Intel® Open Source HD Graphics and Intel Iris™ Graphics

Programmer's Reference Manual

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

Volume 9: Media VEBOX

May 2015, Revision 1.0

Creative Commons License

Notices and Disclaimers

Table of Contents

Media VEBOX Introduction

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

The following sections are contained in Media VEBOX:

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

The IECP consists of these functions:

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

Denoise

This section discusses the Denoise feature in the chipset.

- Denoise Filter - detects noise in the input image and filters the image with either a temporal filter or a spatial filter. The temporal filter is applied when low motion is detected.
- Chroma Denoise Filter - detects noise in the U and V planes separately and applies a temporal filter.
- Block Noise Estimate (BNE) - as part of the Global Noise Estimate (GNE) algorithm, BNE estimates the noise over each block of pixels in the input picture.
- Global Noise Estimate (GNE) - GNE is 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 input frame. Noise estimates are kept between frames and blended together.

Filters and Functions:

- Denoise filter
- Temporal filter
- Context Adaptive Spatial Filter
- Denoise Blend
- Chroma Noise Reduction
- Block Noise Estimate

Motion Detection and Noise History Update

This logic detects motion of the current block for the denoise filter, which it then combines with motion detected in the co-located block of the past frame to be stored in the denoise history table. Denoise history is saved to memory and also used to control the temporal denoise filter.

Temporal Filter

Temporal denoise is applied to each pixel based on the noise strength measured from the input pictures. Each pair of co-located pixels in the current and previous input pictures is blended together to generated the output pixel.

Context Adaptive Spatial Filter

For each pixel in the local 3x3 neighborhood, its luma value is compared (via absolution difference) to the center pixel to be filtered.

Each pixel in the neighborhood for which the absolute difference is less than *good_neighbor_th* is marked as a "good neighbor".

The filtered output pixel is then equal to average of good neighbor pixels.

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 the luma noise detection module, BNE and GNE.

The U & V channels are processed individually to generate a noise estimate for each of the 2 channels.

Chroma Noise Reduction Filter

A simple and effective temporal-domain chroma noise reduction filter is introduced.

The Noise History is updated based on the motion detection result and is saved to the memory.

The Noise History value is used to control the temporal denoise filter.

Temporal Filter

The output of the temporal filter is computed as a weighted blending of two co-located chroma pixels in the current and previous input pictures.

The Noise History computed between the current and previous input pictures is used to control the strength of this temporal blending of two co-located chroma pixels.

The output of the temporal filter is blended again with the input pixel value in the current picture depending on the motion information.

Block Noise Estimate (Part of Global Noise Estimate)

The BNE estimates the amount of noise in each rectangular region of the input picture. The BNE estimate is computed separately for each color component in the input picture. The estimates from BNE are summed together to generate the Global Noise Estimate (GNE) for the entire input picture.

Deinterlacer

The deinterlacer (DI) takes the top and bottom fields of each input frame and converts them into two individual output frames. This block also gathers statistics for a film mode detector (FMD) that runs in software at the end of the frame. If the film mode detector determines that the input is progressive rather than interlaced, then the input fields are put together to construct the progressive output frame.

Features:

- **Deinterlacer** - estimates how much motion is present across the input fields. Low motion scenes are reconstructed by averaging pixels from temporally nearby fields (temporal deinterlacer), while high motion scenes are reconstructed by interpolating pixels from spatially nearby fields (spatial deinterlacer).
- **Film Mode Detection (FMD)** - determines if the input fields are created by sampling film content and converting it to interlaced video. If so, the deinterlacer is turned off in favor of reconstructing the progressive output frame directly from adjacent fields. Various sum-of-absolute differences are computed per block. The FMD algorithm consumes these variances from all blocks of both input fields at the end of the frame.
- **Progressive Cadence Reconstruction** - If the FMD for the previous input frame determines that film content has been 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 data is doubled vertically to convert it to 4:2:2. Chroma data is then processed by its own version of the deinterlacer or progressive cadence reconstruction algorithms.

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 in the current field picture can be provided by looking at the previous or next field pictures (temporal deinterlacing or TDI). If there is motion in the scene, then objects in the previous and next fields are displaced from the location in the current field, so motion estimation and compensation are required to deinterlace using the temporally neighboring field pictures. Instead, spatial interpolation from the neighboring lines above and below in the current field picture is used to fill in the missing lines (spatial deinterlacing or SDI).

