0 Data [5]**
W1.0	31:0	**Vertex 0 Data [4]**
…		**…**
Wn.7	31:0	**Vertex 1 Data [4n+3]**
Wn.6	31:0	**Vertex 1 Data [4n+2]**
Wn.5	31:0	**Vertex 1 Data [4n+1]**
Wn.4	31:0	**Vertex 1 Data [4n+0]**
Wn.3	31:0	**Vertex 0 Data [4n+3]**
Wn.2	31:0	**Vertex 0 Data [4n+2]**
Wn.1	31:0	**Vertex 0 Data [4n+1]**
Wn.0	31:0	**Vertex 0 Data [4n+0]**

2.4.3.4 URB_WRITE_OWORD Write Data Payload

For the URB_WRITE_OWORD messages, the message payload will be written to the URB entries indicated by the URB return handles in the message header.

Payload	Usage
URB_NOSWIZZLE	The message payload contains data to be written to a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The message payload contains data to be written to two separate URB entries.

Payload	Usage
	The payload data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

2.4.3.4.1 URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply write data into a single 128-bit URB location (no data swizzling applied).

Programming Notes:

- The URB function will use (not ignore) the Channel Enables associated with this message.

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 0.

DWord	Bit	Description
M1.7	31:0	Ignored
M1.6	31:0	Ignored
M1.5	31:0	Ignored
M1.4	31:0	Ignored
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 1.

DWord	Bit	Description
M1.7	31:0	**Vertex 0 Data [3]**
M1.6	31:0	**Vertex 0 Data [2]**
M1.5	31:0	**Vertex 0 Data [1]**
M1.4	31:0	**Vertex 0 Data [0]**
M1.3	31:0	Ignored
M1.2	31:0	Ignored
M1.1	31:0	Ignored
M1.0	31:0	Ignored

2.4.3.4.2 URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing 4-DWord vertex elements.

Programming Restrictions:

- The URB function will use (not ignore) the Channel Enables associated with this message.
- Writes to overlapping addresses of vertex0 and vertex1 will have undefined write ordering.

DWord	Bit	Description
M1.7	31:0	**Vertex 1 Data [3]**
M1.6	31:0	**Vertex 1 Data [2]**
M1.5	31:0	**Vertex 1 Data [1]**
M1.4	31:0	**Vertex 1 Data [0]**
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**

2.4.3.5 URB_READ_OWORD Writeback Message

For the URB_READ_HWORD messages, the URB entries indicated by the URB handles in the message header are read and returned in the writeback message. The amount of read data returned is determined by the **Response Length** field.

Programming Restrictions:

- **Response Length** must be set to 1.

While GS threads will read one vertex at a time to the URB, the VS will read two interleaved vertices. The description of the URB read messages will refer to the per-vertex DWords described in the Vertex URB Entry Formats section of the *3D Overview* chapter.

Payload	Usage
URB_NOSWIZZLE	The writeback message contains data read from a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The writeback message contains data read from two separate URB entries. The data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

2.4.3.5.1 URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply read data into consecutive URB locations (no data interleaving applied).

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 0.

DWord	Bit	Description
W0.7	31:0	Reserved (not written to GRF)
W0.6	31:0	Reserved (not written to GRF)
W0.5	31:0	Reserved (not written to GRF)
W0.4	31:0	Reserved (not written to GRF)
W0.3	31:0	**Vertex Data [3]**
W0.2	31:0	**Vertex Data [2]**
W0.1	31:0	**Vertex Data [1]**
W0.0	31:0	**Vertex Data [0]**

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 1.

DWord	Bit	Description
W0.7	31:0	**Vertex Data [3]**
W0.6	31:0	**Vertex Data [2]**
W0.5	31:0	**Vertex Data [1]**
W0.4	31:0	**Vertex Data [0]**
W0.3	31:0	Reserved (not written to GRF)
W0.2	31:0	Reserved (not written to GRF)
W0.1	31:0	Reserved (not written to GRF)
W0.0	31:0	Reserved (not written to GRF)

2.4.3.5.2 URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing 4-DWord vertex elements.

DWord	Bit	Description
W0.7	31:0	**Vertex 1 Data [3]**
W0.6	31:0	**Vertex 1 Data [2]**
W0.5	31:0	**Vertex 1 Data [1]**
W0.4	31:0	**Vertex 1 Data [0]**

DWord	Bit	Description
W0.3	31:0	**Vertex 0 Data [3]**
W0.2	31:0	**Vertex 0 Data [2]**
W0.1	31:0	**Vertex 0 Data [1]**
W0.0	31:0	**Vertex 0 Data [0]**

2.4.4 URB_ATOMIC*

The URB_ATOMIC messages implement atomic operations on a single DWord in the URB. The location of the DWord within the URB is specified by the single URB handle and the **Global Offset** field in the message descriptor, which for these messages is a DWord offset from the URB handle. The DWord selected will be operated on according to the following table:

URB Opcode	new_dst	ret
URB_ATOMIC_MOV	src0	none
URB_ATOMIC_INC	old_dst + 1	old_dst

The previous contents of the DWord are returned in the destination register for the URB_ATOMIC_INC. The URB_ATOMIC_MOV opcode does not return data (response length must be zero).

The URB_ATOMIC* messages consist only of the header. A single URB handle is specified.

2.4.4.1 Message Header

DWord	Bit	Description
M0.7	31:0	
M0.6	31:0	
M0.5	31:0	Ignored
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:0	**Source0 Data** Specifies the source 0 data for the atomic operation. This field is ignored for the URB_ATOMIC_INC message. Format = U32
M0.1	31:0	Ignored
M0.0	31:16	Ignored
	15:0	**URB Handle.** This specifies the URB handle to be accessed.

2.4.4.2 Writeback Message

A writeback message is only returned for the URB_ATOMIC_INC message. Only the low 32 bits of the destination GRF register are overwritten with the return data.

DWord	Bit	Description
W0.7:1		Reserved (not written to GRF)
W0.0	31:0	**Return Data** Specifies the value of the return data for the atomic operation. Format = U32

Here is a description of the 3DMark11 usage model (from their func spec):

4.4.1 Depth of Field

The effect is computed using the following procedure:

1. Circle of confusion radius is computed for all screen pixels and stored in a full resolution DXGI_FORMAT_R16_FLOAT texture.

2. Half and quarter resolution versions are made from the radius texture and the original illumination texture.

3. Positions of out-of-focus pixels whose circle of confusion radius exceeds a predefined threshold are appended to a buffer.

4. The position buffer is used as point primitive vertex data and, utilizing Geometry Shader (GS), image of hexagon-shaped bokeh is splatted to positions of these vertices. Splatting is done to a DXGI_FORMAT_R16G16B16A16_FLOAT texture. Multiple viewports are used to partition the texture to regions with different sizes. First region is screen size and the rest are a series of halved regions down to size 1x1 texels. The radius of the splatted bokeh determines the used viewport. The larger the radius the smaller the used viewport.

5. Steps 3 and 4 are done separately for full, half, and quarter resolution image data with different radius thresholds. Larger bokehs are generated from lower resolution image data.

6. The different regions of the splatting texture are combined by up-scaling the data in the smaller regions to the screen size region.

7. The combined splatted out-of-focus illumination is combined with the original illumination.

3. Shared Functions – Video Motion Estimation

The Video Motion Estimation (VME) engine is a shared function that provides motion estimation services. It includes motion estimation for various block sizes and also standard specific operations such as

- Motion estimation and mode decision for AVC
- Intra prediction and mode decision for AVC
- Motion estimation and mode decision for MPEG2
- Motion estimation and mode decision for VC1

The motion estimation engine may also be used for other coding standards or other video processing applications.

3.1 Theory of Operation

VME performs a sequence of operations to find the best mode for a given macroblock. Each operation step can be enabled/disabled through the control of the income message. Early termination, skipping of subsequent operation steps, is also supported when certain search criteria are met.

VME contains the following operation steps:

1. Skip check
2. IME: Integer motion estimation
3. FME: Fractional motion estimation
4. BME: Bidirectional motion estimation
5. IPE: Intra prediction estimation (AVC only)

3.1.1 Shape Decision

As a terminology, we call sub-block shapes: 8x4, 4x8, and 4x4 minor shapes (corresponding to sub-partitions of 8x8 sub-macroblock), and 16x16, 16x8, 8x16, and 8x8 major shapes (corresponding to sub-macroblocks of a 16x16 macroblock)**.**

If the maximal allowed number of motion vectors **MaxNumMVs** (**MaxNumMVs** = **MaxNumMVsMinusOne** + 1) is less than 4, we will set minor MV flag off: **MinorMVsFlag** = 0, *i.e.* no minor motion vectors will be generated.

The reason of having this parameter **MaxNumMVs** is due to high level AVC conformance restrictions for certain profiles: *the total number of motion vectors of any two consecutive macroblocks not exceeding 16 (or 32)*. The mechanism here allows a reasonable degree of user control. In disable cases, **MaxNumMVs** should be set to 32.

In the coding process of VME, the shape decision is done in multiple locations:

1) After IME and before FME, intermediate shape decision is performed to reduce the FME searching candidates

2) After FME and before BME, existing shape decision is revised among the remaining candidates and to see if there is further reduction.

3) Final shape decision is done after BME.

Partition decision before BME uses unidirectional motion vector count to meet **MaxNumMVs** requirement. Adding BME for the partition candidates may exceed **MaxNumMVs**. As BME is performed on a block by block basis using the block order for a given partition, BME step for a given block is skipped and the best unidirectional motion vectors are used for the block if the overall motion vector count exceeds **MaxNumMVs** when that particular block is switched to bidirectional. The process continues to the last block of the partition.

Note: This is a sub-optimal solution to simplify the hardware implementation. For some cases, bidirectional modes with larger sub-partitions might be better than unidirectional modes with finer sub-partitions.

The VME implementation has the following restriction: Multiple partition candidates are only enabled if **PartCandidateEn** is set. And this only applies to source block of size 16x16.

If **PartCandidateEn** is not set, only the best partition is kept in state 1 (after IME) above and carried through FME and BME. In other words, FME if enabled only operates on one partition candidate, and BME if enabled only operates on one partition candidate. Bidirectional mode check only applies to the partition candidates that meet the bidirectional restriction provided by **BiSubMbPartMask**. For example, if a minor partition determined based on best unidirectional cost function is not 8x8 but one of 4x8, 8x4 or 4x4, VME skips the bidirectional mode check.

If **PartCandidateEn** is set, up to two sets of candidates are maintained by VME hardware, if the second best partition candidate is within **PartToleranceThrhd** from the best one. The second best partition is selected only from the two major partition candidates based on the unidirectional motion vector count, subject to that the major partition is enabled:

- 1MV: The 16x16 partition
- 4MV: The 4x(8x8) partition with no minor shape

The following partitions are not supported as alternative partition.

- 2MV: The best of 2x(16x8) and 2x(8x16) partitions
- More than 4MV: The best of all 4x(8x8) partitions with at least one 8x8 having minor shape of 8x4, 4x8 or 4x4

3.1.1.1 Minor Shape Decision Prior to FME

If any minor shapes are selected, we decide the best minor first.

For each 8x8 sub-block, before performing bidirectional, we reduce code candidates to no more than three based on the best unidirectional motion search results (best of the forward and backward):

0) One MV, *i.e.* the best in shape of 8x8.

1) Up to two MVs, *i.e.* the best in shapes 8x8, 8x4, or 4x8. And

2) Up to four MVs, *i.e.* the best for the sub-block 8x8.

Now for the first and the second sub-blocks, we can merge them into up to six candidates of 2, 3, 4, 5, 6, and 8 possible motion vectors.

Do the same to the third and the fourth sub-blocks, we have similarly up to six candidates.

Now we further combine these two groups, and find the best solution under the constraint of not exceeding the number of motion vectors more than **MaxNumMVs**. (see pseudo-code below for detail.)

Consequently, we have the best combined 8x8 solutions with **N** motion vectors for some **N** less or equal to **MaxNumMVs**.

Assume `distA[k][s]` is the cost-adjusted distortion of the best forward or backward motion vector mix of the *k*-th 8x8 sub-block of the sub-shape *s*, where *s*=0,1,2, and 3 represent shape partitioning 8x8, 8x4, 4x8, and 4x4 respectively. Assume `distA[k][s]` is the bidirectional one of the corresponding bus-block and sub-shape. And assume some large number, say 128x16=2048 is assigned to the variable, if there were no valid corresponding codes. Hence, the following pseudo-code explains the code selection algorithm.

Let's first explain the case **MaxNumMVs** is disbled, *i.e.***MaxNumMVs**≥16:

```
void SelectBestCombinedMinors(
short *distA,
short*MinorShape,
short*MinorDisto
){
shorts[4], d[4];
    s = ShapeList;
    d = DistoList;
    for(int k=0;k<4;k++){
s[k] = 0;d[k] = distA[k][0];
if(distA[k][1])<d[k]){ d[k]=distA[k][1]; s[k]=1; }
if(distA[k][2])<d[k]){ d[k]=distA[k][3]; s[k]=2; }
if(distA[k][3])<d[k]){ d[k]=distA[k][3]; s[k]=3; }
}
    *MinorDisto = d[0]+d[1]+d[2]+d[3];
*MinorShape = s[0]|(s[1]<<2)|(s[2]<<4)|({s[3]<<6};
}
```

Now for the case of using **MaxNumMVs** control:

```
void SelectBestCombinedMinors(
short *distA,
intMaxNumMVs,
short*MinorShape,
short*MinorDisto
){
int k, n;
short dist, best0 = 0, best1 = 0;
    if(MaxNumMVs< 4){ // we reset other parameters
switch(MaxNumMvs){
case 0:
   DoIntraInter &= (~DO_INTER); // not do Inter
   break;
case 1:
   ShapeMask |= (NO_16X8|NO_8X16);
   BidirMask |= NO_16X16;
case 2: case 3:
   ShapeMask |= (NO_8X8|NO_8X4|NO_4X8|NO_4X4);
```

```
   BidirMask |= (NO_16X8|NO_8X16);
   break;
}
    }
    if(MaxNumMVs>=16){ // it should use unrestricted code selection
SelectBestCombinedMinors(DistA,MinorShape,MinorDisto);
return;
    }
short*s, ShapeList[18];
short*d, DistoList[18];
    s = ShapeList;
    d = DistoList;
    for(k=0;k<4;k++){
s[0] = 0;// 1 mv
d[0] = distA[k][0];
s[4] = (distA[k][2]<distA[k][1])+1; // 2 mvs
d[4] = distA[k][s[1]];
s[8] = 3;// 4 mvs
d[8] = distA[k][3];
s ++, d ++;
    }
    // Merge two:
    s = ShapeList;
    d = DistoList;
    for(k=0;k<2;k++){
s[16]= 0x33; // 8 mvs
d[16]= d[8]+d[10];
s[12]= (d[4]+d[10]<d[6]+d[8])?(s[4]|0x30):(0x03|(s[6]<<4));// 6 mvs
d[12]= (d[4]+d[10]<d[6]+d[8])?(d[4]+d[10])<(d[6]+d[8]);
s[10]= (d[0]+d[10]<d[8]+d[2])?0x30:0x03; // 5 mvs
d[10]= (d[0]+d[10]<d[8]+d[2])?(d[0]+d[10])<(d[8]+d[2]);
s[8] = s[4]|(s[6]<<4);// 4 mvs
d[8] = d[4]+d[6];
s[6] = (d[4]+d[2]<d[0]+d[6])?s[4]:(s[6]<<4);    // 3 mvs
d[6] = (d[4]+d[2]<d[0]+d[6])?(d[4]+d[2])<(d[0]+d[6]);
s[4] = 0; // 2 mvs
d[4] = d[0]+d[2];
if(d[ 6]>d[ 4]) d[ 6] = d[ 4];
if(d[ 8]>d[ 6]) d[ 8] = d[ 6];
if(d[10]>d[ 8]) d[10] = d[ 8];
if(d[12]>d[10]) d[12] = d[10];
d[14] = d[12];
if(d[16]>d[12]) d[16] = d[12];
s ++; d ++;
    }
```

```
    s = ShapeList;
    d = DistoList;
*MinorDisto = 2048;
    for(k=0;k<8;k++){
n = MaxNumMVs - k;
if(n>=2 && n<=8)<2){
dist = d[(k<<1)+1]+d[n<<1];
if(dist<*MinorDisto){
*MinorDisto = dist;
best0 = (n<<1);
best1 = (k<<1)+1;
}
}
    }
    while(best0>1 && d[best0]==d[best0-2]) best0 -= 2;
    while(best1>1 && d[best1]==d[best1-2]) best1 -= 2;
    *MinorShape = s[best0]|((s[best1]<<2);
}
```

3.1.1.2 Major Shape Decision Prior to FME

Now considering the best of each 8x8 is done, and we have the total cost-adjusted-distortion for this sub-block level partition. Now among the four choices: the resulting 8x8 sub-partitioning, one 16x16, two 16x8, and two 8x16, the one gives the best cost-adjusted-distortion, will determine the final decision of partitioning shape. Any among these four, if its cost-adjusted-distortion is within the intermediate tolerance (which is a predefined system state) from the best distortion will be marked as **candidate shape**s.

Notice that, when the intermediate tolerance is set to 0, only the best shape will be selected as the candidate. When the intermediate tolerance is large, all four shapes will become candidates.

Assume we have all the distortions for majors enumerated in `DistoMajor[k]`, where `k` = 0, 1, 2, 3, 4, and 5, for 16x16, 16x8, 8x16, the combined minors, 16x8 field, and 8x8 field respectively. Assume `BestDisto` is equal to the minimal of the six values `DistoMajor[k]`, for `k` = 0, …5. Assume the intermediate tolerance is `IntTol`, the major shape `k` is a candidate shape if and only if DistoMajor[k]<=BestDisto+IntTol.

3.1.1.3 Shape Update after FME

Among all the candidate shapes, we recheck the distortion, if any of them is no longer with in the intermediate tolerance **DistortionTolerance** from the best choice; we drop it for reduced calculation.

3.1.1.4 Final Code Decision after BME

For any given candidate shape, for each motion vector, if we do have improved distortion by switch from the single direction to bi-direction, then we do it, unless the increased number of motion vectors hits above **MaxNumMVs**; in this case, we take as many as possible first the ones generate the most improvement.

Then, we choose the best among the improved candidate shapes.

3.1.2 Integer Motion Estimation

IME, the integer motion estimation, is the most key part of VME. In our current design, the minimal functional block is to do a full search over a search unit. This functional block is then called via two distinctive methods:

1) Via a predefined searching path of search units.

2) Via a dynamic process based on the previous results.

This section will describe both.

