# Intel Open Source Graphics Programmer's Reference Manual (PRM) for the 2013 Intel® Core™ Processor Family, including Intel HD Graphics, Intel Iris™ Graphics and Intel Iris Pro Graphics

## Volume 9: Media VEBOX (Haswell)

12/18/2013

## Copyright

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

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

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

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

# Media VEBOX

## Table of Contents

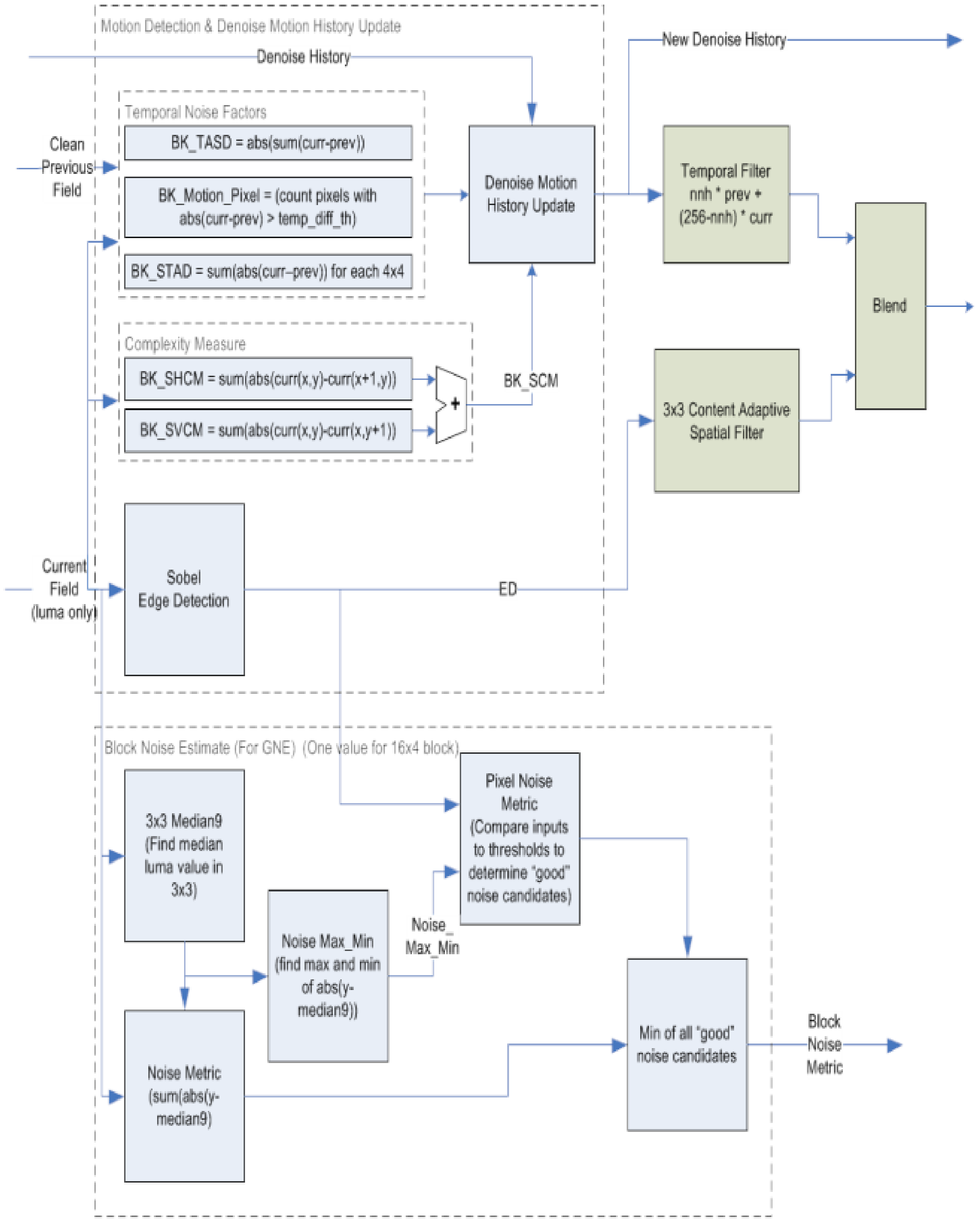
## Denoise

This chapter contains block diagrams and discusses various filters, algorithms and functions that support the Denoise feature in the chipset.

Filters Functions:

- Denoise filter
- Temporal Out-of-Range
- Temporal filter
- Denoise Blend
- Context Adaptive Spatial Filter
- Motion Detection
- Chroma Noise Reduction
- Block Noise Estimate
- Hot Pixel Detection/Correction
- Hot Pixel Algorithm

## Denoise Functional Block Diagram



Motion Detection & Denoise Motion History Update
Denoise History
New Denoise History
Temporal Noise Factors
Clean Previous Field
BK_TASD = abs(sum(curr-prev))
BK_Motion_Pixel = (count pixels with abs(curr-prev) > temp_diff_th)
BK_STAD = sum(abs(curr–prev)) for each 4x4
Denoise Motion History Update
Temporal Filter nnh * prev + (256-nnh) * curr
Blend
Complexity Measure
BK_SHCM = sum(abs(curr(x,y)-curr(x+1,y))
BK_SVCM = sum(abs(curr(x,y)-curr(x,y+1))
+
BK_SCM
3x3 Content Adaptive Spatial Filter
Current Field (luma only)
Sobel Edge Detection
ED
Block Noise Estimate (For GNE) (One value for 16x4 block)
3x3 Median9 (Find median luma value in 3x3)
Pixel Noise Metric (Compare inputs to thresholds to determine "good" noise candidates)
Noise Max_Min (find max and min of abs(y-median9))
Noise_ Max_Min
Noise Metric (sum(abs(y-median9)
Min of all "good" noise candidates
Block Noise Metric

