Image Processing Kernels in C6Accel

^ Up to main **C6Accel_Reference_guide** Table of Contents

This arcticle is part of a collection of articles describing the C6Accel included in DaVinci/OMAPL/OMAP3 devices. To navigate to the main page for the C6Accel reference guide click on the link above.

Image Processing based kernel API call Reference guide

int C6accel_IMG_histogram_8(8-bit Histogram)

Function

```
  int C6accel_IMG_histogram_8(C6accel_Handle hC6accel,const unsigned
char * restrict in_data, int n, short accumulate,unsigned short *
restrict t_hist, unsigned short * restrict hist)
```

Parameters

- hC6accel: C6Accel handle
- in_data[n] Input image. Must be word aligned.
- n Number of pixels in input image. Must be multiple of 8.
- accumulate
  - 1: Add to existing histogram in hist[ ]
  - -1: Subtract from existing histogram in hist[ ]
- t_hist[1024] Array of temporary histogram bins. Must be initialized to zero.
- hist[256] Array of updated histogram bins.
- Return value: Error codes

Description This routine computes the histogram of the array in_data[ ] which contains n 8-bit elements. It returns a histogram in the array hist[ ] with 256 bins at 16-bit precision. It can either add or subtract to an existing histogram, using the accumulate control. It requires temporary storage for four temporary histograms, t_hist[ ], which are later summed together.

Special Requirements

- The temporary array of data, t_hist[ ], must be initialized to zero.
- The input array of data, in_data[ ], must be word-aligned.
- n must be a multiple of 8.
- The maximum number of pixels that can be profiled in each bin is 65535 in the main histogram.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_histogram_16(16 bit histogram)

Function

```
  int C6accel_IMG_histogram_16( C6accel_Handle hC6accel,unsigned short
*restrict in, short *restrict hist, short *restrict t_hist, int n, int
accumulate, int img_bits)
```

Parameters

- hC6accel: C6Accel handle
- in Input image of size n
- hist Array of updated histogram bins

- t_hist Array of temporary histogram bins
- n Nunber of pixels in input image
- accumulate
  - 1: add to existing histogram in hist[ ]
  - -1: subtract from existing histogram in hist[ ]
- img_bits Number of valid data bits in a pixel
- Return value: Error codes

Description This code takes a histogram of an array (of type short) with n number of pixels, with img_bits being the number of valid data bits in a pixel. It returns the histogram of corresponding number of bins at img_bits bits precision. It can either add or subtract to an existing histogram, using the accumulate control. It requires some temporary storage for four temporary histograms, which are later summed together. The length of the hist and the t_hist arrays depends on the value of img_bits. The length of the hist array is 2(img_bits) and that of t_hist is 4 * 2(img_bits) as there are no pixel values greater than or equal to 2(img_bits) in the given image.

Special Requirements

- n must be a multiple of 8 and greater than or equal to 8.
- The elements of arrays of data, t_hist are initialized to zero.
- in and t_hist arrays must be double-word aligned.
- hist array must be word-aligned
- Input and output arrays do not overlap
- img_bits must be at least 1
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_median_3x3_8(8 bit 3x3 Median filter)

Function

```
  int C6accel_IMG_median_3x3_8(C6accel_Handle hC6accel,unsigned char
*in_data, int cols, unsigned char *out_data)
```

Parameters

- hC6accel: C6Accel handle
- in_data Pointer to input image data. No alignment is required.
- cols Number of columns in input (or output). Must be multiple of 4.
- out_data Pointer to output image data. No alignment is required.
- Return value: Error codes

Description This routine performs a 3´3 median filtering algorithm. The gray level at each pixel is replaced by the median of the nine neighborhood values. The function processes three lines of input data pointed to by in_data, where each line is cols' pixels wide, and writes one line of output data to out_data. For the first output pixel, two columns of input data outside the input image are assumed to be all 127. The median of a set of nine numbers is the middle element, so that half of the elements in the list are larger and half are smaller. A median filter removes the effect of extreme values from data. It is a commonly used operation for reducing impulsive noise in images.

Special Requirements

- cols must be a multiple of 4.
- No alignment is required.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_sobel_3x3_8(8 bit 3x3 Sobel filter)

Function

```
  int C6ACCEL_IMG_sobel_3x3_8(C6accel_Handle hC6accel,const unsigned
char *in_data, unsigned char *out_data, short cols, short rows)
```

Parameters

- hC6accel: C6Accel handle
- in_data[ ] Input image of size cols * rows.
- out_data[ ] Output image of size cols * (rows-2).
- cols Number of columns in the input image. Must be multiple of 2.
- rows Number of rows in the input image. cols * (rows-2) must be multiple of 8.
- Return value: Error codes

Description This routine applies horizontal and vertical Sobel edge detection masks to the input image and produces an output image which is two rows shorter than the input image.Within each row of the output, the first and the last pixel will not contain meaningful results.