3.1.2.1 Reference Window and Search Units

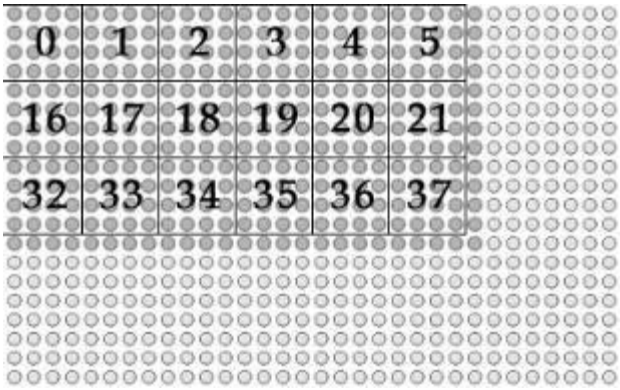
The reference window is a rectangular region fetched put in the reference cache for VME. Either one or two reference windows are allowed to be loaded into the reference cache. In the case of dual windows, both windows follow a common search path or different paths (relative to their corresponding Start Center) depending on the dual search path option flag.

The total reference cache is limited to 2K bytes and only the luma component searching is performed. For example, we may select the reference windows to be one of the following sample choices: one 64x32 area, one 48x40 area, two 40x24 areas, or two 32x32 areas, where the possible reference address will cover an area of 48x16, 32x24, 24x8, and 16x16.

As a convention, we will call the valid reference addressing region the reference region, and its width and height are called the reference window width and height respectively. So the reference loading region of VME has therefore 16 more columns and 16 more rows.

It is not efficient for hardware to search one location at a time due to reference cache access bandwidth and latency constraints. Thus, possible reference search locations are grouped in a predefined pattern, and all locations within the same group must be either all are chosen or all are skipped. These predefined groups are called *search unit* ***(SU).*** *Reference Window and Search Units* ***shows a sample of grouping search locations into searching units. The reference window in the figure has a dimension of*** 32x20, and assuming the source block is 8x8, the dark dots indicate all legitimate reference locations for motion searching. It shows a partitioning of SUs of 16 locations.

In general, the indices of SUs are given by counting rows and units within the row.



Example of Search Units in a reference window

Given a fixed reference cache access latency, SU size is determined solely based on the source block size as shown in *Reference Window and Search Units*. Note that SU sizes for both 16x8 and 8x16 source blocks are both 8x4, which gives a preference for motions along horizontal direction.

Determination of Search Unit size based on Source Block dimension

Source Block Dimension	Search Unit (SU) Size (X x Y)	GT Support
16x16	4x4	Y
16x8	8x4	Y
8x16	8x4	N
8x8	8x8	Y

To keep tracking on whether a SU has been searched or not, an equivalent hardware process is implemented performing as a search record that marks whether any search units being searched is a bit-plan of the bit-length equal to the maximal index of SUs. Before searching starts, the search record must be reset which sets the value 0 (= yet-to-be-searched) to all legitimate SU indices, and the value 1 (= no longer available) for other SUs that are not intended to be searched.

Given a search window, unique indices are assigned to all SUs. A search path (SP), is a sequence of such indices. The number of SUs in a SP is called the length of the walker (denoted by LenSP here), which shall be a number more than one. Instead of storing the absolute indices of a search path, relative *search unit deltas* ***are sent instead. In the current VME a search unit delta is a 8 bit index consisting of a pair of 4-bit signed integers in [-8,8).***

Given start center in a pair of 4-bit unsigned integer (*sx, sy****), and denote a search path described in SU deltas (****dx[i],dy[i]****). The first search unit SU[0] will be the search unit which has the first reference address (****sx*4, sy*4) in integer-pel relative to the reference origin.,*** *i.e.*

SU[0].*x* **=** *sx*****4, and SU[0].****y* **=** *sy*****4.***

The second search unit SU[1] is derived by adding (*dx[0],dy[0]****):***

SU[1].*x* **= SU[0].***x* + *dx****[0]*4, and SU[1].****y* **= SU[0].y +** *dy****[0]*4.***

In general, we have:

SU[i+1].*x* **= SU[i].***x* + *dx****[i]*4, and SU[i+1].****y* **= SU[i].y +** *dy****[i]*4.***

When SU[i] is out of range, it is either always skipped or always wrapped depending on the SU wrapping flag.

When the SU wrapping flag is on, it is equivalent to as we perform

SU[i].*x* **= SU[i].***x****(mod ref_win_width), and***

SU[i].*y* **= SU[i].***y****(mod ref_win_height),***

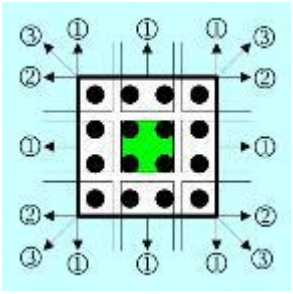
As a convention, a NULL delta marks the end of the search path.

3.1.2.2 Fixed and Adaptive Search Paths

A fixed pattern motion search algorithm is an algorithm following some predefined SP with the designated **MaxNumSU** (*maximal number of search units*) less or equal to **LenSP** (*the fixed search path length*). This is referred to as fixed pattern searching or predetermined searching.

When **MaxNumSU** > **LenSP**, (the maximal number of SU is more than what are given by the SP), the searching continues unless reaching a local minimum, which is called dynamic searching or adaptive searching or gradient searching. In this case, the current best result is used. If it is located in some SU boundary, the neighborhood SUs are checked and any one of them that is yet-to-be-searched will be the next SU to be searched. If all neighbor SU's are done, the process of IME is done. *Fixed and Adaptive Search paths* illustrates on how neighbor SUs are defined for dynamic search.

Hardware maintains one scoreboard per reference to keep track of the state of SUs, whether being searched or yet-to-be-searched. When dual records are enabled on a single reference, both records will share the same scoreboard for the reference.

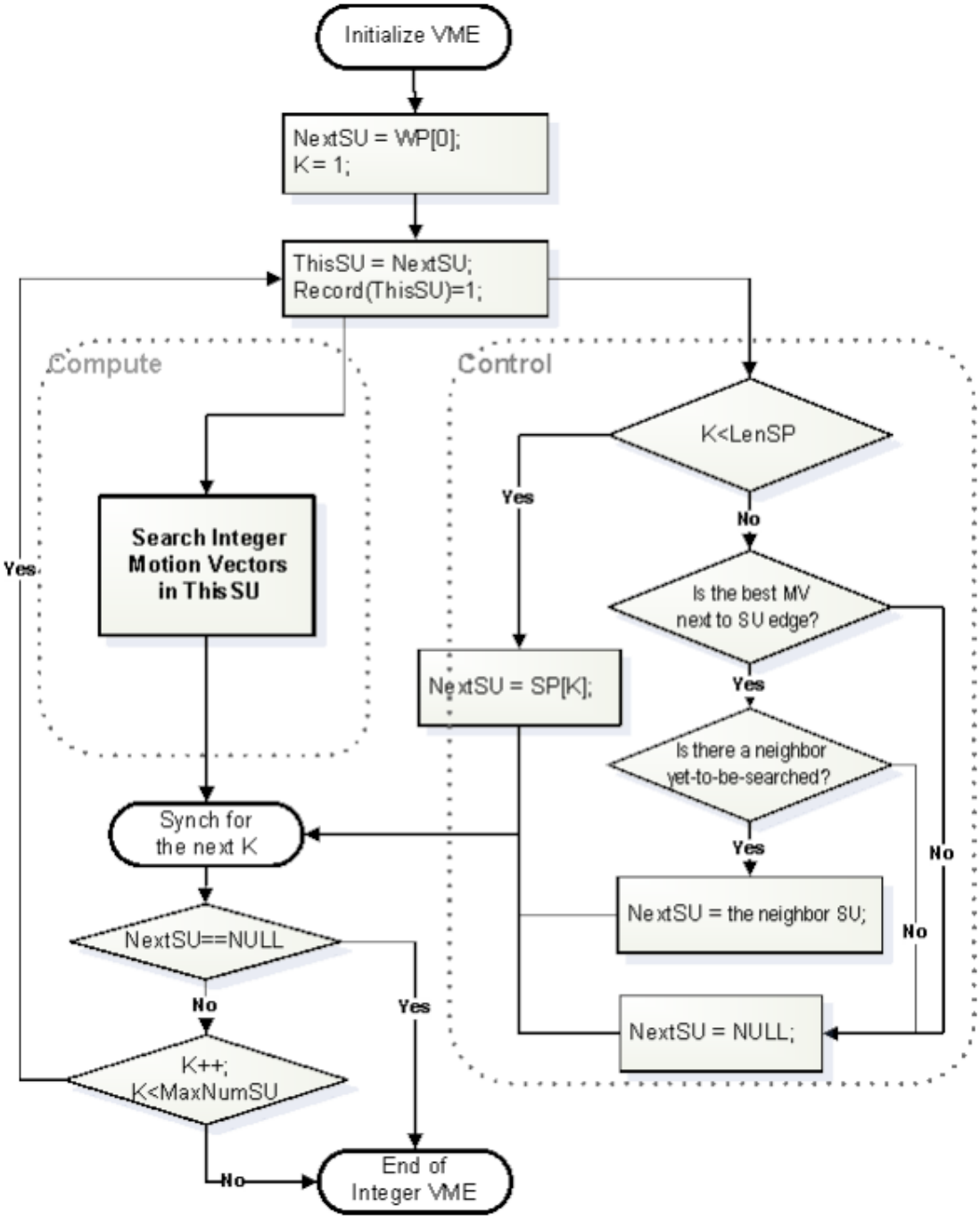


Sample neighborhood SUs in a dynamic search

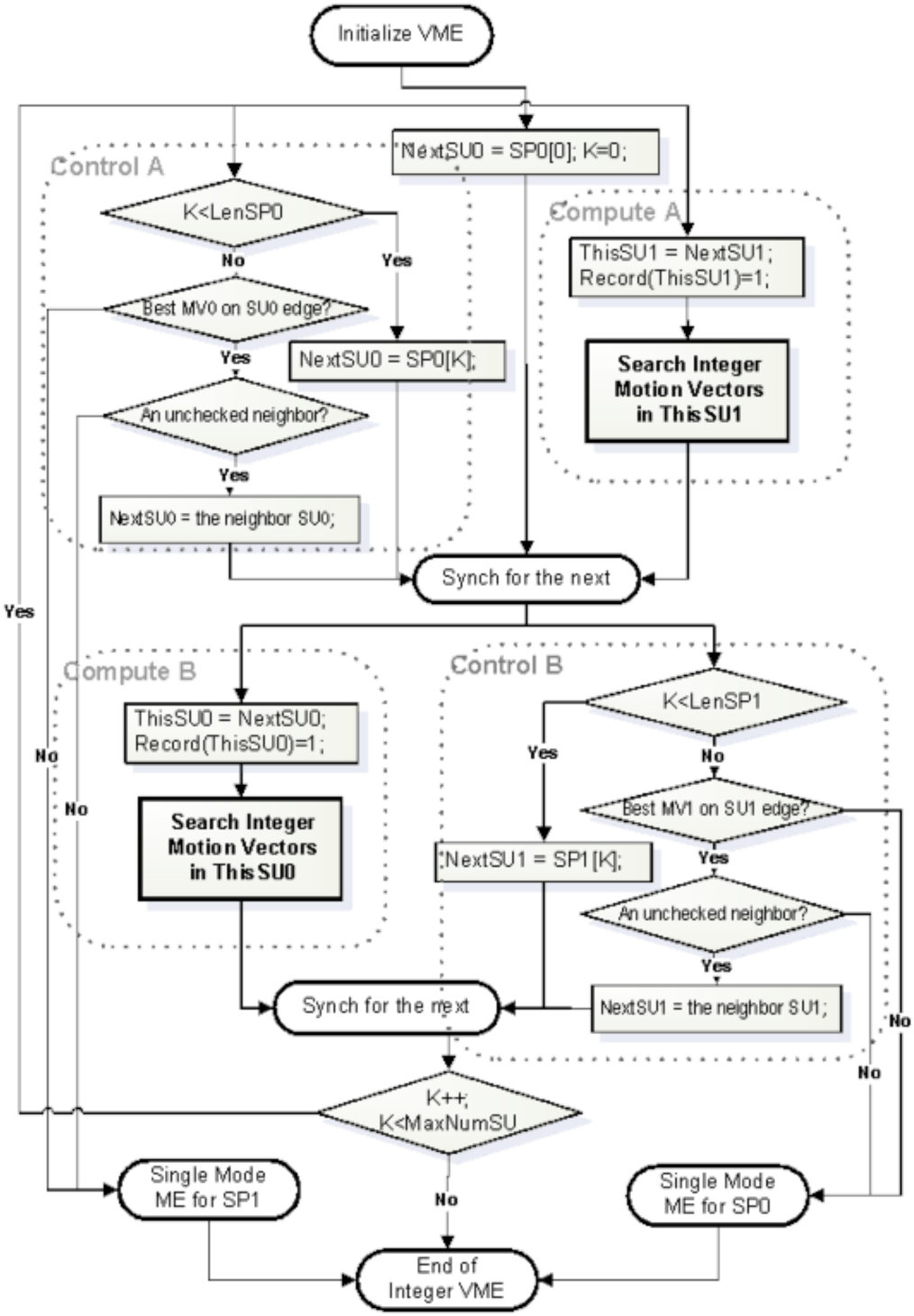
Fixed and Adaptive Search paths shows the algorithm of this integrated solution. In order to hide the decision logic of dynamic walking, the one-step-delayed-queue is implemented. So when searching the current SU, the next SU will be put in the queue. If there are more SUs yet to be searched in the current SP, the next SU will be the next SU according to SP; if there is no more SU from SP, the first unsearched neighbor SU (in some predefined order) based on the current best result will be put instead, and if there is no more unsearched neighbor SU, the integer searching terminates.

To reduce the one-step-delay, and to support bidirectional, we create the dual mode that allows the above algorithm to be ping-pong-ed between two search paths. *Fixed and Adaptive Search paths* illustrates this case.

In both figures, the best MVs refer to the best resulting motion vectors so far. There are potentially total 41 motion vectors (1 for 16x16, 2 for 16x8, 2 for 8x16, 4 for 8x8, and 32 more for 8x4, 4x8 and 4x4 cases). **The current hardware implementation only considers the four 8x8 MVs.**



VME in single SP mode



VME in Dual SP mode

3.1.3 Fractional Motion Estimation

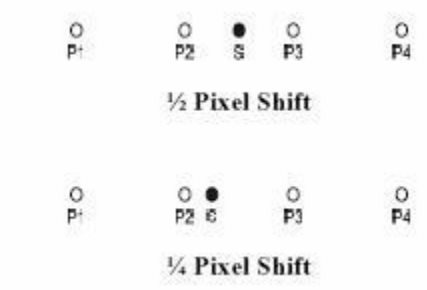
Instead of following the exact interpolation as specified by the individual video standards, 4 tap interpolation is used for the Fractional Motion Estimation (FME) step in the VME engine. It is expected to be adjusted according to different standards.

3.1.3.1 Interpolations

Instead of following the exact interpolation as specified by the individual video standards, fixed 4 tap interpolation is used in the VME engine, as defined below:

1) (-1,5,5,-1)/8 for ½-pel, *i.e.* s = (–P1+P2*5+P3*5–P4+4)/8 and

2) (-1,13,5,-1)/16 for ¼-pel position, *i.e.* c = (–P1+P2*13+P3*5–P4+8)/16.



Fractional pixel locations

The quarter-pels are actually the averages of its nearest integer and half pixel values.

It is not hard to see our suggested interpolation formulas are very much the good approximations of the formulas from various standards

For AVC, they should be the following 6-tap formulas in theory:

1) (1, -5,20,20,-5,1)/32 for ½-pel, *i.e.* s = ((P2+P3)*20–(P1+P4)*5+(F0+F6), and

2) (1,-5,52,20-5,-1)/64 for ¼-pel position.. n.

For VC-1, the 4-tap filters are precisely defined:

1) (-1,9,9,-1)/16 for ½-pel, and

2) (-4,53,18,-3)/64 for ¼-pel position.

In general, bilinear interpolation is accepted too:

3) (0,1,1,0)/2 for ½-pel (as used in MPEG2), and

4) (0,3,1,0)/4 for ¼-pel position.

After IME is done, if the best Inter result is too bad, we may decide to stop the Inter-search to not waste the effort further computationally. If we decided to continue, we have the option to decide shape first to cut down FME calculation or to perform FME for all possible configurations.

VME performs the sub-block level intermediate shape decision first (see *Shape Decision* section for detail), then perform FME only for the reduced candidate shapes. In this way, the computation is reduced significantly with tunable small quality hit.

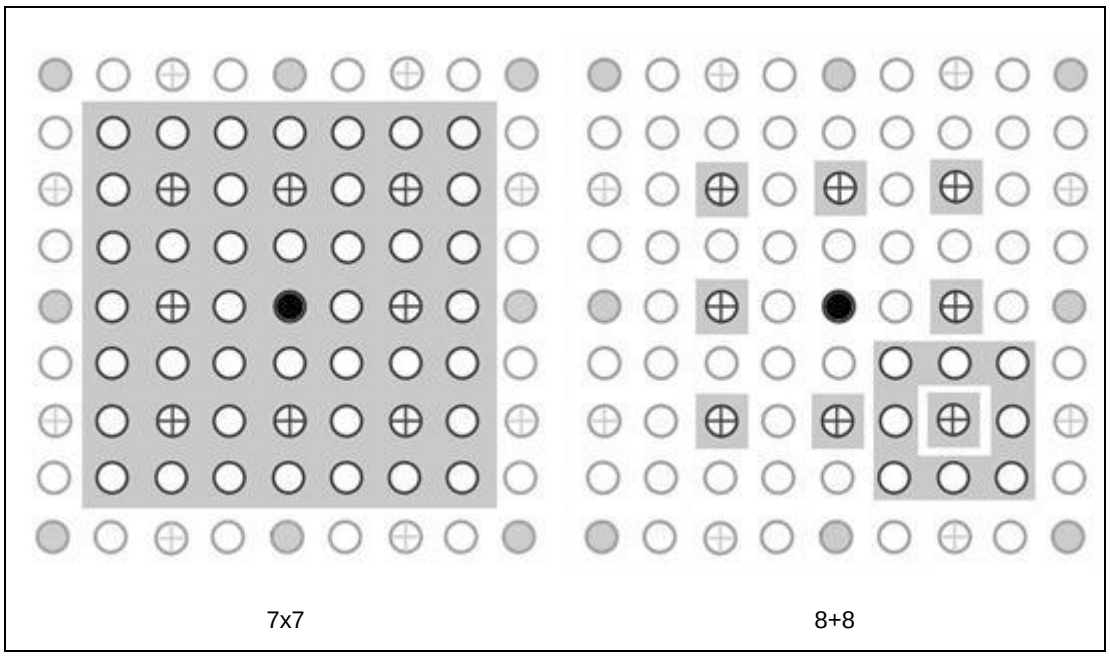
3.1.3.2 8+8 vs. 7x7

With a given sub-block of motion vector search, we also have multiple options to pursue the searching. Name two common extremes: 7x7 and 8+8.

Given an integer motion vector location, surrounding it there are 48 surrounding quarter-pel locations, and among them there are 8 are in half-pel grids. So we may check all 48, which covers the 7x7 region, for the best, or we may adopt a two-step approach by considering the half-pel grids first then followed by the second step of the quarter-pel refinement.