- **Denoise Filter** – detects noise and motion and filters the block with either a temporal filter when little motion is detected or a spatial filter. Noise estimates are kept between frames and blended together. Since the filter is before the deinterlacer it works on individual fields rather than frames. This usually improves the operation since the deinterlacer can take a single pixel of noise and spread it to an adjacent pixel, making it harder to remove. The denoise filter works the same whether deinterlacing or progressive cadence reconstruction is being done. [DevHSW]
- **Chroma Denoise Filter** – detects noise in the U and V planes separately and applies a temporal filter. Noise estimates are kept between frames and blended together.
- **Block Noise Estimate** (BNE) – part of the Global Noise Estimate (GNE) algorithm, this estimates the noise over the entire block. The GNE will be calculated at the end of the frame by combining all the BNEs. The final GNE value is used to control the denoise filter for the next frame.

## Motion Detection and Noise History Update

This block detects motion for the denoise filter, which it then combines with motion detected in the past in the same part of the screen. The Denoise History is both saved to memory and also used to control the temporal denoise filter.

## Temporal Filter

For each pixel, we use previous and current noise history.

## Temporal Out-of-Range

The denoise blend combines the temporal and spatial denoise outputs.

First we check to see if the temporal is out of the local range, if so we use the average of the denoised and the local limit instead:

```
if (temporal_denoised >= block_max)
       temporal_denoised=(temporal_denoised+block_max)>>1;
if (temporal_denoised < block_min)
       temporal_denoised=(temporal_denoised+block_min)>>1;
```

Where block_max and block_min are the largest and smallest luma values in the local 3x3 (can be shared with BNE calculation).

Precision: 8-bit compares and sums [DevHSW] , 8-bit inputs has zeros added to LSB to extend to 12 or 16.

## Denoise Blend

The denoise blend combines the temporal and spatial denoise outputs.

## Chroma Noise Reduction

This chapter contains descriptions of filters that support the chroma noise reduction feature in the chipset.

Filters

- Chroma noise detection
- Temporal filter

### Chroma Noise Detection

The operation of chroma noise detection module is similar to luma noise detection module where the U & V channels are executing individually.

### Chroma Noise Reduction Filter

A simple and effective temporal-domain chroma noise reduction filter is introduced. The Noise History is both saved to memory and also used to control the temporal denoise filter.

### Temporal Filter

For each pixel, we use previous and current noise history.

## Deinterlacer

Features:

- **Film Mode Detection (FMD)** Variances – FMD determines if the input fields were created by sampling film and converting it to interlaced video. If so the deinterlacer is turned off in favor of reconstructing the frame from adjacent fields. Various sum-of-absolute differences are deinterlacer per block. The FMD algorithm is run at the end of the frame by looking at the variances of all blocks for both fields in the frame.
- **Deinterlacer** – Estimates how much motion is deinterlacer across the fields. Low motion scenes are reconstructed by averaging pixels from fields from nearby times (temporal deinterlacer), while high motion scenes are reconstructed by interpolating pixels from nearby space (spatial deinterlacer).
- **Progressive Cadence Reconstruction** – If the FMD for the previous frame determines that film was converted into interlaced video, then this block reconstructs the original frame by directly putting together adjacent fields.
- **Chroma Upsampling** – If the input is 4:2:0 then chroma will be doubled vertically to convert to 4:2:2. Chroma will then either go through it's own version of the d einterlacer or progressive cadence reconstruction.

See the algorithm description in Shared Functions.

## Deinterlacer Algorithm

The overall goal of the motion adaptive deinterlacer is to convert an interlaced video stream made of fields of alternating lines into a progressive video stream made of frames in which every line is provided.

If there is no motion in a scene, then the missing lines can be provided by looking at the previous or next fields, both of which have the missing lines. If there is a great deal of motion in the scene, then objects in the previous and next fields will have moved, so we can't use them for the missing pixels. Instead we have to interpolate from the neighboring lines to fill in the missing pixels. This can be thought of as interpolating in time if there is no motion and interpolating in space if there is motion.

This idea is implemented by creating a measure of motion called the Spatial-Temporal Motion Measure (STMM). If this measure shows that there is little motion in an area around the pixels, then the missing pixels are created by averaging the pixel values from the previous and next frame. If the STMM shows that there is motion, then the missing pixels are filled in by interpolating from neighboring lines with the Spatial Deinterlacer (SDI). The two different ways to interpolate the missing pixels are blended for intermediate values of STMM to prevent sudden transitions.

The deinterlacer uses two frames for reference. The current frame contains the field that we are deinterlacing. The reference frame is the closest frame in time to the field that we are deinterlacing – if we are working on the 1st field then it is the previous frame, if it is the 2nd field then it is the next frame.

## Spatial-Temporal Motion Measure

This algorithm combines a complexity measure with a estimate of motion. This prevents high complexity scenes from incorrectly causing motion to be detected. It is calculated for a set of pixels 2 wide by 1 high.

Complexity is measured in the vertical and horizontal directions with the SVCM and SHCM.

Where c(x,y) is the luma value at that x,y location in the current frame. Note that we are skipping by 2 in the Y direction to ensure that the compares are only done with lines from the same field.

Spatial horizontal complexity measure (SHCM) is a sum of differences in the horizontal direction.