Special Requirements

- cols must be a multiple of 2.
- At least eight output pixels must be processed; i.e., cols * (rows-2) must be a multiple of 8.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_sobel_3x3_16(16 bit Sobel Filter)

Function

```
  int  C6accel_IMG_sobel_3x3_16(C6accel_Handle hC6accel,constant
unsigned short *restrict 3x3 unsigned input in, unsigned short
*restrict out, short cols, short rows)
```

Parameters

- hC6accel: C6Accel handle
- in[ ] Image input of size rows x cols
- out[ ] Image output of size (rows - 2) x cols
- cols Number of columns in the input image
- rows Number of rows in the input image
- Return value: Error codes

Description The IMG_sobel filter is applied to the input image. The input image dimensions are given by the arguments 'cols' and 'rows'. The output image is 'cols' pixels wide and 'rows - 2' pixels tall.

Special Requirements

- At least four output pixels must be processed.
- The input image width must be even (eg. 'cols' must be even). rows >= 3
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_conv_3x3_i8_c8s(8 bit signed Convolution)

Function

```
  int C6ACCEL_IMG_conv_3x3_i8_c8s(C6accel_Handle hC6accel,unsigned char
*in_data, unsigned char *out_data, int cols, char *mask, int shift)
```

Parameters

- hC6accel: C6Accel handle
- in_data[ ] Input image
- out_data[ ] Output image
- cols Number of columns in the input image. Must be multiple of 8
- mask[3][3] 3x3 mask shift Shift value
- Return value: Error codes

Description The convolution kernel accepts three rows of cols input pixels and produces one output row of cols pixels using the input mask of 3 by 3. The user-defined shift value is used to shift the convolution value down to the byte range. The convolution sum is also range limited to 0.255. The shift amount is non-zero for low pass filters, and zero for high pass and sharpening filters.

Special Requirements

- cols output pixels are produced when three lines, each with a width of cols pixels, are given as input.
- cols must be a multiple of 8.
- The array pointed to by out_data should not alias with the array pointed to by in_data.
- The mask to the kernel should be such that the sum for each pixel is less than or equal to 65536. This restriction arises because of the use of the ADD2 instruction to compute two pixels in a register.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_conv_3x3_i16s_c16s(16 bit signed Convolution(with 16 bit image))

Function

```
  int C6accel_IMG_conv_3x3_i16s_c16s(C6accel_Handle hC6accel,const
short *restrict imgin_ptr, short *restrict imgout_ptr, short width,
short pitch,const short *restrict mask_ptr, short shift)
```

Parameters

- hC6accel: C6Accel handle
- imgin_ptr Pointer to input image 16-bit signed
- imgout_ptr Pointer to output image 16-bit signed
- width Number of outputs to be calculated
- pitch Number of columns in the input image
- mask_ptr Pointer to 3x3 mask used-16 bit signed
- shift User specified shift value
- Return value: Error codes

Description The convolution kernel accepts three rows of'pitch input pixels and produces one row of width output pixels using the input mask of 3 by 3. This convolution performs a point by point multiplication of 3 by 3 masks with the input image. The result of 9 multiplications are then summed to produce a 32-bit convolution intermediate sum. Overflow while accumulation is not handled. However assumptions are made on filter gain to avoid overflow. The user-defined shift value is used to shift this convolution sum down to the short range and store in an output array. The result being stored is also saturated to the -32768 to 32767 inclusive. The mask is moved one column at a time,

advancing the mask over the entire image until the entire width is covered. The input, output image pixels and the masks are provided as 16-bit signed values.

Special Requirements

- Width must be >= 2 and multiples of 2
- Pitch should be >= width
- Internal accuracy of the computations is 32 bits. To ensure correctness on a 16 bit input data, the maximum permissible filter gain in terms of bits is 16-bits i.e. the cumulative sum of the absolute values of the filter coefficients should not exceed 2^16 - 1
- Output array must be word aligned
- Input and Mask array must be half-word aligned
- The input and output arrays should not overlap
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_corr_3x3_i8_c8(8 bit mask Correlation with 8 bit image)

Function

```
 int C6accel_IMG_corr_3x3_i8_c8(C6accel_Handle hC6accel,const unsigned
char *restrict inptr, unsigned char *restrict outptr, int x_dim, const
unsigned char *restrict mask_ptr, short shift, short round)
```

Parameters

- hC6accel: C6Accel handle
- inptr Pointer to input image (8-bit signed)
- outptr Pointer to output image (32-bit signed)
- n_out Number of outputs to be calculated
- x_dim Number of columns in the input image
- mask Pointer to 3x3 mask used 16-bit signed
- shift User-specified shift amount
- round User-specified round value
- Return value: Error codes

Description The correlation performs a point by point multiplication of the 3 by 3 mask with the input image. The result of the nine multiplications are then summed up together to produce a convolution sum. A rounding constant is added to the sum and shifted by user specified amount.