The motion adaptive deinterlacing is implemented by computing 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 current pixel, then the missing pixels/lines are filled in by averaging the pixel values from the previous and next fields. If the STMM shows that there is motion, then the missing pixels/lines are filled in by interpolating from spatially neighboring lines. The two results from TDI and SDI are alpha-blended for intermediate values of STMM to prevent sudden transitions between TDI and SDI modes.

The deinterlacer uses two frames for reference. The current frame contains the field that is being deinterlaced. The reference frame is the closest frame in time to the field that is being deinterlaced. for example, when the 1st field is being deinterlaced, the previous frame is the reference and when the 2nd field is being deinterlaced, it is the next frame that is the reference.

Film Mode Detector

The File Mode Detector (FMD) detects film contents that have been converted to interlaced video in which case each pair of input fields are merged together to a frame picture.

Image Enhancement Color Processing (IECP)

Skin Tone Detection Enhancement (STDE)

The STD/E unit, composed of the Skin Tone Detection (STD) and Skin Tone Enhancement (STE) units, is part of color processing pipe located at the Render Cache Pixel Backend (RCBP).

The main goal of the STD/E is to reproduce the skin colors in a way that is more palatable to the observer, and by that to increase the sensed image quality. It may also pass indication of skin tones to the TCC and ACE.

The STD unit detects the skin-like colors and passes a likelihood score for each input pixel indicating the probability that it is a skin tone color to the STE. The STE modifies the saturation and hue of the pixel according to its skin tone likelihood score. Both the STD and STE evaluations are done on a per-pixel basis. The input pixels are required to be in the YUV space.

The skin tone detection score (that is, skin tone likelihood score) is recorded as a 5-bit number and it is passed to ACE and TCC blocks to indicate the strength of skin tone likelihood.

STD Score Output

When the state bit "Output STD Decisions" is set, STD scores fill the output instead of the pixel values. To output STD scores, STD should be enabled and all other functions in the IECP after STDE, except ACE, should be disabled - only ACE can be enabled to collect the histogram of the STD score values.

The output YUV data when "Output STD Decision" is enabled should be as follows:

- Y = 0x7FF + (STD_Score «6)
- U = 0x7FF
- V = 0x7FF

In this mode, a histogram of skin tone distribution can be obtained in ACE, and a special ACE PWLF curve (step function) can be configured to produce a binary picture to illustrate the pixels based on the level of skin tone detection.

Adaptive Contrast Enhancement (ACE)

Automatic Contrast Enhancement (ACE) is a part of the color processing pipe which is located at the render cache in the RCPB block.

The main goals of ACE are to improve the overall contrast of the image and to emphasize details in obscured regions, such as dark regions of the input image.

The ACE algorithm analyzes the input image and modifies contrast of the image according to its content characteristic. Analysis and contrast adjustment are performed over the Y component.

Total Color Control (TCC)

TCC adjusts the color saturation level of the input image based on six anchor colors (Red, Green, Blue, Magenta, Yellow, and Cyan). The TCC algorithm operates on the UV-color components in the YUV color space on a per-pixel basis.

Input and output pixels are in the YUV444 12bpc (HSW/BDW) format. The input to the TCC block is:

- U and V color components (10 bit)
- Skin-tone detection value (5 bit)
- External control parameters

The output of the TCC block is:

- Updated U and V values (10 bit)

The TCC block is implemented in HW to reduce the power of the system and improve the battery life.

The TCC block is controlled by state only and does not require any memory access. The TCC block runs at the same frequency as the existing RCPBunit.

The TCC block includes three sub-blocks: Angle_Calculator, Saturation_Factor_Calculator, UV_Modification.

Computation of the saturation factor SFs2 involves (UVMaxColor, Inv_UVMaxColor) where UVMaxColor is the maximum (and legal) absolute UV values, which in the case of YUV color space equals 448 in 10-bit representation. Inv_UVMaxColor is the inverse calculation of UVMaxColor, that is, 1/UVMaxColor.