Temporal Difference Measure (TDM) is a measure of differences between pairs of fields with the same lines. It uses filtered versions of c(x,y) from the current frame and r(x,y) from the reference frame (either the previous or next frame).

STMM is then calculated by:

STMM ≈ TDM/Complexity

| **Project:** | |
|---|---|
| STMM = ((TDM >> tdm_shift1) << tdm_shift2) / ((SCM >> 4) + stmm_c2) | |

| **Project:** | |
|---|---|
| since TDM has 13 bits this results in between 9 and 7 bits of precision. Tdm_shift2 can range from 6 to 8, producing a value between 17 and 13 bits, of which only 9-bits are non-zero. The divide can be implemented by an 8-bit reciprocal table followed by a 9-bit x 8-bit multiply by the TDM value, which finally produces an output of 8-bits. | |

STMM is then smoothed with the STMM saved from the previous field:

This process prevents sudden changes in STMM. The smoothed STMM is stored to memory to be read as STMM_s by the next frame.

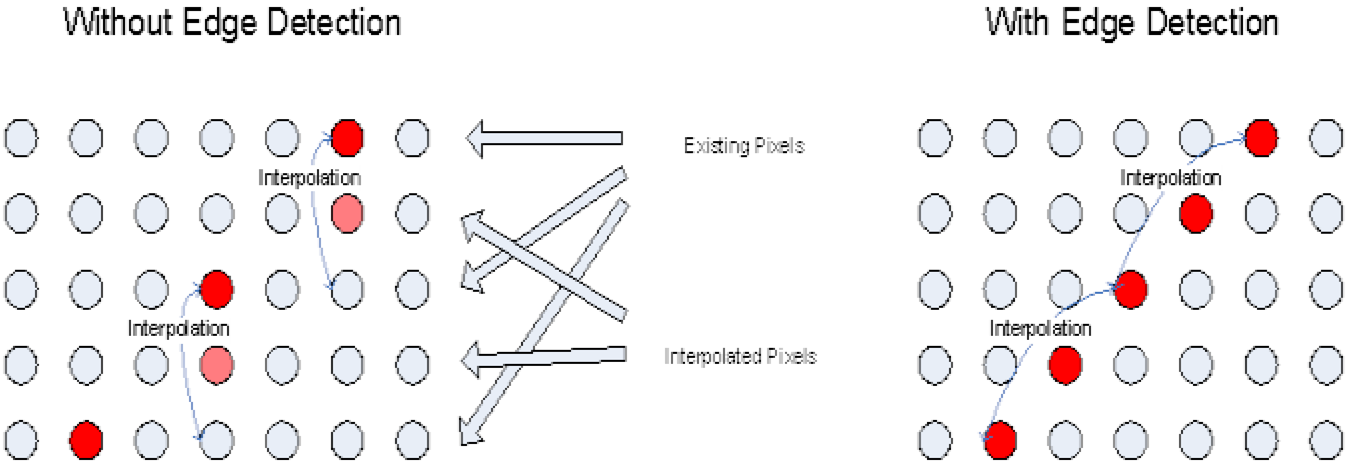
One final step is used to prevent sudden drops in STMM in the horizontal direction – taking the maximum of the STMM on the right and left sides.

The resulting STMM3 is used as a blending factor between the spatial and temporal deinterlacer.

## Spatial Deinterlacer Angle Detection

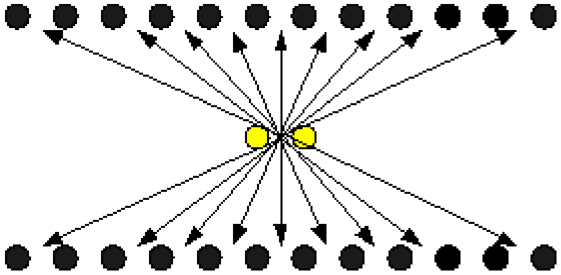
Deciding the best pixels to interpolate in the current field is the job of the spatial deinterlacer. The simplest method would be to interpolate directly from the pixels above and below the missing pixels, but this can look bad; edges and lines particularly look jagged with this solution.

A better solution is to detect the direction of edges in the pixel neighborhood and interpolate along the edge direction.



Edge detection is done by taking a window of pixels around the pixels of interest and comparing with a window offset in the direction being tested. The more similarity between the windows the more likely it is that the movement is in the direction of an edge.

We test 9 different directions to pick the best edge: vertical, +/-45°, +/-27°, +/-18° and +/-11 degrees.



## Spatial Deinterlacer Interpolation

Once the best angle is picked, both the chroma and luma need to be interpolated (see Chroma Up-Sampler for chroma). Only 422 output is needed, so there will be a chroma pair for each 2 lumas. The interpolation itself is very simple: take a pixel from the line above and the line below along one of the 9 possible angles, and average the 8-bit luma and chroma values to get the result pixel. We will do 2 lumas per clock to get enough performance.

## Chroma Up-Sampler

The DN/DI block supports 4:2:0 and 4:2:2 inputs, but only outputs 4:2:2. For 4:2:0 the chroma needs to be up-sampled to 4:2:2 before interpolation.

The 4:2:0 input has chroma at ¼ the rate of the luma; ½ in the horizontal and ½ in the vertical directions. The output needs to be 4:2:2, where chroma is ½ the rate of luma; ½ the horizontal but the same in the vertical direction. Then chroma can be de-interlaced in the vertical direction.

The 4:2:0 to 4:2:2 conversion requires doubling the chroma in the vertical direction to match the luma:

The chroma is doubled by a simple interpolation in both time and space.

Note that this simple chroma interpolation is not correct, since the chroma sample position is ¼ of a pixel different between 420 and 422. The polyphase filter in the scaler will be used to correct this inprecision by modifying the filter coefficients in software.