The image mask to be correlated is typically part of the input image and indicates the area of the best match between the input image and mask. The mask is moved one column at a time, advancing the mask over the portion of the row specified by 'n_out'. When 'n_out' is larger than 'x_dim', multiple rows will be processed. An application may call this kernel once per row to calculate the correlation for an entire image. Alternately, the kernel may be invoked for multiple rows at a time, although the two outputs at the end of each row will have meaningless values. This will produce two rows of outputs into o_data. The outputs at locations o_data[x_dim- 2], o_data[x_dim - 1], o_data[2*x_dim - 2] and o_data[2*x_dim - 1] will have meaningless values. This is harmless, although the application must account for this when interpreting the results.

Special Requirements

- hC6accel: C6Accel handle
- The array pointed to by outptr does not alias with the array pointed to by inptr
- x_dim>=4 and is a multiple of 2
- n_out should be a multiple of 4
- This kernel is developed for LITTLE ENDIAN target

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_corr_3x3_i8_c16s(16 bit signed mask Correlation(with 8 bit image))

Function

```
  int C6accel_IMG_corr_3x3_i8_c16s(C6accel_Handle hC6accel,const
unsigned char *restrict imgin_ptr, int *restrict imgout_ptr, short
width, short pitch, const short *restrict mask_ptr)
```

Parameters

- hC6accel: C6Accel handle
- imgin_ptr Pointer to input image (8-bit signed)
- imgout_ptr Pointer to output image (32-bit signed)
- width Number of outputs to be calculated
- pitch Number of columns in the input image
- mask_ptr Pointer to 3x3 mask used (16-bit signed)
- Return value: Error codes

Description The correlation kernel accepts three rows of pitch input pixels and produces one row of width output pixels using the input mask of 3x3. This correlation performs a point-by-point multiplication of 3x3 masks with the input image. The result of the nine multiplications are then summed to produce a 32-bit sum and then stored in an output array. The mask is moved one column at a time, advancing the mask over the entire image until the entire width is covered. The masks are provided as 16-bit signed values, the input image pixels are provided as 8-bit unsigned values, and the output pixels will be 32-bit signed. The image mask to be correlated is typically part of the input image or another image.

Special Requirements

- Width must be >= 2 and multiples of 2.
- Pitch should be >= width.
- Output array must be double-word aligned.
- No alignment restrictions on input array.
- mask_ptr should be half-word aligned.
- Input and output arrays should not overlap.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_corr_3x3_i16s_c16s(16bit mask Correlation with 16 bit signed image)

Function

```
  int C6accel_IMG_corr_3x3_i16s_c16s(C6accel_Handle hC6accel,const
short *restrict imgin_ptr, int *restrict imgout_ptr, short width, short
 pitch,const short *restrict mask_ptr, int round)
```

Parameters

- hC6accel: C6Accel handle
- imgin_ptr Pointer to input image (16-bit signed)
- imgout_ptr Pointer to output image (32-bit signed)
- width Number of outputs to be calculated
- pitch Number of outputs to be calculated
- mask_ptr Pointer to 3x3 mask used (16-bit signed)
- shift User-specified shift amount

- round User-specified round value
- Return value: Error codes

Description The correlation kernel accepts three rows of pitch input pixels and produces one row of width output pixels using the input mask of 3x3. This correlation performs a point by point multiplication of 3x3 masks with the input image. The result of the 9 multiplications are then summed to produce a 40-bit sum which is added to user-specified round value, right-shifted by the specified value, and then stored in an output array. Overflow and saturation of the accumulated sum is not handled. However assumptions are made on filter gain to avoid them. The mask is moved one column at a time, advancing the mask over the entire image until the entire width is covered. The masks are provided as 16-bit signed values and the input image pixels are provided as 16-bit signed values and the output pixels will be 32-bit signed. The image mask to be correlated is typically part of the input image or another image.

Special Requirements

- Width must be >= 2 and multiples of 2
- Pitch should be >= width.
- Internal accuracy of the computations is 40 bits. To ensure correctness on a 16 bit input data, the maximum permissible filter gain in terms of bits is 24-bits i.e. the cumulative sum of the absolute values of the filter coefficients should not exceed 2^24- 1.
- Output array must be double word aligned.
- Input and mask array should be half-word aligned.
- Input and output arrays should not overlap
- Shift is appropriate to produce a 32-bit result.
- Range of filter co-efficients is -32767 to 32767.
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_median_3x3_16s(3x3 16 bit signed Median Filter)

Function

```
  int C6accel_IMG_median_3x3_16s(C6accel_Handle hC6accel,const short
*restrict i_data, int n, short *restrict o_data)
```

Parameters

- hC6accel: C6Accel handle
- i_data Pointer to input array of size 3 x n
- n Width of the input image
- o_data Pointer to output array of size 1 x n
- Return value: Error codes