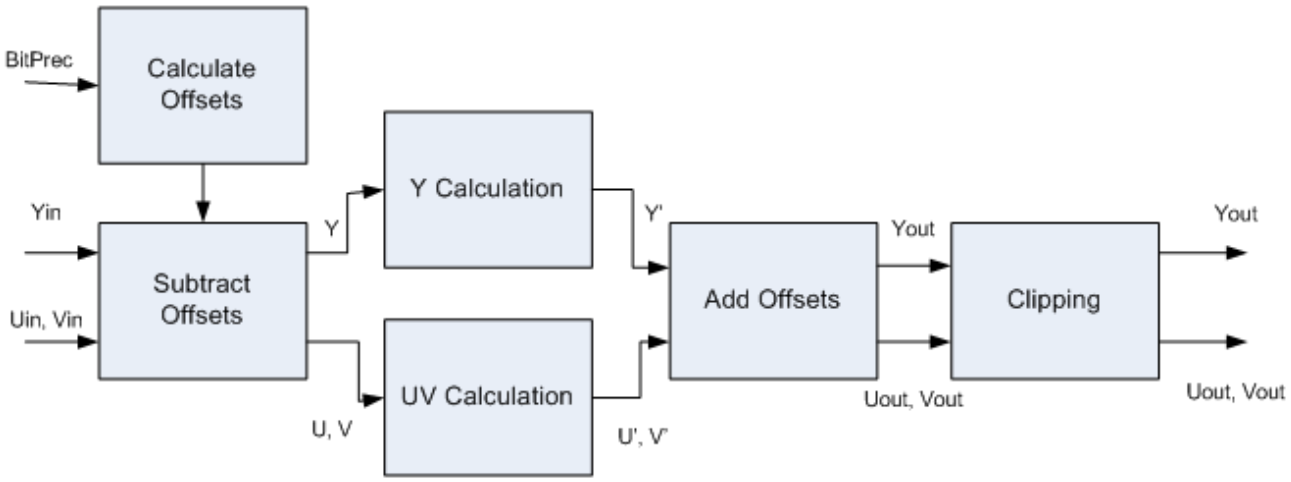
The third saturation factor SFs3 involves CLF which is Color Limiting Factor and ranges from 0 to 1. CLF is computed using a threshold value UV_Threshold.

The last and fourth saturation factor SFs4 considers the skin-tone pixels and a threshold value STE_Threshold.

ProcAmp

The PROCAMP block modifies the brightness, contrast, hue and saturation of the input image in YUV color space.

Input and output pixels are in the YCbCr 444 12bpc (bits per channel) format. Precision=12.



Y Processing:

An offset of 256 (that is, 16 in 8bpc) is subtracted from the 12-bit Y values to position the black level at zero. This removes the DC offset so that adjusting the contrast does not vary the black level. Since Y values may be less than 256, negative Y values should be supported at this point. Contrast is adjusted by multiplying the YUV pixel values by a constant. If U and V are adjusted, a color shift will result whenever the contrast is changed. The brightness property value is added (or subtracted) from the contrast adjusted Y values; this is done to avoid introducing a DC offset due to adjusting the contrast. Finally the offset 256 is added back to reposition the black level at 256.

The equation for processing of Y values is:

$$Yout' = ((Yin-256) \times C) + B + 256,$$

where C is the Contrast adjustment value and B is the Brightness adjustment value.

UV Processing:

An offset of 2048 (that is, 128 in 8bpc) is subtracted from the 12-bit U and V values. The hue adjustment is implemented by combining the U and V input values together as in:

```
Uout' = (Uin-2048) x Cos(H) + (Vin-2048) x Sin(H)
Vout' = (Vin-2048) x Cos(H) - (Uin-2048) x Sin(H)
```

where H represents the desired Hue angle; Saturation is adjusted by multiplying the U and V input values by a constant S.

Finally, the offset value 2048 is added back to both U and V.

The combined processing of Hue, Saturation and Contrast on the UV data is:

```
Uout' = (((Uin-2048) x Cos(H) + (Vin-2048) x Sin(H)) x C x S) + 2048
Vout' = (((Vin-2048) x Cos(H) - (Uin-2048) x Sin(H)) x C x S) + 2048
```

where C is the contrast, H is Hue angle and S is the Saturation.

The multiplication factors Cos(H)x*Cx*S and Sin(H)x*Cx*S are programmed by the parameters Cos_c_s and Sin_c_s.