## Chroma Deinterlace

The next step is to do the deinterlacing. Chroma uses the output of the luma angle decision, but reduces the number of angles. The actual spatial deinterlace algorithm is a little different for chroma, since there are only 1 chroma per 2 lumas: some of the chromas are missing and must be filled in.

The chromas for +/-45° are derived by a simple average of the 90° and 27° chromas. +/-18° and +/-11° both use the chroma for +/-27°.

## Static Image Fallback Mode

This algorithm has a problem with static images – alternate fields use different luma angle detections and can select different angles, causing noticeable flicker. Rather than calculating a separate set of angles for chroma, we instead will blend with STMM so that a static image will use 90 degrees.

## Temporal Deinterlacer and Final Deinterlacer Blend

The temporal deinterlacer is a simple average between the previous and next field; when deinterlacing the 1st field of current the average is between the 2nd field of previous and the 2nd field of current.

The interpolation between spatial and temporal:

```
if ( STMM3 < stmm_min )
    deinterlace_out = tdi;
else if ( STMM3 > stmm_max )
    deinterlace_out = sdi;
else
    deinterlace_out = Blending (sdi, tdi)
```

## Progressive Cadence Reconstruction

When the FMD for the previous frame indicates that a progressive mode is being used rather than interlaced, the luma and chroma will be taken from adjacent fields rather than spatially interpolated.

Since we are deinterlacing 2 fields at a time – one from the previous frame and one from the current frame , we will need a state variable which says how each one should be put together. In each case there are only two possibilities – either the field should be put together with the matching field in the same frame or it should be put together with the adjacent field in the other frame.

Chroma is reconstructed the same as luma – only the first step of doubling chroma is done in the chroma upsampling block for the two needed fields.

## Motion Search

Motion will be estimated independently for each horizontal pair of pixels in the 16x4 block. The area around each pixel pair will be compared to areas in adjacent fields with small X/Y offsets.

The motion vector with the smallest SAD is kept as the best motion estimate; if two motion vectors have the same SAD then the last one tested will be kept.

## Robustness Checks

The motion estimate output goes through 2 checks to make sure it is not an aberration – a smoothness check and a consistency check.

### Consistency Check

The consistency check is done per pixel and makes sure that the pixels we are interpolating for MC have a lower delta than the ones that would be interpolated for spatial DI.

### Smoothness Check

The smoothness check compares the motion vector found for neighboring pixel pairs, to measure how similar they are to each other.

## Motion Comp

The MCDI output is on pixels chosen from adjacent field.

$$P_{DI}(x, y) = P_{ref\_same}(x + M_x / 2, y + M_y / 2);$$

## Merge with TDI & SDI

The MADI equation used in Gen6 was:

```
if (STMM3 < stmm_min)
deinterlace_out = tdi;
else if (STMM3 > stmm_max)
deinterlace_out = sdi;
Else
deinterlace_out = Blending (sdi, tdi)
```

Where STMM3 is a measure of the complexity of the scene and how much motion is in it.

The equation with MCDI is:

```
if (STMM3 < stmm_min)
        Deinterlace_out = tdi;
else if (STMM3 > stmm_max)
        deinterlace_out = DItemp;
else
deinterlace_out = lending (sdi, tdi)
```

Where DItemp is defined below:

```
If (Consistency check is passed && Smoothness check is passed)
        DItemp = MCDI;
Else
        DItemp = sdi;
```

## Film Mode Detector

The Film Mode Detector is generated in either the EU or in the driver with a set of differences gathered across entire fields. It is used to detect when a non-interlaced source like a film has been converted to interlaced video – in this case there will be pairs of fields which can be put back together to make frames rather than interpolating.

## VEBOX Output Statistics

The following statistics are covered in this section:

- Statistic offsets
- Capture Pipe Statistics
- Encoder Statistics

### Encoder Statistics Format

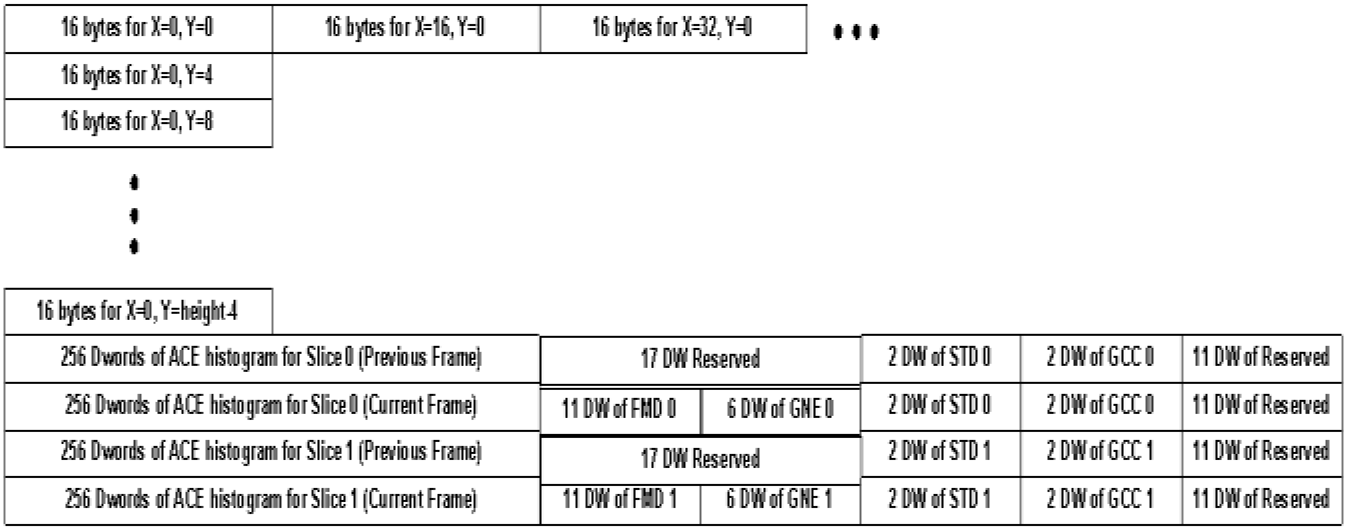
- Per Command Statistics
- Histograms

## Overall Surface Format

Statistics are gathered on both a per 16x4 block basis as well as on a per frame basis. There are 16 bytes of encoder statistics data per 16x4 block, plus a variety of per frame data which are stored in a linear surface. The 16 bytes of encoder statistics per block are output if either DN or DI are enabled and are organized into a surface with a pitch equal to the output surface width rounded up to 64 (so that each line starts and ends on a cache line boundary). The height of the surface is ¼ the height of the output surface. If both DN and DI are disabled then the encoder stats are not output and the per frame information is output at the base address.

The per frame information is written twice per frame to allow for a 2 slice solution – in a single slice the second set of data will be all 0. The final per frame information is found by adding each individual Dword, clamping the data (except for the ACE histogram, which is 24-bits in each 32-bit Dword) to prevent it from overflowing the Dword.

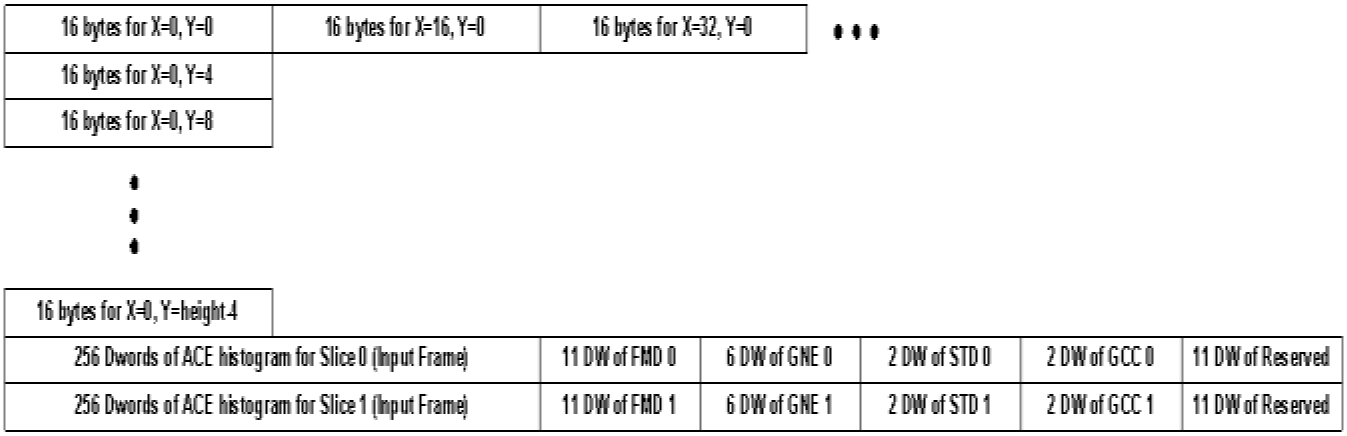
The Deinterlacer outputs two frames for each input frame. When IECP is applied to the Deinterlacer output, separate ACE, GCC and STD per frame statistics are created for each output frame. In this case, 4 copies of the per frame information are written – two copies for the two slice solution times two output frames. For the case of DN and no DI, only the first set of per frame statistics will be written.



| | | | |
|---|---|---|---|
| 16 bytes for X=0, Y=0 | 16 bytes for X=16, Y=0 | 16 bytes for X=32, Y=0 | • • • |
| 16 bytes for X=0, Y=4 | | | |
| 16 bytes for X=0, Y=8 | | | |
| • • • | | | |
| 16 bytes for X=0, Y=height-4 | | | |

| | | | | | |
|---|---|---|---|---|---|
| 256 Dwords of ACE histogram for Slice 0 (Previous Frame) | 17 DW Reserved | | 2 DW of STD 0 | 2 DW of GCC 0 | 11 DW of Reserved |
| 256 Dwords of ACE histogram for Slice 0 (Current Frame) | 11 DW of FMD 0 | 6 DW of GNE 0 | 2 DW of STD 0 | 2 DW of GCC 0 | 11 DW of Reserved |
| 256 Dwords of ACE histogram for Slice 1 (Previous Frame) | 17 DW Reserved | | 2 DW of STD 1 | 2 DW of GCC 1 | 11 DW of Reserved |
| 256 Dwords of ACE histogram for Slice 1 (Current Frame) | 11 DW of FMD 1 | 6 DW of GNE 1 | 2 DW of STD 1 | 2 DW of GCC 1 | 11 DW of Reserved |