Description This function performs a 3x3 median filter operation on 16-bit signed values. The median filter comes under the class of non-linear signal processing algorithms. The grey level at each pixel is replaced by the median of the nine neighboring values. The median of a set of nine numbers is the middle element so that half of the elements in the list are larger and half are smaller. The i_data points to an array which consists of three rows of pixel values. The median value is calculated corresponding to the middle row of i_data, and written into memory location pointed by o_data. The first two values in the output array will not be containing any meaningful data. The 3rd value in the output array will be the median of 2nd value in the middle row of input array and so on. The nth value in the output array will be the median of the (n-1)th value in the mid row of input array. Hence, the output array will not contain the median values corresponding to first and last elements in the middle row of input image.

Special Requirements

- The minimum value for width of input image ‘n’ is 4.

- Width of input image ‘n’ should be a multiple of 4.
- Input and output arrays must be double word aligned.
- Input and output arrays should not overlap .
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_median_3x3_16(3x3 Median filter)

Function

```
  int C6accel_IMG_median_3x3_16(C6accel_Handle hC6accel,const short
*restrict i_data, int n, short *restrict o_data)
```

Parameters

- hC6accel: C6Accel handle
- i_data Pointer to input array of size 3 x n
- n Width of the input image
- o_data Pointer to output array of size 1 x n

Description This function performs a 3x3 median filter operation on 16-bit signed values. The median filter comes under the class of non-linear signal processing algorithms. The grey level at each pixel is replaced by the median of the nine neighboring values. The median of a set of nine numbers is the middle element so that half of the elements in the list are larger and half are smaller. The i_data points to an array which consists of three rows of pixel values. The median value is calculated corresponding to the middle row of i_data, and written into memory location pointed by o_data. The first two values in the output array will not be containing any meaningful data. The 3rd value in the output array will be the median of 2nd value in the middle row of input array and so on. The nth value in the output array will be the median of the (n-1)th value in the mid row of input array. Hence, the output array will not contain the median values corresponding to first and last elements in the middle row of input image. .

Special Requirements

- The minimum value for width of input image ‘n’ is 4.
- Width of input image ‘n’ should be a multiple of 4.
- Input and output arrays must be double word aligned.
- Input and output arrays should not overlap .
- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_mulS_16s(16 bit signed Image multiplication)

Function

```
 int C6accel_IMG_mulS_16s(C6accel_Handle hC6accel,short * restrict
imgR,int * restrict imgW, short constData, int count )
```

Parameters

- hC6accel: C6Accel handle
- imgR: Read pointer for the input image
- restrict imgW: Write pointer for the output image
- constData: Constant data to be multiplied (16 bit value)
- count: Number of samples in the image
- Return value: Error codes

Description This function performs multiplication of each pixel in a image with a constant value. The image consist of 16bits per pixel. The constant is 16bits in size

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_mulS_8(8 bit signed Image Multiplication)

Function

```
  int C6accel_IMG_mulS_8(C6accel_Handle hC6accel,short * restrict
imgR,int * restrict imgW,short constData,int count )
```

Parameters

- hC6accel: C6Accel handle
- imgR: Read pointer for the input image
- restrict imgW: Write pointer for the output image
- constData: Constant data to be multiplied (16 bit value)
- count: Number of samples in the image
- Return value: Error codes

Description This function performs multiplication of each pixel in a image with a constant value. The image consist of 8 bits per pixel. The constant is 8 bits in size

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_addS_16s(16 bit signed Image addition)

Function

```
  int C6accel_IMG_addS_16s(C6accel_Handle hC6accel,short * restrict
imgR,short * restrict imgW, short constData, int count)
```

Parameters

- hC6accel: C6Accel handle
- imgR: Read pointer for the input image
- restrict imgW: Write pointer for the output image
- constData: Constant data to be multiplied (16 bit value)
- count: Number of samples in the image
- Return value: Error codes

Description This function performs addition of each pixel in a image with a constant value. The image consist of 16bits per pixel. The constant is 16bits in size

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_addS_8(8 bit signed Image Addition)

Function

```
  int C6accel_IMG_addS_8(C6accel_Handle hC6accel,char * restrict imgR,
char * restrict imgW, char constData,int count )
```

Parameters

- hC6accel: C6Accel handle
- imgR: Read pointer for the input image
- restrict imgW: Write pointer for the output image
- constData: Constant data to be multiplied (16 bit value)
- count: Number of samples in the image
- Return value: Error codes

Description This function performs addition of each pixel in a image with a constant value. The image consist of 8 bits per pixel. The constant is 8 bits in size.