Color Space Conversion

The CSC block enables linear conversion between different color spaces such as YCbCr and RGB using vector shifts and matrix multiplication.

The CSC algorithm is a linear coordinate transformation, comprising of the following steps:

1. Shift the input color coordinate
2. Multiply by 3x3 matrix
3. Shift the output color coordinate

The formula representation of the 3 steps is:

$$\begin{pmatrix} vout\_1 \\ vout\_2 \\ vout\_3 \end{pmatrix} = \begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} * \begin{pmatrix} vin\_1+v0\_1 \\ vin\_2+v0\_2 \\ vin\_3+v0\_3 \end{pmatrix} + \begin{pmatrix} u0\_1 \\ u0\_2 \\ u0\_3 \end{pmatrix}$$

The output pixel values are clipped to ensure that each color component is within the valid range.

Color Gamut Compression

With the rapid development of capture and display devices, images can be captured, manipulated, and reproduced in a variety of forms. Due to the fact that different devices support different color gamuts, gamut mapping becomes an important feature in media processing where video/image data on multiple platforms are often shared and exchanged. The problems of gamut mapping can be divided into two categories: (1) mapping from wider gamut to narrow gamut, and (2) mapping from narrow gamut to wider gamut. The first category is defined as Color Gamut Compression while the second one is defined as Color Gamut Expansion.

Color Gamut Compression module provides the functionality of mapping the color content in a color gamut wider than that of the output display to the color gamut of the output display while maintaining the hue of the input content. Color Gamut Compression module, for example, maps xvYCC color space to sRGB color space.

The simplest gamut compression method is to clip the out-of-range color values to the valid range (i.e., 0-1 in normalized, linear space). Although this simple clipping method leads to acceptable visual appearance in some cases, the loss of color depth can be problematic as multiple out-of-range color values are mapped to the same color at the gamut boundary. This simple clipping method treats each color channel (i.e., R/G/B) independently and this may also lead to unexpected color distortion since the composite ratio of three primaries (i.e., color hue) is changed.

An advanced approach takes these two factors into account, maintains the original color content information of the image after gamut compression, and is capable of producing output pictures that are more visually pleasant than those produced by the simple clipping.

Usage Models

There are two usage models of the gamut compression module:

- Basic mode: fixed-hue color gamut clipping mode
- Advanced mode: fixed-hue full range mapping mode

The application of basic mode (that is, fixed-hue color gamut clipping) is preferred when the content has a smaller percentage of out-of-range pixels in the scene. The advanced mode (that is, fixed-hue full range mapping) may change the color of the in-range pixels in addition to the color of the out-of-range pixels and is thus preferred when the percentage of out-of-range pixels is high. The percentage of the out-of-range pixels is derived from the out-of range color gamut detection module to provide an indicator to select between basic mode and advanced mode.

Gamut Compression Module Overview

The main goal of color gamut compression algorithm is to compress out-of-range pixel values while keeping their hue values same as before.

At the IECP pipeline level, the input to the gamut compression unit comes from the STDE unit and the output of the gamut compression goes to the TCC unit.

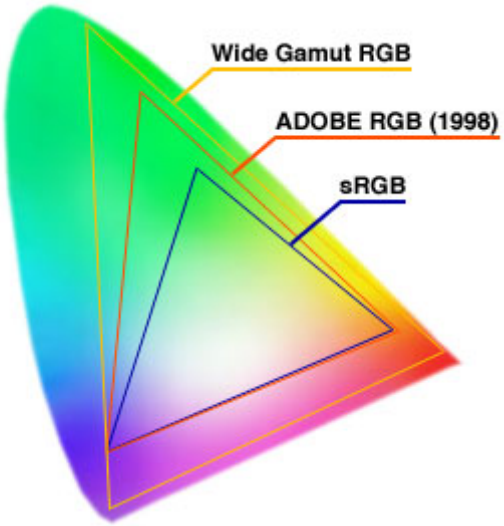
The gamut compression comprises of the following stages:

- xvYCC decoding
- YUV2LCH color space conversion
- Fixed-hue Gamut Compression
- xvYCC encoding

Color Correction (Gamut Expansion)