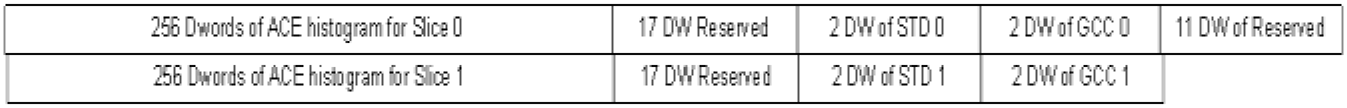
### Figure 7 – Statistics Surface when DI Enabled and DN either On or Off



| | | | |
|---|---|---|---|
| 16 bytes for X=0, Y=0 | 16 bytes for X=16, Y=0 | 16 bytes for X=32, Y=0 | • • • |
| 16 bytes for X=0, Y=4 | | | |
| 16 bytes for X=0, Y=8 | | | |
| • • • | | | |
| 16 bytes for X=0, Y=height-4 | | | |

| | | | | | |
|---|---|---|---|---|---|
| 256 Dwords of ACE histogram for Slice 0 (Input Frame) | 11 DW of FMD 0 | 6 DW of GNE 0 | 2 DW of STD 0 | 2 DW of GCC 0 | 11 DW of Reserved |
| 256 Dwords of ACE histogram for Slice 1 (Input Frame) | 11 DW of FMD 1 | 6 DW of GNE 1 | 2 DW of STD 1 | 2 DW of GCC 1 | 11 DW of Reserved |

### Figure 8 – Statistics Surface when DN Enabled and DI Disabled



| | | | | |
|---|---|---|---|---|
| 256 Dwords of ACE histogram for Slice 0 | 17 DW Reserved | 2 DW of STD 0 | 2 DW of GCC 0 | 11 DW of Reserved |
| 256 Dwords of ACE histogram for Slice 1 | 17 DW Reserved | 2 DW of STD 1 | 2 DW of GCC 1 | |

### Figure 9 – Statistics Surface when both DN and DI Disabled

When DN and DI are both disabled, only the per frame statistics are written to the output at the base address.

## Statistics Offsets

The statistics have different offsets from the base address depending on what is enabled.

The encoder statistics size is based on the frame size:

Encoder_size = width * (height+3)/4

Width is the width of the output surface rounded to the next higher 64 boundary. Height is the output surface height in pixels.

| | DI on | DI off + DN on | DI off + DN off |
|---|---|---|---|
| ACE_Histo_Previous_Slice0 | Encoder_size | N/A | N/A |
| Per_Command_Previous_Slice0 | Encoder_size + 0x400 | N/A | N/A |
| ACE _Histo_Current_Slice0 | Encoder_size + 0x480 | Encoder_size | 0x0 |
| Per_Command_Current_Slice0 | Encoder_size + 0x880 | Encoder_size + 0x400 | 0x400 |
| ACE_Histo_Previous1 | Encoder_size + 0x900 | N/A | N/A |
| Per_Command_Previous_Slice1 | Encoder_size + 0xD00 | N/A | N/A |
| ACE_Histo_Current1 | Encoder_size + 0xD80 | Encoder_size + 0x480 | 0x480 |
| Per_Command_Current_Slice1 | Encoder_size + 0x1180 | Encoder_size + 0x880 | 0x880 |

## Per Command Statistics Format

The Per Command Statistics are placed after the encoder statistics if either DN or DI is enabled. If the frame is split into multiple calls to the VEBOX, each call will output only the statistics gathered during that call and software will have to provide different base addresser per call and sum the resulting output to get the true per frame data.

The final address of each statistic is:

> **Statistics Output Address + Per_Command_Offset (pick the one for the slice desired and the current/previous frame for Deinterlacer) + PerStatOffset**

## FMD Variances/GNE Count

These are the FMD variances and Global Noise Estimate collected across the call. See vol5c Shared Functions, section 1.8.5 for a description of each one. Note that FMD values for blocks at the edge of the frame (within a 16x4 block that intersects or touches the frame edge) are not summed into the final value. FMD variances are 0 when the Deinterlacer is disabled, and GNE entries are 0 when the Denoise filter is disabled.

| Counter Id | PerStatOffset | Associated Counter |
|---|---|---|
| 0 | 0x00 | FMD Variance 0 |
| 1 | 0x04 | FMD Variance 1 |
| 2 | 0x08 | FMD Variance 2 |
| 3 | 0x0C | FMD Variance 3 |
| 4 | 0x10 | FMD Variance 4 |
| 5 | 0x14 | FMD Variance 5 |
| 6 | 0x18 | FMD Variance 6 |
| 7 | 0x1C | FMD Variance 7 |
| 8 | 0x20 | FMD Variance 8 |
| 9 | 0x24 | FMD Variance 9 |
| 10 | 0x28 | FMD Variance 10 |
| 11 | 0x2C | GNE Sum Luma (Sum of BNEs for all passing blocks) |
| 12 | 0x30 | GNE Sum Chroma U |
| 13 | 0x34 | GNE Sum Chroma V |
| 14 | 0x38 | GNE Count Luma (Count of number of block in GNE sum) |
| 15 | 0x3C | GNE Count Chroma U |
| 16 | 0x40 | GNE Count Chroma V |

## Simple Differences

The first set of variances are simply a sum of absolute pixel differences. The equations are done for every pixel with an even y coordinate:

**variance[0]** – difference between pixels from the top fields of the current and previous frame.

**variance[1]** – difference between pixels from the bottom fields of the current and previous frame.

**variance[2]** – difference between pixels from the top field and bottom field in the current frame.

**variance[3]** – difference between pixels from the top field of the current frame and bottom field of previous frame.

**variance[4]** – difference between pixels from the bottom field of the current frame and top field of previous frame.

The variances summed for each 16x4 block are divided by 16 before adding them to the sum for the frame to make sure the frame-level sum fits in a 32-bit register.

## Counter Variances

The rest of the variances are counters for variance conditions as described in the following string:

When sum of variance[0] and variance[1] is larger than moving pixel TH, Increase variance[5] when variance[2] is larger than sum of difference between two consecutive top field and difference two consecutive bottom field. Otherwise increase variance[6].