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_subS_16s(16 bit signed Image subtraction)

Function

```
  int C6accel_IMG_subS_16s(C6accel_Handle hC6accel,short * restrict
imgR, short * restrict imgW, short constData,int count)
```

Parameters

- hC6accel: C6Accel handle
- imgR: Read pointer for the input image
- restrict imgW: Write pointer for the output image
- constData: Constant data to be multiplied (16 bit value)
- count: Number of samples in the image

Description This function performs subtraction of each pixel in a image with a constant value. The image consist of 16bits per pixel. The constant is 16bits in size

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_subS_8(8 bit signed Image subtraction)

Function

```
  int C6accel_IMG_subS_8(C6accel_Handle hC6accel,char * restrict imgR,
char * restrict imgW, char constData,int count)
```

Parameters

- hC6accel: C6Accel handle
- imgR: Read pointer for the input image
- restrict imgW: Write pointer for the output image
- constData: Constant data to be multiplied (16 bit value)
- count: Number of samples in the image

Description This function performs subtraction of each pixel in a image with a constant value. The image consist of 8bits per pixel. The constant is 8bits in size

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory

int C6accel_IMG_yc_demux_8(8 Bit Deinterlacing)

Function

```
  int C6accel_IMG_yc_demux_8(C6accel_Handle hC6accel,int n, const
unsigned char *restrict yc,unsigned char *restrict y, unsigned char
*restrict cr, unsigned char *restrict cb)
```

Parameters

- hC6accel: C6Accel handle
- n :Number of luma pixels
- yc: Interleaved luma/chroma
- y : Luma plane (8-bit)
- cr: Cr chroma plane (8-bit)
- cb: Cb chroma plane (8-bit)
- Return value: Error codes

Description:

This function reads the interleaved byte-oriented pixel data and then writes it to the appropriate result array. As thedata width does not change during de-interleaving, no sign extension or zero-filling is performed. The data is expected to be in an order consistent with reading byte oriented data from a word-oriented peripheral that is operating in LITTLE ENDIAN mode, while the CPU is in LITTLE ENDIAN mode. This function unpacks the byte-oriented data so that further processing may proceed in LITTLE ENDIAN mode.

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory
- Input and output arrays are double-word aligned.
- The input must be a multiple of 32 luma pixels long

int C6accel_IMG_yc_demux_16(16 bit Deinterlacing)

Function

```
  int C6accel_IMG_yc_demux_16(C6accel_Handle hC6accel,int n, const
unsigned short * yc, short *restrict y, short *restrict cr, short
*restrict cb)
```

Parameters

- hC6accel: C6Accel handle

n :Number of luma pixels
yc: Interleaved luma/chroma
y : Luma plane (16-bit)
cr: Cr chroma plane (16-bit)
cb: Cb chroma plane (16-bit)

- Return value: Error codes

Description:

This function reads the byte-oriented pixel data, zero-extends it, and then writes it to the appropriate result array. Both the luma and chroma values are expected to be unsigned. The data is expected to be in an order consistent with reading byte oriented data from a word-oriented peripheral that is operating in LITTLE ENDIAN mode, while the CPU is in LITTLE ENDIAN mode. This function unpacks the byte-oriented data so that further processing may proceed in LITTLE ENDIAN mode.

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory
- Input and output arrays are double-word aligned.
  The input must be a multiple of 16 luma pixels long

int C6accel_IMG_yuv420pl_to_rgb565(8 bit YUV420 to RGB565 Color Space Conversion)

Function

```
  int C6accel_IMG_yuv420pl_to_rgb565(C6accel_Handle hC6accel,const
short *coeff,int Height, Int Width, unsigned char *pbuf_y, unsigned
char *pbuf_cb, unsigned char *pbuf_cr,unsigned short *pbuf_rgb)
```

Parameters

- hC6accel: C6Accel handle
- coeff[5] Matrix coefficients
- y_data Luminence data (Y'). Must be double-word aligned.
- cb_data Blue color-diff (B'-Y'). Must be word aligned.
- cr_data Red color-diff (R'-Y'). Must be word aligned.
- rgb_data RGB 5:6:5 packed pixel out. Must be double-word aligned.
- num_pixels Number of luma pixels to process. Must be multiple of 8.
- Return value: Error codes

Description This kernel performs Y'CbCr to RGB conversion. The coeff[] array contains the color-space-conversion matrix coefficients. The y_data, cb_data and cr_data pointers point to the separate input image planes. The rgb_data pointer points to the output image buffer, and must be word aligned. The kernel is designed to process 4:2:0 image data. The coefficients in the coeff array must be in signed Q13 form. This code can perform various flavors of Y'CbCr to RGB conversion, as long as the offsets on Y, Cb, and Cr are -16, -128, and -128, respectively, and the coefficients match the pattern shown. The kernel implements the following matrix form, which involves 5 unique coefficients:

```
coeff[] = { 0x2000, 0x2BDD, -0x0AC5, -0x1658, 0x3770 };
[ 1.0000 0.0000 1.3707 ] [ Y' - 16 ] [ R']
[ 1.0000 -0.3365 -0.6982 ] * [ Cb - 128 ] = [ G']
[ 1.0000 1.7324 0.0000 ] [ Cr - 128 ] [ B']
```