The one step method is named as 7x7, and the two step method called 8+8 as only 16 block comparisons are performed as shown in the next figure.

VME hardware follows the 8+8 approach.



7x7 vs. 8+8, whereas the 8+8 method is used by VME

3.1.3.3 Partitioning Refinement

When the partitioning refinement is enabled, the FME refinement results will be propagated to or sub-blocks as well, and a shape partitioning will be redone after the completion of both half-pel and quarter-pel searching for a possible better choice.

In the case when alternative candidate is enabled, both half-pel refinements are done in parallel, and then records are combined. Then, both quarter-pel refinements are done again in parallel, and combined again prior to the final repartitioning. In HW implementation, we do the coarser on first, and the finer one later to achieve the above equivalence.

3.1.4 BME and Weighted Prediction

Bidirectional searching is performed to all candidate shapes.

A weighted bidirectional search is supported particularly for AVC implicit weighted prediction. Only a common subset of frame relations, which falls into linear interpolation with positive weight, is

implemented. The weight between forward and backward is approximated into 5 cases only: 16 (quarter distance like Rf B X X Rb), 21 (one third distance like Rf B X Rb), 32 (half distance like Rf B Rb), 43 (two third distance like Rf X B Rb), and 48 (three quarter distance like RXXBR). Here the notation is for bidirectional prediction with display picture order, whereas Rf stands for forward reference, Rb for backward reference, B for the current bidirectional predicted picture and X is another picture in the sequence.

So if the forward prediction is {**`Ref0[i]`**}, and the corresponding backward reference is {**`Ref1[i]`**}, then the combined bidirectional motion prediction is calculated as the following:

```
Ref[i] = ((64-alpha)*Ref0[i] + alpha*Ref1[0] + 32)>>6;
```

where, **alpha** is one of the 5 weighting numbers mentioned above.

3.1.5 Skip Check

There are two SKIP modes:

- **SKIP_1MVP** – one MV pair for 16x16 macroblock, and
- **SKIP_4MVP** – four MV pairs for four 8x8 subblocks.

Otherwise, when Skip Check is enabled and the skip MV number does not exceed **MaxMumMV**, VME will first perform the fractional motion estimation at the skip centers provided by the motion vector pairs as specified by the corresponding mode. In this case the following distortions will be calculated:

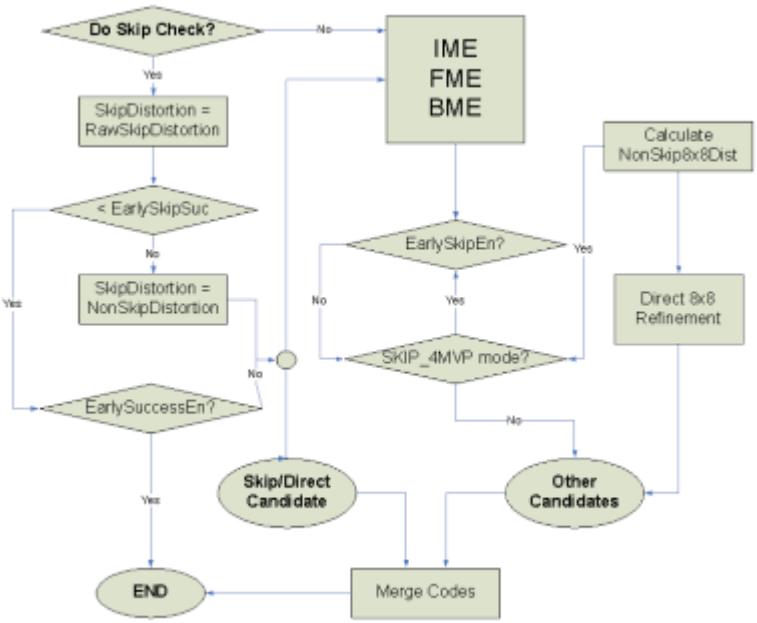
1. **RawSkipDist** – (intended for AVC PB_Skip) the raw SAD/HAAR distortion calculated from the skip motion vectors with no costing added.
2. **NonSkipDist** – (intended for AVC B_Direct16x16) the adjusted non-skip distortion is defined by adding optionally the zero motion vector cost and 16x16 Inter mode penalty to **RawSkipDist**. And
3. **NonSkip8x8Dist[4]** – (intended for AVC B_Direct8x8) the four adjusted non-skip distortions for four individual 8x8 subblocks with the ZMV cost and 8x8q Inter mode penalty optionally added. (Note: This case may produce partitions with 8x8 subblock even if the 8x8 subblock shape is disabled.)

"Optional" implies whether add or not is purely depende on two enabling input bits: **NonSkipModeAdded** and **NonSkipMvAdded**. It should be also noted that MODE_INTER_BWD is not added to **NonSkipDist** or **NonSkip8x8Dist[]** even though a skip center contains backward motion vector (this is for a direct mode, whether the motion vector for a block is forward, backward or bidirection is derived from its spatial or temporal predictor and there is no coding cost).

If **RawSkipDist** is less than or equal to **EarlySkipSuccess** threshold, **MinDist** will be set to **RawSkipDist** if the skip MV number does not exceed **MaxMumMV**.

- If **EarlySuccessEn** flag is on, VME exits immediately afer setting **MbSkipFlag** on, and **Direct8x8Pattern** = Fh.
- If **EarlySuccessEn** flag is off, VME continue the IME, FME, BME, and Direct8x8 searching after setting **MbSkipFlag** on and **Direct8x8Pattern** = Fh. VME will choose the skip output unless another better choice of code with less adjusted distortion is found.

If **RawSkipDist** is greater than **EarlySkipThreshold**, **MinDist** will be set to **NonSkipDist** if the skip MV number does not exceed **MaxMumMV**. **MbSkipFlag** will be always set to off. VME continue the IME, FME, BME, and Direct8x8 searchings. VME will still choose the skip output (with **MbSkipFlag** off) unless another better choice of code with less adjusted distortion is found.



3.1.5.1 Direct 8x8 Search

Direct8x8 Searching and then possible replacement is performed ONLY for 16x16 source block.

Direct8x8 Searching is performed only for **Skip_4MVP** mode when skip check is on to candidates of MbType in a partition that is in the Inter shape of 8x8 or minors, after IME, FME, and BME searchings.

For each candidate in 8x8 or smaller partition, and for each 8x8 sub-block, the corresponding codes will be replaced by the skip motion vector (pair) of the same 8x8 subblock, if all of the following requirements are satisfied:

- The non-skip 8x8 distortion **NonSkip8x8Dist[k]** is less than or equal to the adjusted 8x8 distortion of the corresponding codes.
- The merge does not violate uni-mix and bi-mix rules (the violating cases are skipped).
- The number of MVs used for the cadidate adding the number of subblocks of the shape 8x8 must be less than or equal to **MaxNumMV**. Or otherwise it does not replace a uni-directional 8x8 MV with a true bi-directional skip MV pair.

Note that, during all of the above comparisons, we skip the process whenever the MV numbers exceeding the **MaxNumMV**.

If either **UniMixDisable** or **BiMixDis** is set, then there would be no direct8x8 block level replacement.

3.1.5.2 Skip Check Only Mode

VME supports the skip check only mode, when **Intra** is set off, **Inter** and **Skip** are enabled, and all 7 inter shapes (**SubMbPartMask**): 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4 are set to 1 (all sub partitions are turned off). That indicates that none of the partition and sub partitions are valid for IME. Therefore IME is not performed, and no subsequent FME/BME is performed. This is another performance optimization choice if the intended usage is to check the skip centers only.

3.1.6 Intra Prediction Estimation

Intra Prediction Estimation state supports all Intra16x16, Intra8x8, and Intra4x4 modes. All predictions are based on original frame pixels for quick performance, as widely adopted in HW industry. There is a known quality drop.

For supporting AVS as well as providing finer knobs for AVC, five enabling flags are defined:

- Enable Intra16x16: whether Intra16x16 shall be performed.
- Enable Intra8x8: Enable all Intra8x8 modes, and the next flag determines which ones are actually performed.
- AVS Intra8x8 Flag: whether should perform the subset of 5 AVS modes or perform the super set of 8 AVC modes.
- Enable Intra4x4: Enable all Intra4x4 modes, and the next flag determines which ones are actually performed.
- AVS Intra4x4 Flag: whether should perform the subset of 5 AVS modes or perform the super set of 8 AVC modes.

3.1.7 Transform Adjusted SAD

A simple Wavelet transform, Haar transform, is used to refine the cost function measure of SAD. The per pixel difference goes through a 4x4 Haar transform. Then the SAD is replaced by the sum of the absolute values the transform domain coefficients (L1 norm) in the cost function. Haar transform here is used as a coarse estimation of the integer transform.

Assume the a 4x4 block **Blk** is given in the following order:

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

The 4x4 Haar transform is performed using cascaded 2x2 Haar filters of the following steps:

- Four 4-tap row filter
- Two 4-tap column filter
- Two 2-tap row filter
- Two 2-tap column filter

Where the 2x2 Haar transform is give as

```
1  1
1 -1
```

	x00 x01 x02 x03 **x10 x11 x12 x13** **x20 x21 x22 x23** **x30 x31 x32 x33**	1 0 1 0 1 0 -1 0 0 1 0 1 0 1 0 -1
1 1 0 0	**t00 t01** h08 h09	

0 0 1 1 1 -1 0 0 0 0 1 -1	**t02 t03** h10 h11 **t04t05** h12 h13 **t06t07** h14 h15	
	t00 t01 **t02 t03** h04 h05 h06 h07	1 1 1 -1
1 1 1 -1	**t00** h02 **t01** h03	
	h00 h01	

This is equivalent to the following pseudo codes:

```
void Haar(short Blk4x4[16], short Haar4x4[16])
{
      shortTmp[16];
      //First level 4-element horizontal Haar for 4 rows
      For(int i=0;i<8;i++) {
            Haar4x4[8+i] = (Blk4x4[i*2]-Blk4x4[i*2+1]);
            // Storing LP 2x4 in scan order
            Tmp[i]      = (Blk4x4[i*2]+Blk4x4[i*2+1]));
      }
      //First level 4-element vertical Haar for 2 columns
     Haar4x4[4] = (Tmp[0]-Tmp[2]);
     Haar4x4[5] = (Tmp[1]-Tmp[3]);
     Haar4x4[6] = (Tmp[4]-Tmp[6]);
     Haar4x4[7] = (Tmp[5]-Tmp[7]);
     Tmp[0] = (Tmp[0]+Tmp[2]); //Storing LP 2x2 in scan order
     Tmp[1] = (Tmp[1]+Tmp[3]);
     Tmp[2] = (Tmp[4]+Tmp[6]);
     Tmp[3] = (Tmp[5]+Tmp[7]);
     //Second level 2-element horizontal Haar for 2 columns
     Haar4x4[3] = (Tmp[0]-Tmp[1]);
     Haar4x4[2] = (Tmp[2]-Tmp[3]);
     Tmp[0] = (Tmp[0]+Tmp[2]); //Storing LP 1x2
     Tmp[1] = (Tmp[1]+Tmp[3]);
     //Second level 2-element vertical Haar
     Haar4x4[1] = (Tmp[0]-Tmp[1]);
     Haar4x4[0] = (Tmp[0]+Tmp[1]);
}
int AdjustedSAD(BYTE Src[16], BYTE Ref[16]){
     short diff[16], diffH[16];
```

```
    for(int i=0;i<16;i++) diff[i] = r[i]-s[i];
    Haar(diff,diffH);
    intasad = 0;
    for(int i=0;i<16;i++) asad += (diff[i]<0?-diff[i]:diff[i]);
    asad >>= (12-DISTBIT4X4);
    return (asad);
}
```

Thus instead of calculating the SAD of the actual pixel values, now we apply SAD to the after transformation values.

As the Haar transform basis vectors have a magnitude of ½, instead of the normalized Haar of 1/sqrt(2), the resulting transformed coefficients maintain the same bit precision as the input. Thus the sum tree has the same precision as without the transform adjustment. However, this version of Haar transform has low weightings on the DC and low AC terms, which may not be optimal as a motion-search cost function.

3.1.8 Early Decisions

There are 5 programmable early decision states are available for fine control of the VME process. All stored in one byte of U4U4 format to representing a value of (B<<S), (where B, called **base**, is the 4-LSB of the byte and S, called **shift**, is the 4-MSB of the byte,) they are the following:

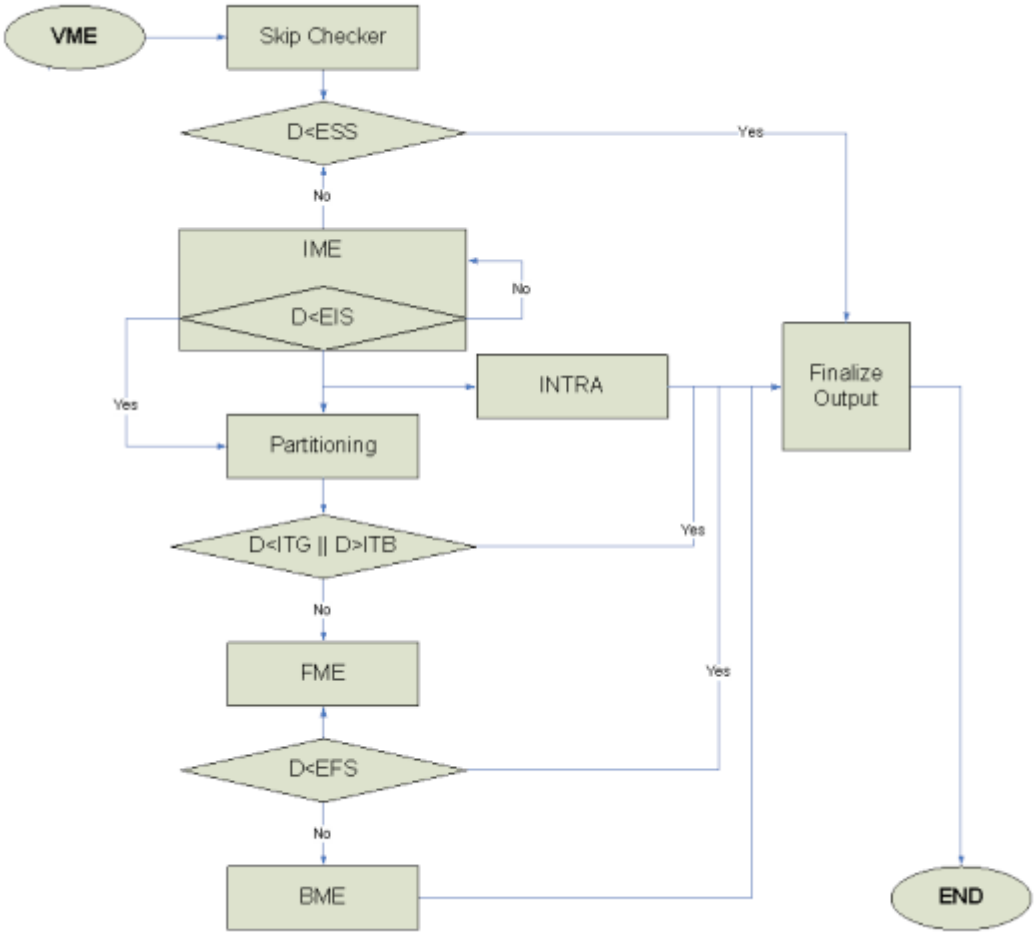
a) ESS: EarlySkipSuccess = Early successful return after Skip is checked

b) EIS: EarlyImeStop = Early IME stop when a good match is found inside of IME process.

b) ITG: ImeTooGood = Early successful return after IME is done when a good enough match is found.

a) ITB: ImeTooBad = Early termination do skip fractional and bidirectional refinement after IME is done with a hopelessly bad match as the best result.

c) EFS: EarlyFmeSuccess = Early Success after Fractional ME to skip bidirectional search.



Note. For any reason, if all possible code types are not chosen, VME will return Intra16x16 type with all modes set to 0, and the **MinDist** is set to 0x3FFF.

3.1.9 Performance Information

VME makes many internal decisions such as whether or not early exits occurred. Additionally, the number of search units processed and the total clocks spent per message are valuable to software for real-time adjustments or testing and statistical analysis. VME output message contains such information to fulfill this basic feature.

The output message for VME contains fields to encode decision and performance counters. This includes performed sub-functions (IME, FME, BME, etc), the early exit conditions, and other internal decisions.

Of the “other” internal decisions, there are fields for whether or not FME or BME improved the primary candidate. These bits will be set when FME or BME modifies the best mv decision. If the “alternate partition” or “extra candidate” results in a lower cost at the end of VME, a bit will be used to represent that the alternate beat the original best. Lastly, 1 bit will be used to indicate partitioning was constrained by MaxMV. For example, if 16x16 was the lowest sad+cost and MaxMV was set to 10, the partitioning was not constrained. However, if 8x8 was the lowest sad+cost and MaxMV was set to 1, partitioning was constrained by MaxMV and this bit would be set.

There are also 3 counter values. One is to report the total number of search units processed by the back-end (max is 48). Another is to report the total time the front-end is starved due to cache misses, counted in divisions of 16 clocks (max is 1024*16 clocks). This will most likely be active at the beginning of a VME request, however, even after processing has begun, if any front-end stalls occur this counter should resume counting. Hence, when the VME request has finished, this counter will have the total time the front-end is stalled. The third field is used to report the total time the back-end consumed for computation, also counted in divisions of 16 clocks (max is 256*16 clocks) [**Note**: this should include any bubbles in the pipe, simply put, if front-end is not stalled, this counter should be free-running]. Thus, by adding total front-end starved time with total back-end computation time, the exact total VME message time can be obtained.

3.1.10 VME Changes

VME remains fundamentally unchanged (same sub-functions, etc). However there are a few features being added:

- Bilinear interpolation,
- AVC Intra mode mask,
- Native multi-call support,
- Expanded MV cost distance),
- Motion vector, Skip center, and Cost center redefinitions to be relative to source MB,
- Removal of a skip motion vector restriction that required skip centers must be contained within the search window.

These have a non-trivial impact to the input & output message format and it is cleaner to describe a new message for, which can be found in section 6.5 and 6.6 along with further details.

3.2 Surfaces

The data elements accessed by VME are called “surfaces”. Surfaces are accessed using the surface state model.

VME uses the binding table to bind indices to surface state, using the same mechanism used by the sampling engine. A **Binding Table Index** (specified in the message descriptor) of less than 255 is used to index into the binding table, and the binding table entry contains a pointer to the SURFACE_STATE. SURFACE_STATE contains the parameters defining the surface to be accessed, including its location, format, and size.

3.3 State

3.3.1 BINDING_TABLE_STATE

VME uses the binding table to retrieve surface state. Refer to *Sampling Engine* for the definition of this state.

3.3.2 SURFACE_STATE

VME uses the surface state for current and reference surfaces. Refer to *Sampling Engine* for the definition of this state.