When sum of variance[0] and variance[1] is larger than moving pixel TH, if fields are vertically smooth, increase variance[6].

## Tear Variances

**variance[8]** = sum of TEAR1(x,y)

**variance[9]** = sum of TEAR_2(x,y)

**variance[10]** = sum of TEAR_3(x,y)

```
if (variance[8] > variance[9] && variance[8] > variance[10])
        variance[7] = variance[8] = variance[9] = variance[10] = 0
if (variance[8] < fmd_thr_tear)   variance[8] = 0
if (variance[9] < fmd_thr_tear)   variance[9] = 0
if (variance[10] < fmd_thr_tear) variance[10] = 0
```

## Skin Data Min/Max

If luma is smaller than the einter Ymin then Ymin will be replaced with Y. If Y is larger than Ymax then Ymax will be replaced by Y. These two registers are reset at the start of a command – Ymax is reset to zero and Ymin is reset to 0x3FF [HSW] or .

The 10 MSB of a 12-bit pixel or an 8-bit pixel with 2 zeros appended to the LSB are used for comparison [HSW].

There is also a simple count of all the skin data valids. Register values are 0 if the STD/STE function is disabled.

| PerStatOffset | Associated Register |
|---|---|
| 0x044 | Ymax (bits 25:16), Ymin (bits[9:0]), other bits zero [HSW] |
| 0x048 | Number of skin pixels (bits [28:0], other bits zero). |

## Gamut Compression Out of Range

The statistics gathered for Gamut Compression are a count of pixels out-of-range and a sum of the distances they are out of range. If the sum is greater than the maximum value of 0xFFFFFFFF then the value is clamped to the maximum. Both values are reset to zero at the start of each command. Both values are zero if the GCC function is disabled.

| PerStatOffset | Associated Register |
|---|---|
| 0x04C | Sum of distances of out-of-range pixels (clamps to 0xFFFFFFFF). |
| 0x050 | Number of pixels out of range (bits[28:0], other bits zero). |

## Histograms

The histograms are included in the main statistics surface along with the encoder statistics and the other per command statistics.

## Ace Histogram

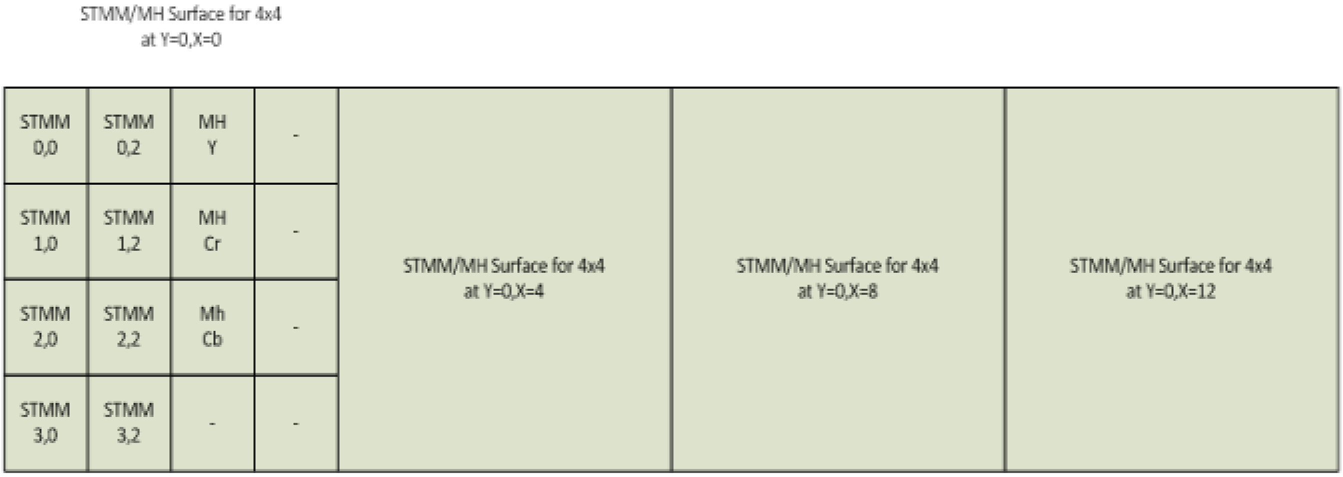
The Ace Histogram counts the number of pixels at different luma values. It has 256 bins, each of which is 24 bits. Any count that exceeds 24-bits is clamped to the maximum value. The data is stored on Dword boundaries with the upper 8-bits equal to zero.

| Y[9:2] | PerStatOffset | Associated Counter |
|---|---|---|
| 0 | 0x000 | ACE histogram, bin 0 |
| 1 | 0x004 | ACE histogram, bin 1 |
| 2 | 0x008 | ACE histogram, bin 2 |
| … | … | … |
| 255 | 0x3fc | ACE histogram, bin 255 |

## STMM/Denoise

The STMM/Denoise history is a custom surface used for both input and output. The previous frame information is read in for the DN (Denoise History) and DI (STMM) algorithms; while the current frame information is written for the next frame.