Below are some common coefficient sets, along with the matrix equation that they correspond to. Coefficients are in signed Q13 notation, which gives a suitable balance between precision and range. Y'CbCr -> RGB conversion with RGB levels that correspond to the 219-level range of Y'. Expected ranges are [16..235] for Y' and [16..240] for Cb and Cr.

```
[ coeff[0] 0.0000 coeff[1] ] [ Y' - 16 ] [ R']
[ coeff[0] coeff[2] coeff[3] ] * [ Cb - 128 ] = [ G']
[ coeff[0] coeff[4] 0.0000 ] [ Cr - 128 ] [ B']
```

Y'CbCr -> RGB conversion with the 219-level range of Y' expanded to fill the full RGB dynamic range. (The matrix has been scaled by 255/219). Expected ranges are [16..235] for Y' and [16..240] for Cb and Cr.

```
[ 1.1644 0.0000 1.5960 ] [ Y' - 16 ] [ R']
[ 1.1644 -0.3918 -0.8130 ] * [ Cb - 128 ] = [ G']
[ 1.1644 2.0172 0.0000 ] [ Cr - 128 ] [ B']
```

Other scalings of the color differences (B'-Y') and (R'-Y') (sometimes incorrectly referred to as U and V) are supported, as long as the color differences are unsigned values centered around 128 rather than signed values centered around 0, as noted above. In addition to performing plain color-space conversion, color saturation can be adjusted by scaling coeff[1] through coeff[4]. Similarly, brightness can be adjusted by scaling coeff[0]. However, general hue adjustment cannot be performed, due to the two zeros hard-coded in the matrix.

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory
- The number of luma samples to be processed must be a multiple of 8.
- The input Y array and the output image must be double-word aligned.
- The input Cr and Cb arrays must be word aligned.

int C6accel_IMG_ycbcr422pl_to_rgb565(8 bit YUV422 to RGB565 Color Space Conversion)

Function

```
  int C6accel_IMG_ycbcr422pl_to_rgb565(C6accel_Handle hC6accel,const
short * restrict coeff, const unsigned char * restrict y_data, const
unsigned char * restrict cb_data, const unsigned char * restrict
cr_data, unsigned short * restrict rgb_data, unsigned num_pixels)
```

Parameters

- hC6accel: C6Accel handle
- coeff[5] Matrix coefficients
- y_data Luminence data (Y'). Must be double-word aligned.
- cb_data Blue color-diff (B'-Y'). Must be word aligned.
- cr_data Red color-diff (R'-Y'). Must be word aligned.
- rgb_data RGB 5:6:5 packed pixel out. Must be double-word aligned.
- num_pixels Number of luma pixels to process. Must be multiple of 8.
- Return value: Error codes

Description This kernel performs Y'CbCr to RGB conversion. The coeff[] array contains the color-space-conversion matrix coefficients. The y_data, cb_data and cr_data pointers point to the separate input image planes. The rgb_data pointer points to the output image buffer, and must be word aligned. The kernel is designed to process arbitrary amounts of 4:2:2 image data, although 4:2:0 image data may be processed as well. For 4:2:2 input data, the y_data, cb_data and cr_data arrays may hold an arbitrary amount of image data. For 4:2:0 input data, only a single scan-line (or portion thereof) may be processed at a time. The coefficients in the coeff array must be in signed Q13 form. This code can perform various flavors of Y'CbCr to RGB conversion, as long as the offsets on Y, Cb, and Cr are -16, -128, and -128, respectively, and the coefficients match the pattern shown. The kernel implements the following matrix form, which involves 5 unique coefficients:

```
coeff[] = { 0x2000, 0x2BDD, -0x0AC5, -0x1658, 0x3770 };
[ 1.0000 0.0000 1.3707 ] [ Y' - 16 ] [ R']
[ 1.0000 -0.3365 -0.6982 ] * [ Cb - 128 ] = [ G']
```

```
[ 1.0000 1.7324 0.0000 ] [ Cr – 128 ] [ B’]
```

Below are some common coefficient sets, along with the matrix equation that they correspond to. Coefficients are in signed Q13 notation, which gives a suitable balance between precision and range. Y’CbCr -> RGB conversion with RGB levels that correspond to the 219-level range of Y’. Expected ranges are [16..235] for Y’ and [16..240] for Cb and Cr.

```
[ coeff[0] 0.0000 coeff[1] ] [ Y’– 16 ] [ R’]
[ coeff[0] coeff[2] coeff[3] ] * [ Cb – 128 ] = [ G’]
[ coeff[0] coeff[4] 0.0000 ] [ Cr – 128 ] [ B’]
```