3.3.3 VME_STATE

This state structure contains the state used by the VME engine for data processing. VME state contains the motion search path location tables and rate-distortion weight look-up-tables. As the two sets of tables are fairly large, they are accessed as two separate states via state indexing mechanism so that applications can inter-mix the use of the search path tables and RDLUT tables.

Even though VME engine has its unique shared function ID (see Target Function ID field in the SEND instruction), the VME state is delivered through the Sampler State Pointer. When the General Purpose Pipe is used, the **Sampler State Pointer** is programmed in the MEDIA_INTERFACE_DESCRIPTOR_LOAD command and delivered directly to Sampler/VME by hardware. This posts one usage limitation. As the VME state is overloaded on top of the Sampler State Pointer, VME messages cannot be intermixed with other Sampler messages.

Each VME state may contain up to 8 VME_SEARCH_PATH_LUT_STATE. When multiple VME_SEARCH_PATH_LUT_STATE are used, they need to be stored in memory contiguously. Each VME_SEARCH_PATH_LUT_STATE contains 32 dwords in comparison of 4 dwords of a Sampler State. When enabling sampler state pre-fetch (programming the **Sampler Count** field in the MEDIA_INTERFACE_DESCRIPTOR_LOAD command), one VME_SEARCH_PATH_LUT_STATE is equivalent to 8 Samplers. Hardware may support up to two VME_SEARCH_PATH_LUT_STATE to be pre-fetched (See vol2b Media chapter for more details).

3.3.3.1 VME_SEARCH_PATH_LUT_STATE

Up to eight VME_SEARCH_PATH_LUT_STATE allowed for a message to select. Each state contains one set of search path locations, and four sets of rate distortion cost function LUT for various modes and rate distortion cost function LUT for motion vectors (relative to ‘cost center’). Motion vector cost function is provided as a piece-wise-linear curve with only the values of the power-of-2 positions provided.

DWord	Bit	Description
0:13		**Search Path**
0	31:24	Search Path Location [3] (X, Y) – Relative distance from location [2]
	23:16	Search Path Location [2] (X, Y) – Relative distance from location [1]
	15:8	Search Path Location [1] (X, Y) – Relative distance from location [0]
	7:4	Search Path location [0] (Y) – specifies relative Y distance of the **next walk** from the starting position in unit of Search Unit (SU) in U4 Format = U4, (e.g. 0x3 + 0xE = 0x1)
	3:0	Search Path Distance [0] (X) – specifies relative X distance of the next walk from the starting position in unit of SU. Format = U4
1:13		Search Path Location [4 – 55] (X, Y)
14:31		**RD LUT SET 0-4**
14	31:24	LUT_MbMode [9] for Set 1 Format = U4U4 (encoded value must fit in 12-bits)

DWord	Bit	Description
	23:16	LUT_MbMode [8] for Set 1 Format = U4U4 (encoded value must fit in 12-bits)
	15:8	LUT_MbMode [9] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [8] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
15	31:24	LUT_MbMode [9] for Set 3 Format = U4U4 (encoded value must fit in 12-bits)
	23:16	LUT_MbMode [8] for Set 3 Format = U4U4 (encoded value must fit in 12-bits)
	15:8	LUT_MbMode [9] for Set 2 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [8] for Set 2 Format = U4U4 (encoded value must fit in 12-bits)
16	31:24	LUT_MbMode [3] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	23:16	LUT_MbMode [2] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	15:8	LUT_MbMode [1] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [0] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
17	31:24	LUT_MbMode [7] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MbMode [6] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	15:8	LUT_MbMode [5] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	7:0	LUT_MbMode [4] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)

DWord	Bit	Description
18	31:24	LUT_MV [3] – For MV = 4 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MV [2] – For MV = 2 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	15:8	LUT_MV [1] – For MV = 1 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	7:0	LUT_MV [0] – For MV = 0 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
19	31:24	LUT_MV [7] – For MV = 64 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MV [6] – For MV = 32 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	15:8	LUT_MV [5] – For MV = 16 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	7:0	LUT_MV [4] – For MV = 8 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
20-23		**Finish RD LUT SET 1**
24-27		**Finish RD LUT SET 2**
28-31		**Finish RD LUT SET 3**

The assignment of LUT_MbMode entries is according to the MbTypeEx definition:

The value of each byte of the LUTs will be viewed as a pair of 4-bit units: (shift, base), and constructed as

base << shift.

For example, an entry 0x4A represents the value (0xA<<0x4) = 10*16 = 160. Encoded value must fit in **12**-bits (unsigned number); otherwise, the hardware behavior is undefined.

The only exception is for Index of 9, MODE_INTER_BWD, which is used as a bias for the two search directions. It is a signed number instead, in the form of (SU3U4) = (sign, shift, base). The sign bit indicates whether the bias is added to the forward (if sign = 1) or the backward (if sign = 0). The bias has a magnitude of (base << shift), which has 11-bits precision. It should be noted that the number is always added, there is no subtraction.

Intra Modes only apply to AVC standard. The mode penalty doesn't apply to Skip Mode Checking. Note that while other mode penalty applies to a fixed macroblock partition, MODE_INTRA_NONPRED applies to all three intra modes. It is a constant cost adder for intra-mode coding regardless of the block size.

For source block that is less than 16x16 (like a 16x8 source block), the proper mode penalty that is stated as “added per 16x16 macroblock” is added once to the source block (like MODE_INTER_16x8 is added once to a 16x8 source block). It will not be divided by the source block size.

The LUT_MV is added to all motion vector coordinate deltas in quarter-pel unit except for the SKIP mode, which no costing penalty applies. Given motion vector coordinate, e.g. *mvx*, which is in quarter-pel precision (S5.2), the mv delta is defined to be its difference from the given costing center, e.g. *ccx*, and the costing penalty is applied to $dx = |mvx-ccx|$. The cost penalty is a piecewise linear interpolation from the LUT_MV table whereas the values on power-of-2 integer samples are provided. The piecewise linear interpolation is performed using quarter-pel precision, while the LUT_MV are only provided for the given power-of-2 integer positions. The maximum distance provided in the table is 64 pixels. A linear ramp with gradient of 1 on integer distance is applied for bigger distances with maximum penalty capped to 0x3FF (10 bits). Thus,

Costing_penalty_*x* = LUT_MV[int(*dx*)], if $dx < 3$ and $dx = \text{int}(dx)$;

Costing_penalty_*x* = LUT_MV[p+1], else if $dx = 2^p$, for any $p \leq 6$;.

Costing_penalty_*x* = LUT_MV[p+1] + ((LUT_MV[p+2] – LUT_MV[p+1])*k)>>p, else if $dx = 2^p+k$, for any $p<6$ and $k<.2^p$, and

Costing_penalty_*x* = min (LUT_MV[7] + int(*dx*)– 64, 255), else if $dx > 64$.

The total costing penalty for a motion vector is

Costing_penalty = Costing_penalty_*x* + Costing_penalty_*y*

As a convention, a (0,0) relative search path distance (meaning a repeat search path location) is treated as the ending of the search path. Or the search path may also end when **Max Predetermined Search Path Length** is reached, or one of the Early Success conditions is reached**.**

Software must program the search path to terminate with at least one (0,0).

3.4 Change Details

3.4.1 Record Stream-out and Stream-in

3.4.1.1 Overview

VME internally keeps track of the best motion vectors for all shapes and sub-shapes, totaling 41 for each record of the two records (forward and backward). Once IME is finished, each record is mined for the best combination of shapes (i.e. the combination of the least distortion). The return message from VME to the EU contains only the best shape combination and the remainder of the record is discarded.

For cases when the user wants to search beyond the VME window limits (64x32 for single reference, 32x32 for dual reference) the user must call VME multiple times. Since only partial information is returned to the kernel, extracting the best shape combination across multiple calls is impossible. The best workarounds require the kernel to limit the types of shapes VME is allowed to return and then the kernel will manually merge shapes from multiple calls, cumbersome and suboptimal with respect to quality.

By returning more of the record to the kernel and allowing the kernel to feed in that information on subsequent calls as initialization information, the process of searching beyond VME size limitations is vastly improved. Now the merging of best shapes will occur inside VME and the global best shape combination is more optimized.

If both records are returned in their entirety, this would require 16 additional message phases (each shape requires 3 DWs, total of 82 shapes) for both input and output messages. A compromise to reduce this burden yet still gain the bulk of the improvement is to stream-out only the best major shapes (9 shapes, one 16x16, two 16x8, two 8x16, and four 8x8) for both records. This adds only 4 additional message phases (when under search control == 111b, otherwise 2 additional phases) and carries the most important shape data across multiple calls.

3.4.1.2 Implementation Details

In essence this feature creates 2 types of records inside VME, a local and a global record. The local record contains the best shapes within a single call to VME, i.e. the current call only. The global record is carried via stream-in and stream-out, containing the best major shapes.

VME should only consider the local record during IME and FME, finding the local call's optimal shapes independent of the global record. For purposes of partitioning, the merging of the global record's shapes into the local record should occur *after* FME is finished on the current call and *prior* to repartitioning. Otherwise local shapes identified during IME might not be considered for FME if the global shape was superior to the IME result.

There is a new stage immediately following FME where the local record major shapes are compared to the stream-in data, replacing the local record's major shapes with the stream-in shape if it has a lower distortion. Steps following this (repartitioning, BME, final mode decision) proceed like the previous generation.

As a part of the final stage, the stream-out record is generated simply taking the 9 major shapes out of the local record (which was merged with local record earlier).

The merging of global and local motion vectors prior to BME could allow the winning shape combination to not have all of its corresponding pixels in the SC (since the SC would only have local motion vector pixels). Hence, a simple check is required prior to performing BME that ensures the motion vectors are from the local call only, passing cases will perform BME and failing cases will not (test is applied on a per-shape basis).

No native support within VME for multi-reference unidirectional surface mixing, the kernel can implement a workaround if required, but there is no justification for such feature in the HW at this time.

3.4.2 MV Definitions and Precision

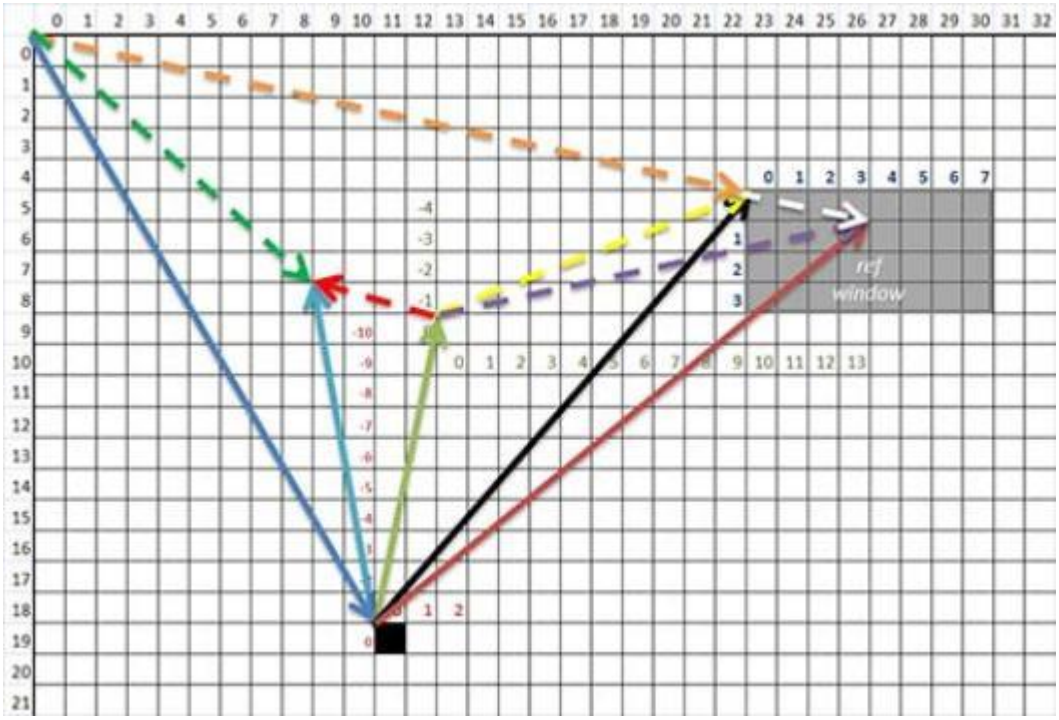
3.4.2.1 Overview

Given that VME is trying to natively support larger search windows with stream-in, due to both necessity and general improvements a number of input and output vectors (aka centers) must grow in precision. At the same time, the points from which they are relative to are also being redefined. In most cases for the previous generation (motion vectors, skip centers, cost centers) were defined relative to the reference window origin, requiring the kernel to calculate the necessary offsets from the source MB location. Now all vectors are defined relative to the source MB location (and the source MB is defined relative to the picture origin).

3.4.2.2 Implementation Details

The following diagrams provide details regarding the precision, range, and origin of all input (4 types), output (1 type), and internal vectors (first shown all together, then individually). Many vectors are composed from input or other internal vectors (via addition or subtract) and those equations are present.

										Example	
vector	origin	XRange	YRange	Equation	input	local	static	dyn	output	X	Y
src	pic	U14	U14	n\a	X		X			11	19
ref	src	S11	S9	n\a	X		X			12	-14
cost	src	S11.2	S9.2	n\a	X		X			2	-10
skip	src	S11.2	S9.2	n\a	X		X			-2	-11
pic_to_ref	pic	S15	S15	= src + ref		X	X			23	5
pic_to_skip	pic	S15.2	S15.2	= src + skip		X	X			9	8
cost_to_skip	cost	S12.2	S10.2	= skip - cost		X	X			-4	-1
cost_to_ref	cost	S12.2	S10.2	= ref - cost		X	X			10	-4
local_mv	ref	U6.2	U6.2	n\a		X		X		4	1
cost_to_local	cost	S13.2	S11.2	= cost_to_ref + local_mv		X		X		14	-3
record	src	S11.2	S9.2	= ref + local_mv				X	X	16	-13



3.4.3 Expanded MV Costs

3.4.3.1 Overview

Given that VME will be searching larger areas with the Record Stream-out feature, it is also necessary that we revisit our MV costing methodology. Given this is calculated in terms of quarter-pel units, this allows the user to provide variable costing penalty for a maximum distance of 16 pixels away from the cost center.

We would like to expand this range by implementing a variable scaling factor (i.e. right shift, binary divide) of the MV distance prior to comparison to the user-defined intervals (where VME previously looked at the lsbs only). This will be provided to VME as a 2 bit value, specifying the shift amount (0: qpel, 1: hpel, 2: single-pel, 3: two-pel). For instance, if the user a selects the MV cost scaling to be "3", this expands the maximum MV costing interval to a distance of 128 pixels.

3.4.4 Remove Skip MV Restriction

3.4.4.1 Overview

We will remove this restriction and allow the 8 skip centers to be located anywhere within the legal AVC motion vector definitions *("Horizontal motion vector range does not exceed the range of -2048 to 2047.75, inclusive, in units of luma samples." And "Vertical MV component range MaxVmvR (luma frame samples) = [-512, +511.75]").*

This restriction was originally imposed to reduce the complexity and cost of the hardware for processing skips and directs require pixels beyond that of the reference window used for IME, FME and BME.

3.4.4.2 Implementation Details

Skips must still be associated with the same surface state as their corresponding reference window (4 skip centers are for ref0, 4 are for ref1).

Skip centers are still bound as pairs. Hence, if the fwd x-component was 0xff, that meant this skip center pair was unidirectional and only in the bwd direction. If neither x-component are 0xff, then this is a bidirectional pair.

However, mv.x = 0xff is now a legal motion vector value and thus we cannot overload this field to control the skip center pair's type.We will incorporate a new 8b field, "Skip Center Enables" (M1-DW7-31:24), to control which of the 8 skip center pairs is valid. At least 1 of the skip centers for each pair must be valid when in 4MVP mode (in 1MVP mode only 1 of the skip centers for the 1st pair must be valid).

3.4.5 Bilinear Interpolation

3.4.5.1 Overview

Since MPEG2 only allows for half-pel interpolation, implementation of this bilinear filter is required only for half-pel mode. However, if there are no HW concerns implementing bilinear for quarter-pel also, please go ahead as there could be users who prefer it over our general purpose filter.

3.4.6 AVC Intra Mode Mask

3.4.6.1 Overview

AVC has 9 different intra modes for both 4x4 and 8x8 transforms and 4 modes for 16x16 transform. ***For a mask will be feed into VME (9b+9b+4b), telling it which modes cannot be selected as output candidates. This will be a 9 bit field, disabling a given mode for the entire macroblock.***

3.5 Messages

Request message bearing SFID of VME is routed to VME engine.

Programming Note:

- Use of any message to the Video Motion Estimation function with the **End of Thread** bit set in the message descriptor is not allowed.

3.5.1 VME Motion Search Request

Restrictions:

- the only surface type allowed is SURFTYPE_BUFFER.
- the surface format is ignored, data is returned from the constant buffer to the GRF without format conversion.

Applications:

- Motion search for video encoding
- Motion search for video processing such as deinterlace, frame rate conversion, etc.

Execution Mask. The execution mask is ignored.

Out-of-Bounds Accesses. Pixel replication is invoked for reads to areas outside of the surface.

3.5.2 Message Descriptor

Bit	Description	Same as Prev. Gen?
19	**Header Present.** If set, indicates that the message includes the header. This bit must be set to one for all VME messages Format = Enable	yes
18:17	Reserved: MBZ	yes
16	**Stream-in Enable.** If set, additional message phases of record stream-in will be present with the input: 4 additional phases only when search control (M0.3 10:8) is 111b (dual reference & dual record) and 2 additional phases otherwise. Format = Enable	no
15	**Stream-out Enable.** If set, additional message phases of record stream-out will be present with the output: 4 additional phases only when search control (M0.3 10:8) is 111b (dual reference & dual record) and 2 additional phases otherwise. Format = Enable	no
14:13	**Message Type** 00: Reserved 01: Inter-search only 10: Intra-search only 11: Inter- and intra-search enabled	yes
12:11	**LUT_SUBINDEX.** Specifies the index into the RDLUT state table.	yes
10:8	**VME_SEARCH_PATH_LUT State Index.** Specifies the index into the VME_SEARCH_PATH_LUT state table. When dual records are used, both records share the same predetermined search path.	yes
7:0	**Binding Table Index.** Specifies the index into the binding table for the current surface.	yes

Bit	Description	Same as Prev. Gen?
	Forward reference surface is implied as [Binding Table Index + 1] and the backward reference surface is implied as [Binding Table Index + 2] Format = U8 Range = [0,254]	

3.5.3 Input Message

3.5.3.1 Message Header and Payload

The message header and payload size is determined based on the Message Type:

Message Type	Mnemonic	Message Length	Response Length
01	Inter-search only	5 + (stream-in)	6 + (stream-out)
10	Intra-search only	5	1
11	Inter- and intra-search enabled	5 + (stream-in)	6 + (stream-out)

When stream-in is enabled:

- If (search control == 111b), the message length is +4 for total of 9 phases.
- Else (search control != 111b), the message length is +2 for total of 7 phases.