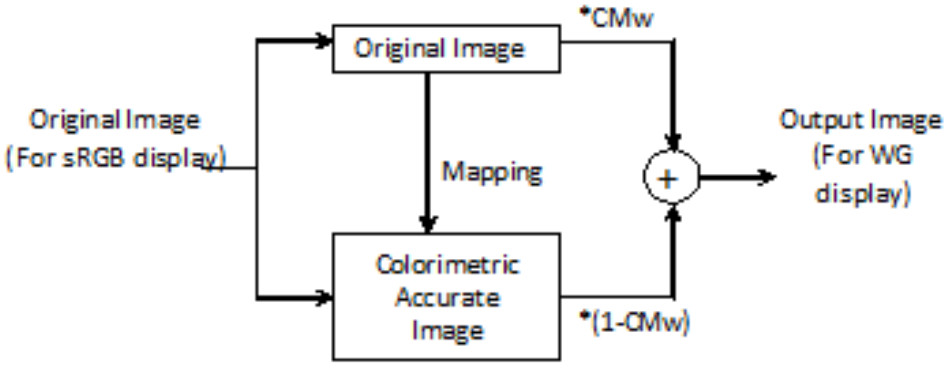
Color Correction is an important and commonly used feature where input RGB colors are modified to output RGB colors in linear RGB space. Color Correction shares the same HW with Gamut Expansion and all descriptions of the Gamut Expansion process in this section apply equally to the Color Correction usage of the HW.

An increasing number of wide gamut (WG) displays are available, which provide additional colors over the traditional display. While most photography today complies with the sRGB standard color space, which covers around 72% of the color perceived by humans, this sRGB color content looks incorrect/unnatural on wide gamut displays. Therefore, a gamut mapping (GM) algorithm is required to adjust the input gamut range to fit the output gamut range.



Overview

The main goal of the gamut expansion algorithm is to produce an output image as the composite of the original image and the accurate-color-reproduction image as shown below:



Gamut Expansion: The image output for a WG display is composed of the original image and the colorimetric accurate image.

Usage Models

There are two usage models depending on the set up of the *Global Mode Enable* bit:

- Basic mode: global color gamut expansion mode
- Advanced mode: pixel adaptive color gamut expansion mode

The basic mode (global color gamut expansion) provides uniform blending among the colorimetric accurate color and the original color stretched from the color primaries of the wider color gamut display.

The advanced mode (pixel adaptive color gamut expansion) provides per-pixel adaptive weighting to take advantage of the property of the extended color gamut of the current display panel. The pixel adaptive color gamut expansion mode is based on the characteristics of the currently available wide gamut panels. The global color gamut expansion mode may fit in the usage model if the property of the future wide gamut display panels allows it. This is subject to the application configuration upon the product delivery.

Overall Surface Format

Statistics are gathered on both per-block (16x4) basis and 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 to the next higher 64 boundary (so that each line starts and ends on a cache line boundary). The height of the surface is ¼ of 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. For the case of DN and no DI, only the first set of per frame statistics will be written.

Statistics Surface when DI Enabled and DN either On or Off

<table>
<tr><td>16 bytes for X=0, Y=0</td><td>16 bytes for X=16, Y=0</td><td>16 bytes for X=32, Y=0</td><td>• • •</td></tr>
<tr><td>16 bytes for X=0, Y=4</td></tr>
<tr><td>16 bytes for X=0, Y=8</td></tr>
<tr><td>•
•
•</td></tr>
<tr><td>16 bytes for X=0, Y=height-4</td></tr>
</table>

<table>
<tr><td>256 Dwords of ACE histogram for Slice 0 (Previous Frame)</td><td colspan="2">17 DW Reserved</td><td>2 DW of STD 0</td><td>2 DW of GCC 0</td><td>11 DW of Reserved</td></tr>
<tr><td>256 Dwords of ACE histogram for Slice 0 (Current Frame)</td><td>11 DW of FMD 0</td><td>6 DW of GNE 0</td><td>2 DW of STD 0</td><td>2 DW of GCC 0</td><td>11 DW of Reserved</td></tr>
<tr><td>256 Dwords of ACE histogram for Slice 1 (Previous Frame)</td><td colspan="2">17 DW Reserved</td><td>2 DW of STD 1</td><td>2 DW of GCC 1</td><td>11 DW of Reserved</td></tr>
<tr><td>256 Dwords of ACE histogram for Slice 1 (Current Frame)</td><td>11 DW of FMD 1</td><td>6 DW of GNE 1</td><td>2 DW of STD 1</td><td>2 DW of GCC 1</td><td>11 DW of Reserved</td></tr>
</table>