Y’CbCr -> RGB conversion with the 219-level range of Y’ expanded to fill the full RGB dynamic range. (The matrix has been scaled by 255/219). Expected ranges are [16..235] for Y’ and [16..240] for Cb and Cr.

```
[ 1.1644 0.0000 1.5960 ] [ Y’– 16 ] [ R’]
[ 1.1644 –0.3918 –0.8130 ] * [ Cb – 128 ] = [ G’]
[ 1.1644 2.0172 0.0000 ] [ Cr – 128 ] [ B’]
```

Other scalings of the color differences (B'-Y') and (R'-Y') (sometimes incorrectly referred to as U and V) are supported, as long as the color differences are unsigned values centered around 128 rather than signed values centered around 0, as noted above. In addition to performing plain color-space conversion, color saturation can be adjusted by scaling coeff[1] through coeff[4]. Similarly, brightness can be adjusted by scaling coeff[0]. However, general hue adjustment cannot be performed, due to the two zeros hard-coded in the matrix.

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory
- The number of luma samples to be processed must be a multiple of 8.
- The input Y array and the output image must be double-word aligned.
- The input Cr and Cb arrays must be word aligned.

int C6accel_IMG_yuv420pl16_to_rgb565(16 bit YUV420 to RGB565 Color Space Conversion)

Function

```
  int C6accel_IMG_yuv420pl16_to_rgb565(C6accel_Handle hC6accel,const
short *coeff,int Height, Int Width, short *pbuf_y, short *pbuf_cb,
short *pbuf_cr,unsigned short *pbuf_rgb)
```

Parameters

- hC6accel: C6Accel handle
- coeff[5] Matrix coefficients
- y_data Luminence data (Y’). Must be double-word aligned.
- cb_data Blue color-diff (B’-Y’). Must be word aligned.
- cr_data Red color-diff (R’-Y’). Must be word aligned.
- rgb_data RGB 5:6:5 packed pixel out. Must be double-word aligned.
- num_pixels Number of luma pixels to process. Must be multiple of 8.
- Return value: Error codes

Description

This kernel performs Y'CbCr to RGB conversion and is a 16 bit implementation of the C6ACCEL_IMG_YUV420P_to_RGB565 described above. From the Color FAQ, Click here [1]

This kernel can perform various flavors of Y'CbCr to RGB conversion as long as the offsets on Y, Cb, and Cr are -16,-128, and -128, respectively, and the coefficients match the pattern shown. The kernel implements the following matrix form, which involves 5 unique coefficients:

```
           [ coeff[0] 0.0000   coeff[1] ]   [ Y' -  16 ]     [ R']

           [ coeff[0] coeff[2] coeff[3] ] * [ Cb - 128 ]  =  [ G']
           [ coeff[0] coeff[4] 0.0000   ]   [ Cr - 128 ]     [ B']
```

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory
- The number of luma samples to be processed must be a multiple of 8.
- The input Y array and the output image must be double-word aligned.
- The input Cr and Cb arrays must be word aligned.

int C6accel_IMG_ycbcr_422pl16_to_rgb565(16 bit YUV422 to RGB565 Color Space Conversion)

Function

```
  int C6accel_IMG_ycbcr_422pl16_to_rgb565(C6accel_Handle hC6accel,const
 short * restrict coeff,
                       const unsigned char * restrict y_data, const
unsigned char * restrict cb_data ,
                       const unsigned char * restrict cr_data, unsigned
 short * restrict rgb_data, unsigned num_pixels)
```

Parameters

- hC6accel: C6Accel handle
- coeff[5] Matrix coefficients
- y_data Luminence data (Y'). Must be double-word aligned.
- cb_data Blue color-diff (B'-Y'). Must be word aligned.
- cr_data Red color-diff (R'-Y'). Must be word aligned.
- rgb_data RGB 5:6:5 packed pixel out. Must be double-word aligned.
- num_pixels Number of luma pixels to process. Must be multiple of 8.
- Return value: Error codes

Description

This kernel performs Y'CbCr to RGB conversion and is a 16 bit implementation of the C6ACCEL_IMG_YUV422P_to_RGB565 described above. From the Color FAQ, Click here [1]

This kernel can perform various flavors of Y'CbCr to RGB conversion as long as the offsets on Y, Cb, and Cr are -16,-128, and -128, respectively, and the coefficients match the pattern shown. The kernel implements the following matrix form, which involves 5 unique coefficients:

```
           [ coeff[0] 0.0000   coeff[1] ]   [ Y' -  16 ]     [ R']

           [ coeff[0] coeff[2] coeff[3] ] * [ Cb - 128 ]  =  [ G']
           [ coeff[0] coeff[4] 0.0000   ]   [ Cr - 128 ]     [ B']
```