When stream-out is enabled:

- If (search control == 111b), the response length is +4 for total of 10 phases.
- Else (search control != 111b), the response length is +2 for total of 8 phases.

For Message Type of 01, the VME request message contains the following two phases:

DWord	Bit	Description	Same as Prev. Gen?
M0.7	31:0	**Reserved**	yes
M0.6	31:0	**Reserved**	yes
M0.5	31:24	**Reference Region Height (RefHeight):** This field specifies the reference region height in pixels. When bidirectional search is enabled, this applies to both search regions. Minus 16 provides the number of search point in vertical direction. The value must be a multiple of 4. Format = U8 Range = [20, 64]	yes
	23:16	**Reference Region Width (RefWidth)**: This field specifies the search region width in pixels. When bidirectional search is enabled, this applies to both search regions. Minus 16 provides the number of search point in horizontal direction. The value must be a multiple of 4. Format = U8 Range = [20, 64] **Note:** Please make sure the reference windows are not completely outside of the	yes

DWord	Bit	Description	Same as Prev. Gen?
		video frame, in that case, VME behavior is undefined.	
	15:8	Ignored	yes
	7:0	**Dispatch ID.** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.	yes
M0.4	31:0	Ignored	yes
M0.3	31	Reserved : MBZ (for **Bidirectional Mirror mode**, which is used for AVS mode. 0: disable for non-AVS mode; 1: enabled: the best forward and the best backward MV will be mirrored for AVS bidirectional search. Notice that, the mv cost penalty shall be applied only for one set of mvs in this case.)	yes
	30:24	**Sub-Macroblock Sub-Partition Mask (SubMbPartMask):** This field defines the bit-mask for disabling sub-partition and sub-macroblock modes. The lower 4 bits are for the major partitions (sub-macroblock) and the higher 3 bits for minor partitions (with sub-partition for 4x(8x8) sub-macroblocks. xxxxxx1 : 16x16 sub-macroblock disabled xxxxx1x : 2x(16x8) sub-macroblock within 16x16 disabled xxxx1xx : 2x(8x16) sub-macroblock within 16x16 disabled xxx1xxx : 1x(8x8) sub-partition for 4x(8x8) within 16x16 disabled xx1xxxx : 2x(8x4) sub-partition for 4x(8x8) within 16x16 disabled x1xxxxx : 2x(4x8) sub-partition for 4x(8x8) within 16x16 disabled 1xxxxxx : 4x(4x4) sub-partition for 4x(8x8) within 16x16 disabled *Usage note: one example usage of only enabling 4x(4x4) sub-partition while all other partitions are disabled is for video processing, whereas parallel motion searches are performed for 16 4x4 blocks. For that no further block combination (into larger sub-partitions/sub-macroblocks) is needed.*	yes
	23:22	**Intra SAD Measure Adjustment (IntraSAD):** This field specifies distortion measure adjustments used for the motion search SAD comparison. This field must be set to 00 if Source Block Field Mode is 1 (interleaved). 00 : none 01 : Reserved Better set to 00 if Source Block Field Mode is 1 (interleaved).	yes
	21:20	**Inter SAD Measure Adjustment (InterSAD):** This field specifies distortion measure adjustments used for the motion search SAD comparison. 00 : none 01 : Reserved 10 : Haar transform adjusted	yes

DWord	Bit	Description	Same as Prev. Gen?
		11 : Reserved Better set to 00 if Source Block Field Mode is 1 (interleaved).	
M0.3	19	**Block-Based Skip Enabled:** when this field is set on the skip thresholding passing criterion will be based on the maximal distortion of individual blocks (8x8's or 4x4's) instead of their sum (i.e. the distortion of 16x16). The block size is 8x8 if and only if the **Transform 8x8 Flag** is set to ON and the source size is 16x16..	yes
	18	Not implemented. Reserved: MBZ	no
	17	**Disable Aligned VME Source Fetch:** This field, when set disables the VMEunit functionality that aligns source data requests to 16 pixels. (This bit is ignored if **SrcX** and **SrcSize** are such that requests for source data cannot be aligned to 16 pixels. The source data requests will be misaligned in these cases)	yes
	16	**Disable Aligned VME Reference Fetch:** This field, when set disables the VMEunit functionality that fragments reference data requests which are not aligned to 16 pixels into 16 pixel aligned requests. This may be used when the surface is not a multiple of 16 pixels and a portion of the reference data is outside the surface.	yes
	15	**Disable Field Cache Allocation:** This field, when set to 1, disables the optimized field cache line method in the Sampler Cache for reference block data when RefAccess is 1 (field based). It is ignored by hardware if RefAccess is 0. 0 – frame cache lines 1 – field cache lines	yes
	14	**Skip Mode Type (SkipType):** For B_DIRECT_16x16, both motion vectors of the skip center pair 0 are used. For B_DIRECT_8x8s, all four skip center pairs are **fully** used (VME will never try to combine them with non-skip shapes from IME, FME or BME). **0 : SKIP_1MVP** – one MV pair for 16x16 **1 : SKIP_4MVP** – Four MV pairs for 8x8s (in this case and only this case, **SkipCenter Delta 1-3** will be used) Note: SkipTypeMode should be programmed to 1MVP for non-16x16 Source size	yes
	13:12	**Sub-Pel Mode (SubPelMode):**This field defines the half/quarter pel modes. The mode is inclusive, ie., higher precision mode samples lower precision locations. 00 : integer mode searching 01 : half-pel mode searching 10 : reserved 11 : quarter-pel mode searching	yes
	11	**Dual Search Path Option:** Used only for dual record cases, this field flags	yes

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th><th>Same as Prev. Gen?</th></tr>
<tr><td></td><td></td><td>whether two searching records uses the same or the different paths.

0: use the same path as specified by the Search Path Location array

1: use the different paths, the first one uses the even entries of the Search Path Location array and the second one uses the odd entries of the Search Path Location array.</td><td></td></tr>
<tr><td></td><td>10:8</td><td>Search Control (SearchCtrl): This field specifies how the motion search is performed.

The following table shows the valid encodings. Other encodings are reserved.

<table>
<tr><th>Code</th><th>Mode</th></tr>
<tr><td>000</td><td>Single reference, single record and single start.

Search is performed only on reference 0; only cost center 0 and start 0 are used. There is only one record. Adaptive search is also allowed. However, when AdaptiveEn is on, LenSU must be at least 2 as the adaptive search in VME is one-step delayed.

This is the common single directional motion search mode.</td></tr>
<tr><td>001</td><td>Single reference, single record and dual start.

Search is performed only on reference 0; only cost center 0 is used. There is only one record. Search performs first on start 0 and then on start 1. Then if LenSP is not reached, the predetermined search path will start on start 1 with increment added to start 1 location. It then is followed by adaptive search.

This is used for single direction adaptive search.</td></tr>
<tr><td>011</td><td>Single reference, dual record (and implied dual start).

Search is performed only on reference 0; both cost center 0 and 1 and start 0 and 1 are used. Two records are used for both paths during IME.

When integer search is complete, the two records are combined to find the best search. Sub-pel</td></tr>
</table>
</td><td>yes</td></tr>
</table>

DWord	Bit	Description	Same as Prev. Gen?
		refinement is only performed from the best one. This may be used for search for multiple motion search candidates/predicators.	
		111 **Dual reference, dual record** (and implied dual start). Search is performed on references 0/1 with both cost centers 0/1 and starts 0/1. Two records are used for both paths during IME. When integer search is complete, and then sub-pel refinement is also performed separately, the two records are combined to find the best search on a subblock basis. This may be used for bidirectional motion search, or multi-reference P search. Whether bidirectional is enabled or not depends on the bidirection sub-macroblock mask. If **BiSubMbPartMask** is set to 1111'b, bidirectional search is disabled. VME will output only the best unidirectional search results. Otherwise, BME will be performed. *Note that bidirectional search and sub-pel refinement are orthogonal features that can be enabled indepdently.*	
	7	**Reference Access (RefAccess):** This field defines how the reference blocks are accessed from the reference frames. It indicates if the source picture is a frame picture or a field picture. *Programming Note: For all known video coding standards, reference pictures always have the same picture type as the source picture. Therefore, this field should be programmed to be the same as **SrcAccess**.* 0 : frame based 1 : field based	yes
	6	**Source Access (SrcAccess):** This field defines how the source block is accessed from the source frame. It indicates if the source picture is a frame picture or a field picture. It is similar to the Picture Type used in video standards. 0 : frame based 1 : field based	yes
	5:4	**Inter MbType Remap (MbTypeRemap):** This field controls the mapping of the	yes

DWord	Bit	Description	Same as Prev. Gen?
		output MbType when the VME output is an Inter (IntraMbFlag = INTER). The intended usage, for example, is for two forward (or backward) references or for two search regions from the same reference picture in one VME call. Hardware ignores this field if the VME output is an intra type (IntraMbFlag = INTRA). 00 : no remapping 01 : remapping MbType to forward only (1-3 mapped to 1, even numbers in [4-14h] mapped to 4, odd numbers in [5-15h] mapped to 5, and 16h is unchanged) 10 : remapping MbType to backward only (1-3 mapped to 2, even numbers in [4-14h] mapped to 6, odd numbers in [5-15h] mapped to 7, and 16h is unchanged) 11 : reserved	
	3	**Reserved: MBZ**	yes
	2	**Reserved : MBZ**	yes
	1:0	**Source Block Size (SrcSize):** This field defines how the 16x16 source block is formed. When Source Block Size is less than 16x16, SU larger than 4x4 will be used. 00: 16x16 01: 16x8 10: Reserved (for 8x16) 11: 8x8	yes
M0.2	31:16	**Source Y (SrcY):** This field defines the vertical position (of the block's upper-left pixel) in unit of pixels for the source block in the source picture (relative to picture origin, not frame origin). For field source (SrcAccess=1), the SrcFieldPolarity (M1.7-19), is required by hardware to identify if this is top or bottom field of an interleaved memory surface. The resulting Y address in the reference picture must be in even line aligned within the reference picture. Specifically, if the reference picture is a frame picture. the resulting Y address must be 2-line aligned; if the reference picture is a field picture within a frame storage, and the resulting Y address must be 2-line aligned within the field. i.e. it must be an even number for the frame case, and must be equal to 0 or 1 modulo 4 for the field case. Format = U16	no
	15;0	**Source X (SrcX):** This field defines the horizontal position (of the block's upper-left pixel) in unit of pixels for the source block in the source picture. The source block must be within the source picture starting at any integer grid. Format = U16	yes
M0.1	31:16	**Reference 1 Y Delta (Ref1Y):** This field defines the vertical position (of the upper-left corner of the reference region) in unit of pixels for Reference 1 region relative to the source MB Y value on its respective picture. For field reference (RefAccess=1), the Ref1FieldPolarity (M1.7-21), is required by hardware to identify if this is top or bottom field of an interleaved memory surface.	no

DWord	Bit	Description	Same as Prev. Gen?
		The resulting Y address in the reference picture must be in even line aligned within the reference picture. Specifically, if the reference picture is a frame picture. the resulting Y address must be 2-line aligned; if the reference picture is a field picture within a frame storage, and the resulting Y address must be 2-line aligned within the field. i.e. it must be an even number for the frame case, and must be equal to 0 or 1 modulo 4 for the field case. **Note: For search control=3, this must equal Ref0Y.** Format = S15 Hardware Range: [-2048 to 2047]	
	15;0	**Reference 1 X Delta (Ref1X):** This field defines the horizontal position (of the upper-left corner of the reference region) in unit of pixels for Reference 1 region relative to the source MB X value on its respective picture. The resulting reference region is allowed to be outside the picture. Pixel replication is applied to generate out of bound reference pixels. This field is only valid when dual reference mode is selected **Note: For search control=3, this must equal Ref0X.** Format = S15 Hardware Range: [-2048 to 2047]	no
M0.0	31:16	**Reference 0 Y Delta (Ref0Y):** This field defines the vertical position (of the upper-left corner of the reference region) in unit of pixels for Reference 0 region relative to the source MB Y value on its respective picture. For field reference (RefAccess=1), the Ref0FieldPolarity (M1.7-20), is required by hardware to identify if this is top or bottom field of an interleaved memory surface. The resulting Y address in the reference picture must be in even line aligned within the reference picture. Specifically, if the reference picture is a frame picture. the resulting Y address must be 2-line aligned; if the reference picture is a field picture within a frame storage, and the resulting Y address must be 2-line aligned within the field. i.e. it must be an even number for the frame case, and must be equal to 0 or 1 modulo 4 for the field case. Format = S15 Hardware Range: [-2048 to 2047]	no
	15;0	**Reference 0 X Delta (Ref0X):** This field defines the horizontal position (of the upper-left corner of the reference region) in unit of pixels for Reference 0 region relative to the source MB X value on its respective picture. The resulting reference region is allowed to be outside the picture. Pixel replication is applied to generate out of bound reference pixels. Format = S15 Hardware Range: [-2048 to 2047]	no
M1.7	31:24	**Skip Center Enable Mask (SkipCenterMask):** [bit 31…24] xxxx xxx1: Ref0 Skip Center 0 is enabled [corresponds to M2.0]	no

DWord	Bit	Description	Same as Prev. Gen?
		xxxx xx1x: Ref1 Skip Center 0 is enabled [corresponds to M2.1] xxxx x1xx: Ref0 Skip Center 1 is enabled [corresponds to M2.2] xxxx 1xxx: Ref1 Skip Center 1 is enabled [corresponds to M2.3] xxx1 xxxx: Ref0 Skip Center 2 is enabled [corresponds to M2.4] xx1x xxxx: Ref1 Skip Center 2 is enabled [corresponds to M2.5] x1xx xxxx: Ref0 Skip Center 3 is enabled [corresponds to M2.6] 1xxx xxxx: Ref1 Skip Center 3 is enabled [corresponds to M2.7] Illegal cases: Disable both Ref0 and Ref1 Skip Center 0 in case of Skip_1MVP. Disable both Ref0 and Ref1 for any Skip Center pair in case of Skip_4MVP.	
	23:22	Reserved: MBZ	yes
	21	**Reference1 Field Polarity Select (Ref1FieldPolarity):** If RefAccess = 1 (M0.3-7), meaning field based, than the hardware requires this value is to derive the correct location on the reference surface in memory to fetch pixels. This is because the reference is stored as a frame picture with both fields interleaved in memory and the Ref1Y (M0.1-31:16) is relative to the SrcY location on a field picture. Hence, the starting y-pixel coordinate that will be fetched from the memory will be: (SrcY+Ref1Y) * 2 + Ref1FieldPolarity Else, this field is ignored by the hardware. Format = U1	no
	20	**Reference0 Field Polarity Select (Ref0FieldPolarity):** If RefAccess = 1 (M0.3-7), meaning field based, than the hardware requires this value is to derive the correct location on the reference surface in memory to fetch pixels. This is because the reference is stored as a frame picture with both fields interleaved in memory and the Ref0Y (M0.0-31:16) is relative to the SrcY location on a field picture. Hence, the starting y-pixel coordinate that will be fetched from the memory will be: (SrcY+Ref0Y) * 2 + Ref0FieldPolarity Else, this field is ignored by the hardware. Format = U1	no
	19	**Source Field Polarity Select (SrcFieldPolarity):** If SrcAccess = 1 (M0.3-6), meaning field based, than the hardware requires this value is to derive the correct location on the source surface in memory to fetch pixels. This is because the source is stored as a frame picture with both fields interleaved in memory and the SrcY value (M0.2-31:16) is the location on the field picture (in other words, it does not convey the field polarity). Hence, the starting y-pixel coordinate that will be fetched from the memory will be:	no

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th><th>Same as Prev. Gen?</th></tr>
<tr><td></td><td></td><td>SrcY* 2 + SrcFieldPolarity

Else, this field is ignored by the hardware.

Format = U1</td><td></td></tr>
<tr><td></td><td>18</td><td>Bilinear Filter Enable (BilinearEnable):

If set, the fractional filter will implement a simple bilinear interpolation filter instead of the 4-tap filter. Note: this is supported for both hpel and qpel interpolation.

Format = Enable</td><td>no</td></tr>
<tr><td></td><td>17:16</td><td>MV Cost Scaling Factor (MVCostScaleFactor):

This term allows the user to redefine the precision of the lookup into the LUT_MV based on the MV cost difference from the cost center. The piecewise linear cost function is defined from 0 to 64 in powers of 2 intervals, and the precision of the difference is set by this field. There are 4 precision choices:

00: qpel [Qpel difference between MV and cost center: eff cost range 0-15pel]

01: hpel [Hpel difference between MV and cost center: eff cost range 0-31pel]

10: pel [Pel difference between MV and cost center: eff cost range 0-63pel]

11: 2pel [2Pel difference between MV and cost center: eff cost range 0-127pel]

Format = U2</td><td>no</td></tr>
<tr><td></td><td>15:8</td><td>Macroblock Intra Structure (MbIntraStruct): This is a bitmask specifies neighbor macroblock availability. This allows software to constrain intra prediction mode search.