Statistics Surface when DN Enabled and DI Disabled

<table>
<tr><td>16 bytes for X=0, Y=0</td><td>16 bytes for X=16, Y=0</td><td>16 bytes for X=32, Y=0</td><td>• • •</td></tr>
<tr><td>16 bytes for X=0, Y=4</td></tr>
<tr><td>16 bytes for X=0, Y=8</td></tr>
<tr><td>•
•
•</td></tr>
<tr><td>16 bytes for X=0, Y=height-4</td></tr>
</table>

<table>
<tr><td>256 Dwords of ACE histogram for Slice 0 (Input Frame)</td><td>11 DW of FMD 0</td><td>6 DW of GNE 0</td><td>2 DW of STD 0</td><td>2 DW of GCC 0</td><td>11 DW of Reserved</td></tr>
<tr><td>256 Dwords of ACE histogram for Slice 1 (Input Frame)</td><td>11 DW of FMD 1</td><td>6 DW of GNE 1</td><td>2 DW of STD 1</td><td>2 DW of GCC 1</td><td>11 DW of Reserved</td></tr>
</table>

Statistics Surface when both DN 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	

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
```

where

width is the width of the output surface rounded to the next higher 64 boundary and

height is the output surface height in pixels.

Offset	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

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 outputs only the statistics gathered during that call and software provides different base address per call and sums the resulting output to compute the 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 and GNE Statistics

These are the 11 FMD variances (Variance 0 ~ 10) and Global Noise Estimate Statistics (Sums and Counts) collected in each VEBOX call.

Note that pixel values in blocks that are close to the edge of the frame (within a 16x4 block that intersects or touches the frame edge) are not used in the variance computation.

FMD variances are 0 when the Deinterlacer is disabled.

GNE estimates are 0 when the Denoise 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

Skin-Tone Data

The register Ymax stores the largest luma value. It is reset at the start of a command to zero.

The register Ymin stores the smallest luma value. It is reset at the start of a command to:

Reset Value
0x3FF (1023 in 10 bits)

There is also a 29-bit counter of all the skin pixels (Number of Skin Pixles).

Register values are 0 if the STD/STE function is disabled.

The registers are stored with the offsets as shown below.

PerStatOffset	Associated Register
0x044	Ymax (bits 25:16), Ymin (bits[9:0]), other bits zero
0x048	Number of Skin Pixels (bits [28:0], other bits zero)

Gamut Compression: Out of Range Pixels

The statistics gathered for Gamut Compression are:

1. Count of out-of-range pixels (29 bits) and
2. Sum of the distances from out-of-range pixels to the closest range boundaries (32 bits).

If the sum is greater than the maximum 32-bit value, then it is clamped to the maximum 0xFFFFFFFF.

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 (clamped to 0xFFFFFFFF)
0x050	Number of out-of-range pixels (bits [28:0], other bits are zero)

Histograms

The histograms are included in the main statistics surface along with the encoder statistics and 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 out 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

Byte	Data
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 and Primitive Commands

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

There are two kinds of state information:

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

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

Each frame should have these commands, in this order:

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

VEBOX State

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

VEBOX_CCM_STATE

Black Level Correction State - DW75..76

VEBOX_FORWARD_GAMMA_CORRECTION_STATE

VEBOX_FRONT_END_CSC_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_CAPTURE_PIPE_STATE

VEBOX_RGB_TO_GAMMA_CORRECTION

VEBOX Surface State

VEBOX_SURFACE_STATE

Surface Format Restrictions

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

Surface Format Restrictions [BDW]

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

Surface Formats - Feature Notes

Feature
Surfaces are 4 kb aligned, chroma X offset is cache line aligned (16 byte).
If Y8/Y16 is used as the input format, it must also be used for the output format (chroma is not created by VEBOX).
All 16-bit formats are processed at 12-bit internally.

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.

Feature
VEB_DI_IECP Command

The Surface Control bits for each surface:

VEB_DI_IECP

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 second 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

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.