Special Requirements

- Buffers/vectors being passed to the codec must be assigned contiguous memory

- The number of luma samples to be processed must be a multiple of 8.
- The input Y array and the output image must be double-word aligned.
- The input Cr and Cb arrays must be word aligned.

int C6accel_IMG_ycbcr422sp_to_ycbcr420pl(8 bit YUV422 semiplanar to YUV420 planar Color Space Conversion)

Function

```
  int C6accel_IMG_ycbcr422sp_to_ycbcr420pl(C6accel_Handle hC6accel,
const unsigned char * y_src,
                                          const unsigned char
*cbcr_src, unsigned char * restrict y_dst,
                                          unsigned char * restrict
cb_dst, unsigned char * restrict cr_dst,
                                          unsigned int num_lines,
unsigned int width,  unsigned int src_pitch,
                                          unsigned int dst_y_pitch,
unsigned int dst_cbcr_pitch)
```

Parameters

- hC6accel: C6Accel handle
- y_src Luminance data of YUV422sp input
- cbcr_src Chrominance data of YUV422 semiplanar input
- y_dst Luminence data of YUV420 planar output(Y').
- cb_data Blue color-diff (B'-Y') output (yuv420 planar.)
- cr_data Red color-diff (R'-Y') output(in 420 planar format).
- num_lines Number of lines in the image.
- Width: Width of image data. Must be multiple of 16
- src_pitch: Pitch of the input image
- dst_y_pitch: Pitch of the ouput luminance image
- dst_cbcr_pitch: Pitch of the output chrominance
- Return value: Error codes

Description

This kernel performs Y'CbCr 422 semiplanar format to 420 planar conversion.

Special requirements:

- Input Data must be aligned to double word boundary.
- Width of the image data must be multiple of 16

int C6accel_IMG_ycbcr422pl_to_ycbcr422sp(8 bit YUV422 planar to YUV422 semi planar Color Space Conversion)

Function

```
  int C6accel_IMG_ycbcr422pl_to_ycbcr422sp(C6accel_Handle hC6accel,
const unsigned char * y_dst,
                                         const unsigned char *
cbcr_dst, unsigned char * restrict y_src,
                                         unsigned char * restrict
```

```
cb_src, unsigned char * restrict cr_src,
                                          unsigned int num_lines,
unsigned int width,unsigned int dst_pitch,
                                          unsigned int src_y_pitch,
unsigned int src_cbcr_pitch)
```

Parameters

- hC6accel: C6Accel handle
- y_dst Luminance data of YUV422sp output
- cbcr_src Chrominance data of YUV422 semiplanar output
- y_src Luminence data of YUV420 planar input(Y').
- cb_src Blue color-diff (B'-Y') input (yuv422 planar.)
- cr_src Red color-diff (R'-Y') input(in 422 planar format).
- num_lines Number of lines in the image.
- Width: Width of image data. Must be multiple of 16
- dst_pitch: Pitch of the output image
- src_y_pitch: Pitch of the source luminance image
- src_cbcr_pitch: Pitch of the source chrominance(Should be same for Cb and Cr component)
- Return value: Error codes

Description

This kernel performs Y'CbCr 422 semiplanar format to 422 planar conversion.

Special requirements:

- Input Data must be aligned to double word boundary.
- Width of the image data must be multiple of 16

int C6accel_IMG_ycbcr422sp_to_ycbcr422ile(8 bit YUV422 semiplanar to YUV422 interleaved Color Space Conversion)

Function

```
  int C6accel_IMG_ycbcr422sp_to_ycbcr422ile(C6accel_Handle hC6accel,
const unsigned char * y_src,
                                           const unsigned char *
cbcr_src, unsigned char * restrict ycbcr_dst,
                                           unsigned int num_lines,
unsigned int width, unsigned int src_pitch,
                                           unsigned int
dst_ycbcr_pitch)
```

Parameters

- hC6accel: C6Accel handle
- y_src Luminance data of YUV422sp input
- cbcr_src Chrominance data of YUV422 semiplanar input
- ycbcr_dst Luminence data of YUV422 interleaved output(Y').
- num_lines Number of lines in the image.
- Width: Width of image data. Must be multiple of 16
- src_pitch: Pitch of the output image
- dst_ycbcr_pitch: Pitch of the source chrominance

- Return value: Error codes

Description

This kernel performs Y'CbCr 422 semiplanar format to 422 interleaved format conversion.

Special requirements:

- Input Data must be aligned to double word boundary.
- Width of the image data must be multiple of 16

Optimized Kernels:

All of the below kernels are optimized versions of their corresponding kernels mentioned above. They have been optimized to give better performance on the ARM specifying the number of rows and columns in the image to the DSP.

Return to C6Accel Documentation

Click here.

References

[1] http://home.inforamp.net/~poynton/ColorFAQ.html.

Article Sources and Contributors

Image Processing Kernels in C6Accel *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=62050 *Contributors*: A0272049