Note: user must set Bit6=Bit5.
<table>
<tr><th>Bits</th><th>MotionVerticalFieldSelect Index</th></tr>
<tr><td>7</td><td>Reserved : MBZ</td></tr>
<tr><td>6</td><td>Reserved : MBZ</td></tr>
<tr><td>5</td><td>IntraPredAvailFlagE/A – A (Left neighbor or Left bottom half)</td></tr>
<tr><td>4</td><td>IntraPredAvailFlagB – B (Upper neighbor)</td></tr>
<tr><td>3</td><td>IntraPredAvailFlagC – C (Upper left neighbor)</td></tr>
<tr><td>2</td><td>IntraPredAvailFlagD – D (Upper right neighbor)</td></tr>
<tr><td>1:0</td><td>Reserved : MBZ</td></tr>
</table></td><td>yes</td></tr>
<tr><td></td><td>7</td><td>Luma Intra Source Corner Swap (IntraCornerSwap): This field specifies the format of the intra luma neighbor pixel format in the message.</td><td>yes</td></tr>
</table>

DWord	Bit	Description	Same as Prev. Gen?
		0: top neighbors are in sequential order 1: Left-top corner is swapped with the last left-edge neighbor	
	6	**Non Skip MB Mode Cost Added (NonSkipModeAdded):** This field indicates that the distortion of the survived motion vectors will become non-skip, and the MB mode cost will be added to its distortion.	yes
	5	**Non Skip Zero MV Cost Added (NonSkipZMvAdded):** This field indicates that the distortion of the survived motion vectors will become non-skip, and the zero MV component costs will be added to its distortion.	yes
	4:0	**Luma Intra Partition Mask (IntraPartMask):** This field specifies which Luma Intra partition is enabled/disabled for intra mode decision. xxxx1 : luma_intra_16x16 disabled xxx1x : luma_intra_8x8 disabled xx1xx : luma_intra_4x4 disabled Bits [4:3] MBZ	yes
M1.6	31:0	Reserved: MBZ	no
M1.5	31:16	**CostCenter 1 Delta Y (CostCenter0Y):** This field defines the Y value for the second cost center (associated with the second start) relative to the picture source MB Y value. Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]	no
	15:0	**CostCenter 1 Delta X (CostCenter1X):** This field defines the X value for the second cost center (associated with the second start) relative to the picture source MB X value. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]	no
M1.4	31:16	**CostCenter 0 Delta Y (CostCenter0Y):** This field defines the Y value for the first cost center (associated with the first start) relative to the picture source MB Y value. Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]	no
	15:0	**CostCenter 0 Delta X (CostCenter0X):** This field defines the X value for the first cost center (associated with the first start) relative to the picture source MB X value. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]	no
M1.3	31:24	**IME Success & FME/BME Bypass Threshold (ImeTooGood):** This field specifies the threshold value for the ME distortion computes above which sub-pel refinement search and bidirectional search are skipped (as the integer-pel	yes

DWord	Bit	Description	Same as Prev. Gen?
		distortion is deemed to be good enough). This value, if used, should be set to be greater than Early Success Threshold. Format = U4U4 (encoded value should fit in 14-bits)	
	23:16	**Quit Inter Search Threshold (ImeTooBad):** This field specifies the threshold value for the ME distortion computes above which sub-pel refinement search and bidirectional search are skipped (as the integer-pel distortion is deemed to be too bad). This value, if used, should be set to be greater than Early Success Threshold. Format = U4U4 (encoded value should fit in 14-bits)	yes
	15:8	**Partition Distortion Tolerance Threshold (PartToleranceThrhd):** defines the distortion tolerance used in the intermediate shape decision. (See Shape Decision for more detail). This field is only valid when **PartCandidateEn** is set. Format = U4U4 (encoded value should fit in 14-bits)	yes
	7:0	**FME/BME Pruning Tolerance Threshold (FBPrunThrhd):** This field specifies the threshold when a normalized absolute difference of the two uni-directional distortions is bigger than that, FME is skipped for the losing direction and BME is skipped as well if bidirectional is enabled. The difference is normalized by the number of 4x4 pixels in the tested partition. For example, for an 8x8 partition, the absolute difference of the distortions is divided by 4 (right shifted by 2); and for a 16x16 partition, it is right shifted by 4. With the unsigned byte, this field provides a control of per pixel distortion difference with a large range from 1/16 to 16. This field is only valid when **FBPrunEn** is set to 1 (and for Search Control set to 111 - dual reference and dual record). Format = U4U4 (encoded value should fit in **14**-bits)	no
M1.2	31:28	**Start Center 1 Y (Start1Y):** This field defines the Y position of Search Path 1 relative to the reference Y location. It is in unit of SU. Format = U4	yes
	27:24	**StartCenter 1 (Start1X):** This field defines the X position of Search Path 1 relative to the reference X location. It is in unit of SU. The corresponding reference block must be fully within the reference region. Format = U4	yes
	23:20	**Start Center 0 Y (Start0Y):** This field defines the Y position of Search Path 1 relative to the reference Y location. It is in unit of SU. Format = U4	yes
	19:16	**StartCenter 0 X (Start0X):** This field defines the X position of Search Path 1 relative to the reference X location. It is in unit of SU. The corresponding reference block must be fully within the reference region. Format = U4	yes

DWord	Bit	Description	Same as Prev. Gen?
	15:8	**Maximum Search Path Length (MaxNumSU):** This field defines the maximum number of SUs per reference including the predetermined SUs and the adaptively generated SUs. Note: every SU in fixed path will be counted (including the out-bound ones and repeated ones), and in addition for adaptive SUs only the ones actually searched will be added. Format = U8, with valid range of [1,63]	yes
	7:0	**Max Fixed Search Path Length (LenSP):** This field defines the maximum number of SUs per reference which are evaluated by the predetermined SUs. When adaptive walk is enabled, adaptive walk starts when this number is reached. Note: every SU in fixed path will be counted (including the out-bound ones and repeated ones) Format = U8, with valid range of [1,63]	yes
M1.1	31	**Extented FME Repartition Enable (RepartEn):** This field specifies whether the repartitioning after FME as described in 6.1.3.3 is enabled. 0 : disable 1 : enable	yes
	30	**FME/BME Pruning Enable (FBPrunEn):** This field specifies whether FME/BME pruning is enabled. This is used to speedup the VME operation with low quality impact. This field is only valid for dual reference case (when Search Control is 111). Otherwise, it must be set to zero. 0 : disable 1 : enable Reserved: MBZ	no
	29	**AdaptiveValidationControl**: if it is on, during adaptive IME searching, VME will not pipeline SUs. In other words, VME will have only 1 SU in-flight and will wait until all results are updated prior to deciding which SU to move to next. When this is off (non-validation mode), VME's pipeline will have at most 2 SUs in-flight to enhance performance, but the out-of-order SU completion behavior will make validation efforts exceedingly complex.	yes
	28	**Unidirectional Mix Disable (UniMixDisable):** if it is on, all unidirectional resulting motion vectors must share the same direction, *i.e.* either all are forward, or all are backward. If this field is off, each partition, down to 8x8 subblock, may have a different mix of forward and backward motion vectors. (Within each 8x8 subblock, only one common choice is allowed.) This field is MBZ except for cases of Search Control = 111'b (e.g. 7, dual reference).	yes
	27:24	**Bidirectional Sub-Macroblock and Sub-Partition Mask (BiSubMbPartMask):** This field defines the bit-mask for disabling sub-macroblock and sub-partition modes. The enabled ones must be a subset of that enabled by **SubMbPartMask**.	yes

DWord	Bit	Description	Same as Prev. Gen?
		Note that 16x8 and 8x16 share the same bit and all sub-partitions share the same bit. xxx1 : 16x16 disabled xx1x : 2x(16x8) and 2x(8x16) within 16x16 disabled x1xx : 4x(8x8) within 16x16 disabled 1xxx: sub-partitions 2x(8x4) and 2x(4x8) and 4x(4x4) within 8x8 are disabled	
	23:22	**Reserved: MBZ**	yes
	21:16	**Bidirectional Weight (BiWeight):** This field defines the weighting for the backward and forward terms to generate the bidirectional term. This field is only valid for bidirectional search (**SearchCtrl** = 111). Format = U6 Valid Values: [16, 21, 32, 43, 48]	yes
	15:6	**Reserved: MBZ**	yes
	5:0	**Maximum Number of Motion Vectors (MaxNumMVs):** This field specifies the maximum number of motion vectors allowed for the current macroblock. This field affects the macroblock partition decision. VME will return the best partition with **MvQuantity** not exceeding **MaxNumMVs**. **MaxNumMVs =** 0 will only allow skip as a valid Inter mode. **Note:** This value is used ONLY for 16x16 source MB mode. ***Usage Example:*** *Certain profiles/levels for AVC standard have restriction for the maximum number of motion vectors allowed for two consecutive macroblocks (MaxMvsPer2Mb may be 16 or 32).* Format = U6 **Note:** When skip is enabled, **MaxNumMVs** must be greater or equal to the number of skip MVs.	yes
M1.0	31:24	**Early IME Successful Stop Threshold (EarlyImeStop):** This field specifies the threshold value for the IME distortion computes of single 16x16 mode below which no more search will be performed within the IME unit. This field only takes effect if **EarlyImeSuccessEn** is set. **Note**: Early IME exit only looks at ref0, and uses 8x8 for source 8x8 and 16x8 0 for source 16x8. Format = U4U4 (encoded value should fit in 14-bits)	yes
	23:16	**Early Fme Success Threshold (EarlyFmeSuccess):** Applying after fractional ME, this field defines the threshold value for the ME distortion computes below which the search process will exit early. This field only takes effect if **EarlySuccessEn** is set. This field only looks at primary candidate Format = U4U4 (encoded value should fit in 14-bits)	yes

DWord	Bit	Description	Same as Prev. Gen?
	15:8	**Skip Success Threshold (SkipSuccess):** Applying after skip mode checking (if enabled), this field defines the threshold value for the ME distortion computes below which the search process will exit early. This threshold is **always** used for setting MbSkipFlag, when the corresponding raw distortion is less than or equal to the threshold. This field causes early VME termination only if **EarlySuccessEn** is set to 1. Format = U4U4 (encoded value should fit in 14-bits)	yes
	7	**Transform 8x8 Flag For Inter Enable (T8x8FlagForInterEn):** This field specifies whether Transform8x8Flag is updated for inter mode according the resulting inter-mode sub-partition size. 0 : disable 1 : enable	yes
	6	**Quit Inter Search Enable (QuitInterEn)**: This field specifies whether the inter search may be prematurely terminated after IME when the IME distortion is worse than the predetermined threshold **QuitInterThrhd**. When this field is not set, if early out does occur on full-pel location, hardware switches to local sub-pel refinement search. When this field is set, however, the local sub-pel refinement step is skipped. This field takes effect independent of **EarlySuccessEn**. 0 : disable 1 : enable	yes
	5	**Early IME Success Enable (EarlyImeSuccessEn)**: This field specifies whether the Early Success may terminate on full-pel precision. When this field is not set, if early out does occur on full-pel location, hardware continues to local sub-pel refinement search and so on. When this field is set, however, the local sub-pel refinement step is skipped and intra search is also skipped. This field only takes effect if **EarlySuccessEn** is set. ***Usage example****: This may be used for cases with large static area where (0,0) motion vector delivers very good results that no FME refinement is needed and also intra check is also skipped. This may also be used in place of Skip Mode Checking when the skip center(s) happens to be an integer location inside the SU of the Start Center(s).* 0 : disable 1 : enable	yes
	4	**Early Success Enable (EarlySuccessEn):** This field enables Early Success of the motion search when the ME distortion is below **EarlySuccessThrhd**. Early Success may occur during skip mode check, integer search and sub-pel search stages. Termination directly out of integer search is controlled by the **EarlySuccessImeEn** field. 0 : disable 1 : enable	yes

DWord	Bit	Description	Same as Prev. Gen?
	3	**Partition Candidates Enable (PartCandidateEn):** This field enables multiple partition candidates (VME hardware supports only up to two candidates). When it is set, a second partition candidate that is within **PartToleranceThrhd** from the best partition is kept for subsequent inter-search operations. This field is only allowed to be set to 1 if **SrcSize** is 16x16. 0: a single partition is determined by IME 1: multiple partition candidates are allowed	yes
	2	**Bidirectional Mix Disable (BiMixDis):** if it is on, all resulting motion vectors must share the same direction, *i.e.* either all are unidirectional (i.e. forward or backward), or all bidirectional. If this field is off, each partition may have different search direction (forward, backward or bidirectional). **Usage Example:** MPEG2 bidirectional decision is at whole macroblock level, while AVC decision is at subblock level. 0: bidirectional decision on subblock level that bidirectional mode is enabled 1: bidirectional decision on whole macroblock	yes
	1	**Adaptive Search Enable (AdaptiveEn):** This field defines whether adaptive searching is enabled for IME. When Adaptive Search is enabled, there must be at least two search steps preceded. It is either from a single start with step of >=2 or from a dual-start. 0 : disable 1 : enable	yes
	0	**Skip Mode Enable (SkipModeEn):** This field specifies whether the skip mode checking is performed before the motion search. If this field is set, Skip Center, which may have a sub-pel precision, is first tested before IME. 0 : disable 1 : enable **Note**: It must be 0 if Inter is not ON in **Message Type** or if **SrcType!=00 (**less than 16x16)	yes
M2.7	31:0	**Ref1 SkipCenter 3 Delta XY** *(for definition see M2.0)*	no
M2.6	31:0	**Ref0 SkipCenter 3 Delta XY** *(for definition see M2.0)*	no
M2.5	31:0	**Ref1 SkipCenter 2 Delta XY** *(for definition see M2.0)*	no
M2.4	31:0	**Ref0 SkipCenter 2 Delta XY** *(for definition see M2.0)*	no
M2.3	31:0	**Ref1 SkipCenter 1 Delta XY** *(for definition see M2.0)*	no
M2.2	31:0	**Ref0 SkipCenter 1 Delta XY** *(for definition see M2.0)*	no
M2.1	31:0	**Ref1 SkipCenter 0 Delta XY** *(for definition see M2.0)*	no

DWord	Bit	Description	Same as Prev. Gen?
M2.0	31:16	**Ref0 Skip Center 0 Delta Y:** This field defines the Y value for the forward skip center relative to the 8x8 block offset from the source MB Y location in quarter-pel precision associated with Ref0. To match the relative 8x8 block location, the HW will add fixed offsets to the 4 skip centers in each direction to generate the correct pixel location to fetch the data. For SkipCenter 0: VME will add 0 to the user-input Y value. For SkipCenter 1: VME will add 0 to the user-input Y value. For SkipCenter 2: VME will add 32 to the user-input Y value. For SkipCenter 3: VME will add 32 to the user-input Y value. Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]	no
	15:0	**Ref0 SkipCenter 0 Delta X:** This field defines the X value for the forward skip center relative to the 8x8 block offset from the source MB X location in quarter-pel precision associated with Ref0. To match the relative 8x8 block location, the HW will add fixed offsets to the 4 skip centers in each direction to generate the correct pixel location to fetch the data. For SkipCenter 0: VME will add 0 to the user-input X value. For SkipCenter 1: VME will add 32 to the user-input X value. For SkipCenter 2: VME will add 0 to the user-input X value. For SkipCenter 3: VME will add 32 to the user-input X value. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]	no

For Message Type of 10 and 11, the VME request message has additional two phases to deliver the neighbor macroblock pixels for intra prediction. Here the neighbor pixel location [x, y] is relative to the current 16x16 macroblock, with [x,y] = [-1, -1] for the upper-left corner edge pixel in neighbor D, [-1, 0…15] for the left edge pixels in neighbor A, and [0…15…23, -1] for the upper and upper-right edge pixels in neighbors B and C.

Note that for Message Type of 10, which is intra-search only mode, the fields regarding reference windows and inter-prediction control in the command are ignored by hardware (and no pixels are fetched from the reference window(s)).

To help with vector data access in software, horizontal neighbor pixels from D, B, and C are stored in one register in raster order with 8 pixel alignment. Vertical neighbor pixels from A are stored in a separate register.

DWord	Bit	Description	Same as Prev. Gen?

DWord	Bit	Description	Same as Prev. Gen?
M3.7	31:0	**Neighbor pixel Luma value [23, -1] to [20, -1].** Upper-right pixels from neighbor macroblock C	yes, but was M2
M3.6	31:0	**Neighbor pixel Luma value [19, -1] to [16, -1].** Upper-right edge pixels from neighbor macroblock C	yes, but was M2
M3.5	31:0	**Neighbor pixel Luma value [15, -1] to [12, -1].** Top edge pixels from neighbor macroblock B	yes, but was M2
M3.4	31:0	**Neighbor pixel Luma value [11, -1] to [8, -1].** Top edge pixels from neighbor macroblock B	yes, but was M2
M3.3	31:0	**Neighbor pixel Luma value [7, -1] to [4, -1].** Top edge pixels from neighbor macroblock B	yes, but was M2
M3.2	31:24	**Neighbor pixel Luma value [3, -1].** Fourth top edge pixel from neighbor macroblock B	yes, but was M2
	23:16	**Neighbor pixel Luma value [2, -1].** Third top edge pixel from neighbor macroblock B	yes, but was M2
	15:8	**Neighbor pixel Luma value [1, -1].** Second top edge pixel from neighbor macroblock B	yes, but was M2
	7:0	**Neighbor pixel Luma value [0, -1].** First top edge pixel from neighbor macroblock B	yes, but was M2
M3.1	31:24	**Corner Neighbor pixel 0.** Its content depends on **IntraCornerSwap** field. It swaps with **Corner Neighbor pixel 1**. **Neighbor pixel Luma value [-1, -1]**. The one upper-left edge pixel from neighbor macroblock D, which is the right most edge pixel of D, if **IntraCornerSwap** field is 0. Or **Neighbor pixel Luma value [-1, 15]**. The last left edge pixel from neighbor macroblock A, which is the left most edge pixel of D, if **IntraCornerSwap** field is 1.	yes, but was M2
	23:4	Reserved: MBZ (Hardware ignores this field)	no
	3:0	**AVC Intra 16x16 Mode Mask (Intra16x16ModeMask):** Disables given intra mode as follows. xxx1: xx1x: x1xx: 1xxx:	no
M3.0	31:25	Reserved: MBZ (Hardware ignores this field)	no
	24:16	**AVC Intra 8x8 Mode Mask (Intra16x16ModeMask):** Disables given intra mode as follows.	no

DWord	Bit	Description	Same as Prev. Gen?
		x xxxx xxx1: x xxxx xx1x: x xxxx x1xx: x xxxx 1xxx: x xxx1 xxxx: x xx1x xxxx: x x1xx xxxx: x 1xxx xxxx: 1 xxxx xxxx:	
	15:9	Reserved: MBZ (Hardware ignores this field)	no
	8:0	**AVC Intra 4x4 Mode Mask (Intra16x16ModeMask):** Disables given intra mode as follows. x xxxx xxx1: x xxxx xx1x: x xxxx x1xx: x xxxx 1xxx: x xxx1 xxxx: x xx1x xxxx: x x1xx xxxx: x 1xxx xxxx: 1 xxxx xxxx:	no
M4.7	31:0	Reserved: MBZ	no
M4.6	31:0	Reserved: MBZ	no
M4.5	31:0	Reserved: MBZ	no
M4.4	31:28	**Intra Predictor Mode for Neighbor B15 (IntraMxMPredModeB15):** This field carries the intra prediction mode of the fourth bottom 4x4 block (Block 15 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.	yes, but was M3
	27:24	**Intra Predictor Mode for Neighbor B14 (IntraMxMPredModeB14):** This field carries the intra prediction mode of the third bottom 4x4 block (Block 14 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.	yes, but was M3
	23:20	**Intra Predictor Mode for Neighbor B11 (IntraMxMPredModeB11):** This field carries the intra prediction mode of the second bottom 4x4 block (Block 11 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition	yes, but was M3

DWord	Bit	Description	Same as Prev. Gen?
		of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.	
	19:16	**Intra Predictor Mode for Neighbor B10 (IntraMxMPredModeB10):** This field carries the intra prediction mode of the first bottom 4x4 block (Block 10 in Numbers of Block4x4 in a 16x16 region)of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.	yes, but was M3
	15:12	**Intra Predictor Mode for Neighbor A15 (IntraMxMPredModeA15):** This field carries the intra prediction mode of the fourth rightmost 4x4 block (Block 15 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.	yes, but was M3
	11:8	**Intra Predictor Mode for Neighbor A13 (IntraMxMPredModeA13):** This field carries the intra prediction mode of the third rightmost 4x4 block (Block 13 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.	yes, but was M3
	7:4	**Intra Predictor Mode for Neighbor A7 (IntraMxMPredModeA7):** This field carries the intra prediction mode of the second rightmost 4x4 block (Block 7 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A.	yes, but was M3
	3:0	**Intra Predictor Mode for Neighbor A5 (IntraMxMPredModeA5):** This field carries the intra prediction mode of the first rightmost 4x4 block (Block 5 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification. Intra Predictor Modes for Neighbor A and B are only used if **MODE_INTRA_NOPRED** is not zero. For intra mode selection, bias is applied to the predicted mode if a predictor is present for a partition. This is achieved by applying a penalty term MODE_INTRA_NONPRED defined in the VME state to the cost functions for non-predicted modes. The predictor for a given partition is from its left neighbor and top neighbor. The intra decision for a partition serves as the predictor for the next partition in the partition order as defined in Numbers of Block4x4 in a 16x16 region and Numbers of Block4x4 in an 8x8 region or numbers of Block8x8 in a 16x16 region. This set of intra predictor mode for neighbor macroblocks are only used for INTRA8x8 and INTRA4x4 modes. Format : U4 (The value of this field is defined in Definition of Intra4x4PredMode which is the same as that in Definition of Intra8x8PredMode.)	yes, but was M3
M4.3	31:24	**Corner Neighbor pixel 1.** Its content depends on **IntraCornerSwap** field. It swaps with **Corner Neighbor pixel 0**. **Neighbor pixel Luma value [-1, -1]**. The one upper-left edge pixel from neighbor macroblock D, which is the right most edge pixel of D, if **IntraCornerSwap** field is **1**. Or **Neighbor pixel Luma value [-1, 15]**. The last left edge pixel from neighbor macroblock A, which is the left most edge pixel of D, if **IntraCornerSwap** field is 0.	yes, but was M3
	23:0	**Neighbor pixel Luma value [-1, 14] to [-1, 12].** Left edge pixels from neighbor	yes, but was M3

DWord	Bit	Description	Same as Prev. Gen?
		macroblock A	
M4.2	31:0	**Neighbor pixel Luma value [-1, 11] to [-1, 8].** Left edge pixels from neighbor macroblock A	yes, but was M3
M4.1	31:0	**Neighbor pixel Luma value [-1, 7] to [-1, 4].** Left edge pixels from neighbor macroblock A	yes, but was M3
M4.0	31:24	**Neighbor pixel Luma value [-1, 3].** Fourth left edge pixel from neighbor macroblock A	yes, but was M3
	23:16	**Neighbor pixel Luma value [-1, 2].** Third left edge pixel from neighbor macroblock A	yes, but was M3
	15:8	**Neighbor pixel Luma value [-1, 1].** Second left edge pixel from neighbor macroblock A	yes, but was M3
	7:0	**Neighbor pixel Luma value [-1, 0].** First left edge pixel from neighbor macroblock A	yes, but was M3

3.5.4 Writeback Message

In order to minimize kernel software overhead, the PLACEMENTS of the bit-fields as well as the words/dwords are specifically designed to match with the inline data of the MFC_PAK_OBJECT command of MFX.