### STMM / Denoise Motion History Cache Line



STMM/MH Surface for 4x4 at Y=0,X=0

| | | | | | | |
|---|---|---|---|---|---|---|
| STMM 0,0 | STMM 0,2 | MH Y | - | STMM/MH Surface for 4x4 at Y=0,X=4 | STMM/MH Surface for 4x4 at Y=0,X=8 | STMM/MH Surface for 4x4 at Y=0,X=12 |
| STMM 1,0 | STMM 1,2 | MH Cr | - | | | |
| STMM 2,0 | STMM 2,2 | Mh Cb | - | | | |
| STMM 3,0 | STMM 3,2 | - | - | | | |

| Byte | Data |
|---|---|
| 0 | STMM for 2 luma values at luma Y=0, X=0 to 1 |
| 1 | STMM for 2 luma values at luma Y=0, X=2 to 3 |
| 2 | Luma Denoise History for 4x4 at 0,0 |
| 3 | Not Used |
| 4-5 | STMM for luma from X=4 to 7 |
| 6 | Luma Denoise History for 4x4 at 0,4 |
| 7 | Not Used |
| 8-15 | Repeat for 4x4s at 0,8 and 0,12 |
| 16 | STMM for 2 luma values at luma Y=1,X=0 to 1 |
| 17 | STMM for 2 luma values at luma Y=1, X=2 to 3 |
| 18 | U Chroma Denoise History |
| 19 | Not Used |
| 20-31 | Repeat for 3 4x4s at 1,4, 1,8 and 1,12 |
| 32 | STMM for 2 luma values at luma Y=2,X=0 to 1 |
| 33 | STMM for 2 luma values at luma Y=2, X=2 to 3 |
| 34 | V Chroma Denoise History |
| 35 | Not Used |
| 36-47 | Repeat for 3 4x4s at 2,4, 2,8 and 2,12 |
| 48 | STMM for 2 luma values at luma Y=3,X=0 to 1 |
| 49 | STMM for 2 luma values at luma Y=3, X=2 to 3 |
| 50-51 | Not Used |
| 36-47 | Repeat for 3 4x4s at 3,4, 3,8 and 3,12 |

## VEBOX State & Primitive Commands

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

There are two kinds of state information:

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

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

Each frame should have these commands, in this order:

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

### VEBOX_SURFACE_STATE

## Surface Format Restrictions

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

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

- All 16-bit formats are processed at 12-bit internally.
- Surfaces are 4 kb aligned, chroma X offset is cache line aligned (16 byte).
- If Y8/Y16 is used as the input format, it must also be used for the output format (chroma is not created by VEBOX).
- If IECP and either DN or DI are enabled at the same time, it is possible to select any input that is legal for DN/DI and any output which is legal for IECP. The only exception is that if DN or DI are enabled, the IECP is not able to output P216 and P016.

# State Commands

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

- DN/DI State Table Contents
- VEBOX_IECP_STATE

**VEBOX_STATE**

**VEBOX_Ch_Dir_Filter_Coefficient**

## DN-DI State Table Contents

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

### VEBOX_DNDI_STATE

## VEBOX_IECP_STATE

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

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

**VEBOX_IECP_STATE**

**VEBOX_STD_STE_STATE**

**VEBOX_ACE_LACE_STATE**

**VEBOX_TCC_STATE**

**VEBOX_PROCAMP_STATE**

**VEBOX_CSC_STATE**

**VEBOX_ALPHA_AOI_STATE**

For all piecewise linear functions in the following table, the control points must be monotonically increasing (increasing continuously) from the lowest control point to the highest. Any control points which have the same value must also have the same bias value. The piecewise linear functions include:

- PWL_Gamma_Point11 to PWL_Gamma_Point1
- PWL_INV_Gamma_Point11 to PWL_Gamma_Point1

### VEBOX_GAMUT_STATE

### VEBOX_VERTEX_TABLE

### VEBOX_RGB_TO_GAMMA_CORRECTION

# VEB DI IECP Commands

The VEB_DI_IECP command causes the VEBOX to start processing the frames specified by VEB_SURFACE_STATE using the parameters specified by VEB_DI_STATE and VEB_IECP_STATE.

- VEB_DI_IECP Command
- VEB_DI_IECP Command

### VEB_DI_IECP

The Surface Control bits for each surface:

### VEB_DI_IECP Command Surface Control Bits

## Command Stream Backend - Video

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this 2nd command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project; the offsets will be maintained.

- VECS_ECOSKPD — VECS ECO Scratch Pad

# Registers for Video Codec

## Introduction

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this 2nd command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project, the offsets will be maintained.

| Base Address Value for the memory interface register offset for the Bit Stream Command Stream | Project |
|---|---|
| 0x10000<br>eg: The Ring buffer tail pointer will be 0x10000 + 0x2030 | |

### VECS_ECOSKPD - VECS ECO Scratch Pad

# Video Enhancement Engine Functions

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this 2nd command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project; the offsets will be maintained.

The section contains the following registers:

- Virtual Memory Control
- VECS_RINGBUF — Ring Buffer Registers

## Registers for Video Codec

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this 2nd command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project, the offsets will be maintained.

| Project | Base Address Value for the memory interface register offset for the VEBOX Command Stream |
|---|---|

## Virtual Memory Control

VEBOX supports a 2-level mapping scheme for PPGTT, consisting of a first-level page directory containing page table base addresses, and the page tables themselves on the 2nd level, consisting of page addresses.

| | |
|---|---|
| Project | VECS_PP_DCLV - VECS PPGTT Directory Cacheline Valid Register |
| | VECS_EXCC - VECS Execute Condition Code Register |

## VECS_RINGBUF — Ring Buffer Registers

The following are Ring Buffer Registers:

**RING_BUFFER_TAIL - Ring Buffer Tail**

**RING_BUFFER_HEAD - Ring Buffer Head**

**RING_BUFFER_START - Ring Buffer Start**

**RING_BUFFER_CTL - Ring Buffer Control**

**UHPTR - Pending Head Pointer Register**

**UHPTR - Pending Head Pointer Register**