DWord	Bit	Description	Same as Prev. Gen?
W0.7	31:28	**VME Decisions – Other:** These 4 bits are used to expose internal behavior of VME to the kernel, specifically whether or not FME or BME had a positive impact, whether or not the ExtraCandidate adds any value to be checked, and whether or not the MaxMV value limited partitioning to a larger shape decision. xxx1: After FME, the primary candidate's distortion was improved. xx1x: After BME, the primary candidate's distortion was improved. x1xx: When VME concludes, the ExtraCandidate ends up beating the initial primary candidate. 1xxx: The MaxMV value restricted the final partition decision (VME would have picked a more detailed shape, but couldn't due to motion vector constraint).). This field only applies to the final partition decision of the main partitioning or candidate and not the alternate candidate. It is only valid incase of SrcSize 16x16. Otherwise it is MBZ.	yes
	27:23	**VME Decisions – Early Exit Conditions:** These 5 bits expose to the kernel that VME finished prior to completing all subfunctions and for what early exit criteria this occurred. Note, these values are only set when the VmeFlag "EarlySuccess" is enabled. xxxx1: EarlySkipExit Occurred xxx1x: EarlyImeStop Occurred	yes

DWord	Bit	Description	Same as Prev. Gen?
		xx1xx: ImeTooGood Occurred x1xxx: ImeTooBad Occurred 1xxxx: EarlyFmeExit Occurred	
	22:16	**VME Decisions – Sub-Functions Performed:** These 7 bits expose to the kernel which sub-functions VME performed. Also, each sub-function is explicitly listed for primary or extra candidate for FME and BME. There is some redundancy with respect to Skipcheck and Intra based on input state to VME. xxxxxx1: Performed Skipcheck xxxxx1x: Performed IME xxxx1xx: Performed FME on primary xxx1xxx: Performed FME on extra candidate xx1xxxx: Performed BME on primary x1xxxxx: Performed BME on extra candidate	yes
	15:8	**Sub-Macroblock Prediction Mode (SubMbPredMode):** 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]	yes
	7:0	**Sub-Macroblock Shape (SubMbShape):** This field describes the subdivision of the four 8x8 sub-macroblocks. 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 or equivalent macroblock. This field is forced to 0 for a non-BP_8x8 inter macroblock, and effectively carries redundant information as **MbType**). This field is only valid If **InterMbMode** is INTER8x8, Otherwise, it is set to zero. Bits [1:0]: SubMbShape[0] Bits [3:2]: SubMbShape[1] Bits [5:4]: SubMbShape[2] Bits [7:6]: SubMbShape[3]	yes

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th><th>Same as Prev. Gen?</th></tr>
<tr><td>W0.6</td><td>31:26</td><td>Alternate Search Path Length: Counts the number of unique search units computed by VME for the alternate search path for dual reference or dual search path. If the search path would return to a previously processed SU, it would not be reprocessed and hence not recounted. The value of [W0.1 15:8] is the overall total search units processed from both paths whereas this value is the contribution only from the second search path. Note: Whenever VME is in a mode that processes only a single search path, this field will be 0x0.

Format: U6, Range of 0-48</td><td>yes</td></tr>
<tr><td></td><td>25:16</td><td>Total VME Stalled Clocks by 16: Counts the number of clocks VME is stalled\starved while processing this request, due to cache misses. The result is returned in units of 16 clock intervals. If the maximum value is returned, the full range was exceeded and the value clipped to the max (this is very unlikely).

Format: U10, Range of 0-1023 [logical range of 0-16383 in 16 clock intervals]</td><td>yes</td></tr>
<tr><td></td><td>15:8</td><td>Total VME Compute Clocks by 16: Counts the number of clocks VME is processing this request, but not stalled\starved as a result of cache misses. The result is returned in units of 16 clock intervals. If the maximum value is returned, the full range was exceeded and the value clipped to the max (this is very unlikely).</td><td>yes</td></tr>
<tr><td></td><td>7:0</td><td>Macroblock Intra Structure (MbIntraStruct): This is a bitmask specifies neighbor macroblock availability. This allows software to constrain intra prediction mode search.

This field is simply copied from the input message (to reduce software overhead of forming the output message to PAK).
<table>
<tr><th>Bits</th><th>MotionVerticalFieldSelect Index</th></tr>
<tr><td>7</td><td>Reserved : MBZ (for IntraPredAvailFlagF – F (pixel[-1,7] available for MbAff)</td></tr>
<tr><td>6</td><td>Reserved : MBZ (for IntraPredAvailFlagA/E – A (left neighbor top half for MbAff)</td></tr>
<tr><td>5</td><td>IntraPredAvailFlagE/A – A (Left neighbor or Left bottom half)</td></tr>
<tr><td>4</td><td>IntraPredAvailFlagB – B (Upper neighbor)</td></tr>
<tr><td>3</td><td>IntraPredAvailFlagC – C (Upper left neighbor)</td></tr>
<tr><td>2</td><td>IntraPredAvailFlagD – D (Upper right neighbor)</td></tr>
<tr><td>1:0</td><td>Reserved : MBZ (for ChromaIntraPredMode)</td></tr>
</table></td><td>yes</td></tr>
<tr><td>W0.5</td><td>31:16</td><td>LumaIntraPredModes[3]</td><td>yes</td></tr>
</table>

DWord	Bit	Description	Same as Prev. Gen?
		Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.	
	15:0	**LumaIntraPredModes[2]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.	yes
W0.4	31:16	**LumaIntraPredModes[1]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each.	yes
	15:0	**LumaIntraPredModes[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.	yes
W0.3	31:28	**Direct8x8Pattern** 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. 0 in a bit – Corresponding MVs are sent in the bitstream 1 in a bit – Corresponding MVs are not sent in the bitstream	yes
	27:14	Reserved: MBZ	yes
	13:0	**BestIntraDistortion** The IntraMbMode will indicate if this is a 16x16/8x8/4x4 distortion. Format = U14	yes
W0.2	31	Reserved: MBZ	yes
	30	**SkipRawDistortionInvalid** Format = U14	yes
	29:16	**SkipRawDistortion** Format = U14	yes
	15:14	Reserved: MBZ	yes
	13:0	**InterDistortion**	yes

DWord	Bit	Description	Same as Prev. Gen?
		Format = U14	
W0.1	31:30	Reserved: MBZ	yes
	29:16	**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. Format = U14	yes
	15:8	**Search Path Length:** This field returns the number of SU it takes in the integer search. It includes predetermined search path and dynamic search path. Format: U8	yes
	7:4	**Reference 1 border reached:** bitmask indicating whether any border of reference 1 is reached by one or more motion vectors in the winning inter mode. xxx1: left border reached xx1x: right border reached x1xx: top border reached 1xxx: bottom border reached	yes
	3:0	**Reference 0 border reached:** bitmask indicating whether any border of reference 0 is reached by one or more motion vectors in the winning inter mode. xxx1: left border reached xx1x: right border reached x1xx: top border reached 1xxx: bottom border reached	yes
W0.0	31	**ExtendedForm (IVB only)** This field specifies that **LumaIntraMode**'s are fully replicated in 4x4 sub-blocks respectively. And motion vectors must be in unpacked form as well. This non-DXVA form is used for optimal kernel performance.	yes
	30:29	Reserved: MBZ	
	28:24	**MvQuantity** Specify the number of MVs in packed format (in unit of motion vectors). *Note: this field is provided to help with software to meet conformance requirements such as maximum number of motion vectors for two consecutive macroblocks.* Format: U5, valid from 0 to 32	yes
	23	**Reserved : MBZ** **(reserved for ExternalMvBufFlag.** It is always **0** in this case, since MV's are included)	yes
	22:20	**MvSize (Motion Vector Size)**. This field specifies the size and format of the output motion vectors.	yes

DWord	Bit	Description	Same as Prev. Gen?
		This field is reserved (MBZ) when the output signal **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 **MvQuantity**.)*	
	19	**DcBlockCodedYFlag.** This field specifies if the Luma DC sub-block is coded. 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. VME hardware forces this output to be 1.	yes
	18	**DcBlockCodedCbFlag.** This field specifies if the Chroma Cb DC sub-block is coded. 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. VME hardware forces this output to be 1.	yes
	17	**DcBlockCodedCrFlag.** This field specifies if the Chroma Cb DC sub-block is coded**.** 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. VME hardware forces this output to be 1.	yes
	16	Reserved: MBZ	yes
	15	**Transform8x8Flag (Transform 8x8 Flag)** This field indicates that 8x8 transform is recommended. It is set to 1 if **IntraMbFlag** = INTRA and **IntraMbMode** = INTRA_8x8. For **IntraMbFlag** = INTER. If **T8x8FlagForInterEn** = 0, this field is set to 0 by VME hardware. If **T8x8FlagForInterEn** = 1, this field is set to 1 if there is no sub macroblock size less than 8x8 (noSubMbPartSizeLessThan8x8Flag = 1).	yes

DWord	Bit	Description	Same as Prev. Gen?
		0: 4x4 integer transform 1: 8x8 integer transform Note: This bit will be always 0 for non-16x16 source block cases.	
	14	**FieldMbFlag** This field indicates the inter prediction result is field or frame. It is always set to **SrcAccess**. 0: frame macroblock 1: field macroblock	yes
	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. Even though I_PCM is considered as Intra MB, VME hardware cannot generate I_PCM output. 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)	yes
	12:8	**MbType** This field is encoded to match with the best macroblock type determined as described in the next section. It follows an unified encoding for inter and intra macroblocks according to AVC Spec.	yes
	7	**FieldMbPolarityFlag** This field indicates the field polarity of the current macroblock. Unique for AVC standard, within an MbAff frame picture, this field may be different per macroblock and is set to 1 only for the second macroblock in an MbAff pair if FieldMbFlag is set. Otherwise, it is set to 0. Within a field picture in most coding standard, this field is a constant for the whole field picture. It is set to 1 if the current picture is the bottom field picture. Otherwise, it is set to 0. This field is reserved and MBZ for a progressive frame picture. VME hardware set this field to 1 if the source block is a field block from the bottom field and otherwise sets it to 0. This is accomplished by the following equation using input signals **SrcAccess** and **SrcY**: **SrcAccess** && (bit0(**SrcY**) ==1). 0 = Current macroblock is a field macroblock from the **top** field 1 = Current macroblock is a field macroblock from the **bottom** field	yes
	6	Reserved: MBZ	yes
	5:4	**IntraMbMode**	yes

DWord	Bit	Description	Same as Prev. Gen?
		This field is provided to carry redundant information as that in **MbType**. The full extended definition of this field allows kernel software to help update the **MbType** field when outputting controls to the MFX PAK encoding. VME outputs this field regardless of MbIntraFlag value if intra mode is enabled.	
	3	Reserved: MBZ	yes
	2	**MbSkipFlag** As an output of VME, this bit indicates whether one skip center (possibly of several skip centers for each partition) is the winning motion vector position. VME outputs this field regardless of MbIntraFlag value. *Note that the meaning of this field in VME is not the same as that used in PAK.*	yes
	1:0	**InterMbMode** This field is provided to carry redundant information as that in **MbType**. The full extended definition of this field allows kernel software to help update the **MbType** field when outputting controls to the MFX PAK encoding. VME outputs this field regardless of MbIntraFlag value if inter mode is enabled.	yes
W1.7 to W1.2	31:0 Each	**MVb[3]** to **MVb[1]**. Motion vectors 3 to 1 for Reference 1, and **MVa[3]** to **MVa[1]**. Motion vectors 3 to 1 for Reference 0	no
W1.1	31:16	**MVb[0].y:** returning the y-coordinate of Motion Vector 0 for Reference 1, relative to source MB location. Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]	no
	15:0	**MVb[0].x:** returning the x-coordinate of Motion Vector 0 (co-located w/ sublbock_4x4_0) for Reference 1, relative to source MB location. Its meaning is determined by **MbType**. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]	no
W1.0	31:16	**MVa[0].y:** returning the y-coordinate of Motion Vector 0 for Reference 0, relative to source MB location. Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]	no
	15:0	**MVa[0].x:** returning the x-coordinate of Motion Vector 0 (co-located w/ sublbock_4x4_0) for Reference 0, relative to source MB location. Its meaning is determined by **MbType**. The returned motion vectors are placed in a fixed data format, with up to 16 motion vectors for one reference and the motion vectors from reference 0 and 1 interleaved. Format = S13.2 (2's comp)	no

DWord	Bit	Description	Same as Prev. Gen?
		Hardware Range: [-2048.00 to 2047.75]	
W2.7 to W2.0	31:0 Each	**MVb[7]** to **MVb[4]**. Motion vectors 7 to 4 for Reference 1, and **MVa[7]** to **MVa[4]**. Motion vectors 7 to 4 for Reference 0	no
W3.7 to W3.0	31:0 Each	**MVb[11]** to **MVb[8]**. Motion vectors 11 to 8 for Reference 1, and **MVa[11]** to **MVa[8]**. Motion vectors 11 to 8 for Reference 0	no
W4.7 to W4.0	31:0 Each	**MVb[15]** to **MVb[12]**. Motion vectors 15 to 12 for Reference 1, and **MVa[15]** to **MVa[12]**. Motion vectors 15 to 12 for Reference 0	no
W5.7 to W5.1	31:0 Each	**InterDistortion[15] to InterDistortion[2]**. Inter-prediction-distortion associated with motion vector 15 to 2. Its meaning is determined by sub-shape**.**	yes, but was M3
W5.0	31:30	Reserved: MBZ	yes, but was M3
	29:16	**InterDistortion[1].** Inter-prediction-distortion with motion vector 1 (co-located with subblock_4x4_1). Its meaning is determined by sub-shape**.** Format = U14	yes, but was M3
	15:14	Reserved: MBZ	yes, but was M3
	13:0	**InterDistortion[0].** Inter-prediction-distortion associated with motion vector 0 (co-located with subblock_4x4_0). Its meaning is determined by sub-shape. It must be zero if the corresponding sub-shape is not chosen. This field may be associated with MVa[0] and/or MVb[0], depending on the resulting prediction mode for the sub-block. If the corresponding MV field is created by "duplication", this field must be zero. Format = U14	yes, but was M3

1. mv_format_pic = vin_mv_format * vin_codec_select
2. Change vin_mvunpackenable to (vin_mvunpackenable + mv_format_pic) on all location.
3. extended_form_pic = vin_extended_form * vin_codec_select
4. (vctrl_it_Transform8x8Flag * !extended_form_pic) ? "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_it_lumaintrapredmode3[15:0] & vctrl_it_lumaintrapredmode2

MV Fub:

Mux Output Table

		Select:	ref_index_rep_size[3:0]
Value	Output	Input	Description

0001	enc_refidx_L0addr_B0_int[4:0]	enc_mode_bind_fwd_B0[4:0]	
0001	enc_refidx_L0addr_B1_int[4:0]	enc_mode_bind_fwd_B1[4:0]	
0001	enc_refidx_L0addr_B2_int[4:0]	enc_mode_bind_fwd_B2[4:0]	
0001	enc_refidx_L0addr_B3_int[4:0]	enc_mode_bind_fwd_B3[4:0]	
0001	enc_refidx_L1addr_B0_int[4:0]	enc_mode_bind_bkd_B0[4:0]	
0001	enc_refidx_L1addr_B1_int[4:0]	enc_mode_bind_bkd_B1[4:0]	
0001	enc_refidx_L1addr_B2_int[4:0]	enc_mode_bind_bkd_B2[4:0]	
0001	enc_refidx_L1addr_B3_int[4:0]	enc_mode_bind_bkd_B3[4:0]	
0010	enc_refidx_L0addr_B0_int[4:0]	enc_mode_bind_fwd_B0[4:0]	
0010	enc_refidx_L0addr_B1_int[4:0]	enc_mode_bind_fwd_B1[4:0]	
0010	enc_refidx_L0addr_B2_int[4:0]	enc_mode_bind_fwd_B0[4:0]	
0010	enc_refidx_L0addr_B3_int[4:0]	enc_mode_bind_fwd_B1[4:0]	
0010	enc_refidx_L1addr_B0_int[4:0]	enc_mode_bind_bkd_B0[4:0]	
0010	enc_refidx_L1addr_B1_int[4:0]	enc_mode_bind_bkd_B1[4:0]	
0010	enc_refidx_L1addr_B2_int[4:0]	enc_mode_bind_bkd_B0[4:0]	
0010	enc_refidx_L1addr_B3_int[4:0]	enc_mode_bind_bkd_B1[4:0]	
0100	enc_refidx_L0addr_B0_int[4:0]	enc_mode_bind_fwd_B0_ext[4:0]	
0100	enc_refidx_L0addr_B1_int[4:0]	enc_mode_bind_fwd_B0_ext[4:0]	
0100	enc_refidx_L0addr_B2_int[4:0]	enc_mode_bind_fwd_B1_ext[4:0]	
0100	enc_refidx_L0addr_B3_int[4:0]	enc_mode_bind_fwd_B1_ext[4:0]	
0100	enc_refidx_L1addr_B0_int[4:0]	enc_mode_bind_bkd_B0_ext[4:0]	
0100	enc_refidx_L1addr_B1_int[4:0]	enc_mode_bind_bkd_B0_ext[4:0]	
0100	enc_refidx_L1addr_B2_int[4:0]	enc_mode_bind_bkd_B1_ext[4:0]	
0100	enc_refidx_L1addr_B3_int[4:0]	enc_mode_bind_bkd_B1_ext[4:0]	
1000	enc_refidx_L0addr_B0_int[4:0]	enc_mode_bind_fwd_B0[4:0]	
1000	enc_refidx_L0addr_B1_int[4:0]	enc_mode_bind_fwd_B0[4:0]	
1000	enc_refidx_L0addr_B2_int[4:0]	enc_mode_bind_fwd_B0[4:0]	
1000	enc_refidx_L0addr_B3_int[4:0]	enc_mode_bind_fwd_B0[4:0]	
1000	enc_refidx_L1addr_B0_int[4:0]	enc_mode_bind_bkd_B0[4:0]	
1000	enc_refidx_L1addr_B1_int[4:0]	enc_mode_bind_bkd_B0[4:0]	
1000	enc_refidx_L1addr_B2_int[4:0]	enc_mode_bind_bkd_B0[4:0]	
1000	enc_refidx_L1addr_B3_int[4:0]	enc_mode_bind_bkd_B0[4:0]	

Combinatorial Signals Table

Signal	Equation
extended_form_pic	vin_extended_form * vin_codec_selectR
enc_mode_bind_fwd_B0_ext[4:0]	extended_form_pic? enc_mode_bind_fwd_B0[4:0] : enc_mode_bind_fwd_B0[4:0]
enc_mode_bind_fwd_B1_ext[4:0]	extended_form_pic? enc_mode_bind_fwd_B2[4:0] : enc_mode_bind_fwd_B1[4:0]
enc_mode_bind_bkd_B0_ext[4:0]	extended_form_pic? enc_mode_bind_bkd_B0[4:0] : enc_mode_bind_bkd_B0[4:0]
enc_mode_bind_bkd_B1_ext[4:0]	extended_form_pic? enc_mode_bind_bkd_B2[4:0] : enc_mode_bind_bkd_B1[4:0]

3.5.5 Stream-in\Stream-out Message

Each reference will require 2 message phases when performing multi-call. These phases will be added onto the basic input or output message. Hence, the first stream-in or stream-out message phase location is variable and represented below by M(*X+?*), where X equals the number of phases present in the input or output message, respectively.

When both records are being streamed in or out, phases M+0 and M+1 will contain record0 (associated with RefA) and M+2 and M+3 will contain record1 (associated with RefB). If there is only one reference being searched (SearchControl != 111b) then only one record will be streamed in or out, specifically, only M+0 and M+1 will be present.

DWord	Bit	Description
M(X+0).7	31:0	Reserved MBZ
M(X+0).6	31:0	Reserved MBZ
M(X+0).5	31:16	Rec0 Shape 16x16 Y (relative to source MB) Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]
	15:0	Rec0 Shape 16x16 X (relative to source MB) Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]
M(X+0).4	31:16	Reserved MBZ
	15:14	Reserved MBZ
	13:0	Rec0 Shape 16x16 Distortion Format = U14
M(X+0).3	31:16	Rec0 Shape 8x8_3 Distortion Hardware only uses 14 bits. Upper bits ignored (True for all 8x8_X Distortions).
	15:0	Rec0 Shape 8x8_2 Distortion
M(X+0).2	31:16	Rec0 Shape 8x8_1 Distortion
	15:0	Rec0 Shape 8x8_0 Distortion
M(X+0).1	31:16	Rec0 Shape 8x16_1 Distortion Hardware only uses 15 bits. Upper bits ignored (True for all 8x16_X Distortions).
	15:0	Rec0 Shape 8x16_0 Distortion
M(X+0).0	31:16	Rec0 Shape 16x8_1 Distortion Hardware only uses 15 bits. Upper bits ignored (True for all 16x8_X Distortions).
	15:0	Rec0 Shape 16x8_0 Distortion
M(X+1).7	31:16	Rec0 Shape 8x8_3 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_3 X (relative to source MB)
M(X+1).6	31:16	Rec0 Shape 8x8_2 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_2 X (relative to source MB)

DWord	Bit	Description
M(X+1).5	31:16	Rec0 Shape 8x8_1 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_1 X (relative to source MB)
M(X+1).4	31:16	Rec0 Shape 8x8_0 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_0 X (relative to source MB)
M(X+1).3	31:16	Rec0 Shape 8x16_1 Y (relative to source MB)
	15:0	Rec0 Shape 8x16_1 X (relative to source MB)
M(X+1).2	31:16	Rec0 Shape 8x16_0 Y (relative to source MB)
	15:0	Rec0 Shape 8x16_0 X (relative to source MB)
M(X+1).1	31:16	Rec0 Shape 16x8_1 Y (relative to source MB)
	15:0	Rec0 Shape 16x8_1 X (relative to source MB)
M(X+1).0	31:16	Rec0 Shape 16x8_0 Y (relative to source MB)
	15:0	Rec0 Shape 16x8_0 X (relative to source MB)
M(X+2).7	31:0	Reserved MBZ
M(X+2).6	31:0	Reserved MBZ
M(X+2).5	31:16	Rec1 Shape 16x16 Y (relative to source MB) Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]
	15:0	Rec1 Shape 16x16 X (relative to source MB) Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]
M(X+2).4	31:16	Reserved MBZ
	15:14	Reserved MBZ
	13:0	Rec1 Shape 16x16 Distortion Format = U14
M(X+2).3	31:16	Rec1 Shape 8x8_3 Distortion Hardware only uses 14 bits. Upper bits ignored (True for all 8x8_X Distortions).
	15:0	Rec1 Shape 8x8_2 Distortion
M(X+2).2	31:16	Rec1 Shape 8x8_1 Distortion
	15:0	Rec1 Shape 8x8_0 Distortion
M(X+2).1	31:16	Rec1 Shape 8x16_1 Hardware only uses 15 bits. Upper bits ignored (True for all 8x16_X Distortions).
	15:0	Rec1 Shape 8x16_0 Distortion
M(X+2).0	31:16	Rec1 Shape 16x8_1 Distortion Hardware only uses 15 bits. Upper bits ignored (True for all 16x8_X Distortions).
	15:0	Rec1 Shape 16x8_0 Distortion
M(X+3).7	31:16	Rec1 Shape 8x8_3 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_3 X (relative to source MB)
M(X+3).6	31:16	Rec1 Shape 8x8_2 Y (relative to source MB)

DWord	Bit	Description
	15:0	Rec1 Shape 8x8_2 X (relative to source MB)
M(X+3).5	31:16	Rec1 Shape 8x8_1 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_1 X (relative to source MB)
M(X+3).4	31:16	Rec1 Shape 8x8_0 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_0 X (relative to source MB)
M(X+3).3	31:16	Rec1 Shape 8x16_1 Y (relative to source MB)
	15:0	Rec1 Shape 8x16_1 X (relative to source MB)
M(X+3).2	31:16	Rec1 Shape 8x16_0 Y (relative to source MB)
	15:0	Rec1 Shape 8x16_0 X (relative to source MB)
M(X+3).1	31:16	Rec1 Shape 16x8_1 Y (relative to source MB)
	15:0	Rec1 Shape 16x8_1 X (relative to source MB)
M(X+3).0	31:16	Rec1 Shape 16x8_0 Y (relative to source MB)
	15:0	Rec1 Shape 16x8_0 X (relative to source MB)

4. Shared Functions Pixel Interpolater

The Pixel Interpolator provides barycentric parameters at various offsets relative to the pixel location. These barycentric parameters are in the same format and layout as those received in the pixel shader dispatch. Please refer to the “Windower” chapter in the “3D Pipeline” volume for more details on barycentric parameters.

Barycentric parameters delivered in the pixel shader payload are at pre-defined positions based on **Barycentric Interpolation Mode** bits selected in 3DSTATE_WM. The pixel interpolator allows barycentric parameters to be computed at additional locations.

4.1 Messages

The following is the message definition for the Pixel Interpolator shared function.

Restrictions:

- Pixel Interpolator messages can only be delivered by pixel shader kernels.
- [ivbgt2 pre-K0] Hang possible if linear PI message when Barycentric Interpolation mode has any perspective bits set, or Pixel Shader Uses Source W is set.
- [ivbgt2 pre-K0] Hang possible if perspective PI message when Barycentric Interpolation mode has any non-perspective bits set.

Execution Mask. Each bit in the execution mask enables the corresponding slot’s barycentric parameter return to the destination registers.

4.1.1 Initiating Message

4.1.1.1 Message Descriptor

Bit	Description
19	**Header Present:** Specifies whether the message includes a header phase. Must be zero for all *Pixel Interpolator* messages. Format = Enable
18:17	Ignored
16	**SIMD Mode.** Specifies the SIMD mode of the message being sent. Format = U1 0: SIMD8 mode 1: SIMD16 mode
15	Ignored
14	**Interpolation Mode.** Specifies which interpolation mode is to be used. Format = U1 0: Perspective Interpolation

Bit	Description
	1: Linear Interpolation **Programming Notes:** • This field cannot be set to “Linear Interpolation” unless **Non-Perspective Barycentric Enable** in 3DSTATE_CLIP is enabled.
13:12	**Message Type.** Specifies the type of message being sent. Format = U2 0: Per Message Offset (eval_snapped with immediate offset) 1: Sample Position Offset (eval_sindex) 2: Centroid Position Offset (eval_centroid) 3: Per Slot Offset (eval_snapped with register offset)
11	**Slot Group Select.** This field selects whether slots 15:0 or slots 31:16 are used for bypassed data. Bypassed data includes the X/Y addresses and centroid position. For 8- and 16-pixel dispatches, SLOTGRP_LO must be selected on every message. For 32-pixel dispatches, this field must be set correctly for each message based on which slots are currently being processed. 0: SLOTGRP_LO:choose bypassed data for slots 15:0 1: SLOTGRP_HI:choose bypassed data for slots 31:16 **Programming Notes:** This field must be set to SLOTGRP_LO for SIMD8 messages. SIMD8 messages always use bypassed data for slots 7:0.
10:8	Ignored
7:0	**Message Specific Control.** Refer to the sections below for the definition of these bits based on **Message Type**.

4.1.1.1.1 “Per Message Offset” Message Descriptor

Bit	Description
7:4	**Per Message Y Pixel Offset** Specifies the Y Pixel Offset that applies to all slots. Format = S4 2’s complement representing units of 1/16 pixel. Range = [-8/16, +7/16]
3:0	**Per Message X Pixel Offset** Specifies the X Pixel Offset that applies to all slots. Format = S4 2’s complement representing units of 1/16 pixel. Range = [-8/16, +7/16]

4.1.1.1.2 "Sample Position Offset" Message Descriptor

Bit	Description
7:4	**Sample Index** Specifies the sample index that applies to all slots. Format = U4 Range = [0,7
3:0	Ignored

4.1.1.1.3 "Centroid Position" and "Per Slot Offset" Message Descriptor

Bit	Description
7:0	Ignored

4.1.1.2 Message Payload for most messages

This message payload applies to the following message types:

- Per Message Offset
- Sample Position Offset
- Centroid Position Offset

DWord	Bit	Description
M0.7:0		Ignored

4.1.1.3 SIMD8 Per Slot Offset Message Payload

This message payload applies only to the SIMD8 Per Slot Offset message type. The message length is 2.

DWord	Bit	Description
M0.7	31:0	**Slot 7 X Pixel Offset** Specifies the X pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M0.6	31:0	**Slot 6 X Pixel Offset**
M0.5	31:0	**Slot 5 X Pixel Offset**
M0.4	31:0	**Slot 4 X Pixel Offset**
M0.3	31:0	**Slot 3 X Pixel Offset**
M0.2	31:0	**Slot 2 X Pixel Offset**
M0.1	31:0	**Slot 1 X Pixel Offset**
M0.0	31:0	**Slot 0 X Pixel Offset**

DWord	Bit	Description
M1.7	31:0	**Slot 7 Y Pixel Offset** Specifies the Y pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M1.6	31:0	**Slot 6 Y Pixel Offset**
M1.5	31:0	**Slot 5 Y Pixel Offset**
M1.4	31:0	**Slot 4 Y Pixel Offset**
M1.3	31:0	**Slot 3 Y Pixel Offset**
M1.2	31:0	**Slot 2 Y Pixel Offset**
M1.1	31:0	**Slot 1 Y Pixel Offset**
M1.0	31:0	**Slot 0 Y Pixel Offset**

4.1.1.4 SIMD16 Per Slot Offset Message Payload

This message payload applies only to the SIMD16 Per Slot Offset message type. The message length is 4.

DWord	Bit	Description
M0.7	31:0	**Slot 7 X Pixel Offset** Specifies the X pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M0.6	31:0	**Slot 6 X Pixel Offset**
M0.5	31:0	**Slot 5 X Pixel Offset**
M0.4	31:0	**Slot 4 X Pixel Offset**
M0.3	31:0	**Slot 3 X Pixel Offset**
M0.2	31:0	**Slot 2 X Pixel Offset**
M0.1	31:0	**Slot 1 X Pixel Offset**
M0.0	31:0	**Slot 0 X Pixel Offset**
M1.7	31:0	**Slot 15 X Pixel Offset**

DWord	Bit	Description
M1.6	31:0	**Slot 14 X Pixel Offset**
M1.5	31:0	**Slot 13 X Pixel Offset**
M1.4	31:0	**Slot 12 X Pixel Offset**
M1.3	31:0	**Slot 11 X Pixel Offset**
M1.2	31:0	**Slot 10 X Pixel Offset**
M1.1	31:0	**Slot 9 X Pixel Offset**
M1.0	31:0	**Slot 8 X Pixel Offset**
M2.7	31:0	**Slot 7 Y Pixel Offset** Specifies the Y pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M2.6	31:0	**Slot 6 Y Pixel Offset**
M2.5	31:0	**Slot 5 Y Pixel Offset**
M2.4	31:0	**Slot 4 Y Pixel Offset**
M2.3	31:0	**Slot 3 Y Pixel Offset**
M2.2	31:0	**Slot 2 Y Pixel Offset**
M2.1	31:0	**Slot 1 Y Pixel Offset**
M2.0	31:0	**Slot 0 Y Pixel Offset**
M3.7	31:0	**Slot 15 Y Pixel Offset**
M3.6	31:0	**Slot 14 Y Pixel Offset**
M3.5	31:0	**Slot 13 Y Pixel Offset**
M3.4	31:0	**Slot 12 Y Pixel Offset**
M3.3	31:0	**Slot 11 Y Pixel Offset**
M3.2	31:0	**Slot 10 Y Pixel Offset**
M3.1	31:0	**Slot 9 Y Pixel Offset**
M3.0	31:0	**Slot 8 Y Pixel Offset**

4.1.2 Writeback Message

4.1.2.1 SIMD8

The response length for all SIMD8 messages is 2. The data for each slot is written only if its corresponding execution mask bit is set.

DWord	Bit	Description
W0.7	31:0	**Barycentric[1] for Slot 7** Format = IEEE_Float
W0.6	31:0	**Barycentric[1] for Slot 6**
W0.5	31:0	**Barycentric[1] for Slot 5**
W0.4	31:0	**Barycentric[1] for Slot 4**
W0.3	31:0	**Barycentric[1] for Slot 3**
W0.2	31:0	**Barycentric[1] for Slot 2**
W0.1	31:0	**Barycentric[1] for Slot 1**
W0.0	31:0	**Barycentric[1] for Slot 0**
W1.7	31:0	**Barycentric[2] for Slot 7** Format = IEEE_Float
W1.6	31:0	**Barycentric[2] for Slot 6**
W1.5	31:0	**Barycentric[2] for Slot 5**
W1.4	31:0	**Barycentric[2] for Slot 4**
W1.3	31:0	**Barycentric[2] for Slot 3**
W1.2	31:0	**Barycentric[2] for Slot 2**
W1.1	31:0	**Barycentric[2] for Slot 1**
W1.0	31:0	**Barycentric[2] for Slot 0**

4.1.2.2 SIMD16

The response length for all SIMD16 messages is 4. The data for each slot is written only if its corresponding execution mask bit is set.

DWord	Bit	Description
W0.7	31:0	**Barycentric[1] for Slot 7** Format = IEEE_Float
W0.6	31:0	**Barycentric[1] for Slot 6**
W0.5	31:0	**Barycentric[1] for Slot 5**
W0.4	31:0	**Barycentric[1] for Slot 4**
W0.3	31:0	**Barycentric[1] for Slot 3**
W0.2	31:0	**Barycentric[1] for Slot 2**
W0.1	31:0	**Barycentric[1] for Slot 1**
W0.0	31:0	**Barycentric[1] for Slot 0**
W1.7	31:0	**Barycentric[2] for Slot 7** Format = IEEE_Float
W1.6	31:0	**Barycentric[2] for Slot 6**
W1.5	31:0	**Barycentric[2] for Slot 5**
W1.4	31:0	**Barycentric[2] for Slot 4**
W1.3	31:0	**Barycentric[2] for Slot 3**
W1.2	31:0	**Barycentric[2] for Slot 2**
W1.1	31:0	**Barycentric[2] for Slot 1**
W1.0	31:0	**Barycentric[2] for Slot 0** Format = IEEE_Float
W2.7	31:0	**Barycentric[1] for Slot 15**
W2.6	31:0	**Barycentric[1] for Slot 14**
W2.5	31:0	**Barycentric[1] for Slot 13**
W2.4	31:0	**Barycentric[1] for Slot 12**

DWord	Bit	Description
W2.3	31:0	**Barycentric[1] for Slot 11**
W2.2	31:0	**Barycentric[1] for Slot 10**
W2.1	31:0	**Barycentric[1] for Slot 9**
W2.0	31:0	**Barycentric[1] for Slot 8**
W3.7	31:0	**Barycentric[2] for Slot 15**
W3.6	31:0	**Barycentric[2] for Slot 14**
W3.5	31:0	**Barycentric[2] for Slot 13**
W3.4	31:0	**Barycentric[2] for Slot 12**
W3.3	31:0	**Barycentric[2] for Slot 11**
W3.2	31:0	**Barycentric[2] for Slot 10**
W3.1	31:0	**Barycentric[2] for Slot 9**
W3.0	31:0	**Barycentric[2] for Slot 8**

Revision History

Revision Number	Description	Revision Date
1.0	First 2012 OpenSource edition	May 2012

§§