Understanding the DaVinci Preview Engine

Xiangdong Fu

ABSTRACT

The Preview Engine block in the DaVinci video processing sub-system (VPSS) provides some critical functions for image and video processing. These functions, if implemented in software, require a significant number of computations in terms of million instructions per second (MIPs). By offloading these functions, the valuable MIPs can be used for more differentiating tasks, such as video compression and content analysis.

This application report provides an overview of the Preview Engine. Discussions are focused on the usage of each sub-block from an application development point of view. Examples are provided using the Linux® Preview Engine driver that is developed by Texas Instruments.

This application report contains project code that can be downloaded from this link. http://www-s.ti.com/sc/techlit/sprc434.gz. The example files attached are created in Linux and are to be extracted, built and executed in Linux. Please use tar -xzf file_name to uncompress the archives in Linux.

Contents

List of Figures

List of Tables

1 Overview

The Preview Engine is primarily a hardware block that performs image pre-processing tasks. It is commonly referred to as the Image PIPE, or IPIPE. It is input from either the CDC controller (CCDC) or from the dual data rate (DDR) memory controller. The input image is always in RGB Bayer pattern format. The output image is always in Y/Cb/Cr 4:2:2 format. The processing tasks performed there include color filter array (CFA) interpolation, color space conversion, gamma correction, and other image enhancement tasks, such as noise filtering and RGB blending. The purpose of this block is to get the input image in the right format, with enhanced quality so it is ready for further processing.

The Preview Engine is part of the video processing front end (VPFE) in the VPSS, shown in Figure 1.

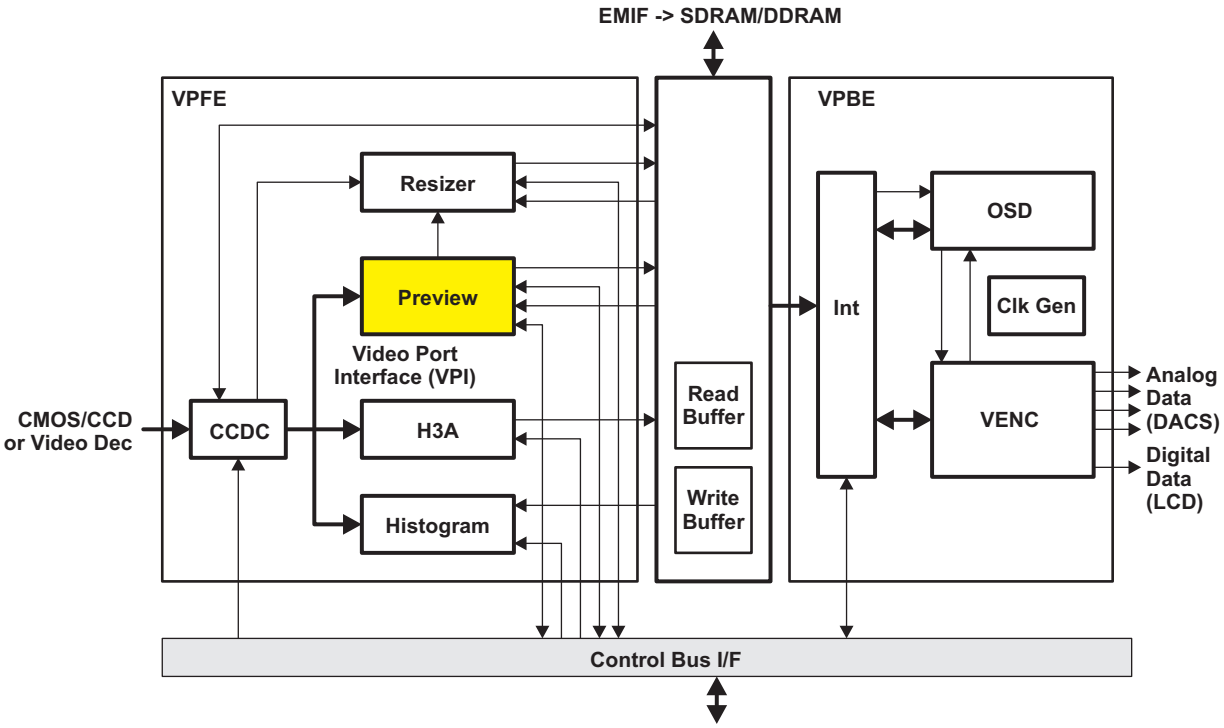

Figure 1. VPSS Top Level Preview Engine Diagram

The Preview Engine is useful only when the input data from CCDC is in RGB Bayer pattern format, which is common for CMOS and CCD sensors widely used in digital still cameras, camcorders, and other camera systems. The term Bayer pattern refers to a data format in which, at each pixel location, only one of the three primary colors [red (R), green (G) and blue (B) (RGB)] is available, rather than all of them, as shown in Figure 2.

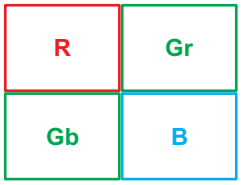

Figure 2. Bayer Pattern Data Format

The primary task of the Preview Engine is to convert the Bayer pattern input data into Y/Cb/Cr 4:2:2 format, as shown in Figure 3.

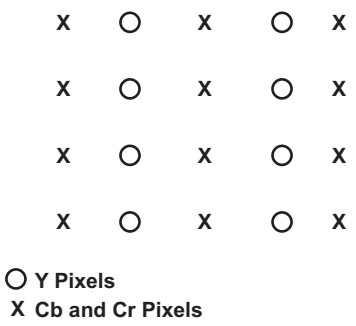

Figure 3. Y/Cb/Cr 4:2:2 Format

In this format, The Y signal is the luminance, and Cb and Cr are chrominance signals. This color space is widely used for image compression and transmission because it is less redundant than the RGB color space. The 4:2:2 notion suggests that the chrominance signals are 2:1 down-sampled, as shown in Figure 3, only every other pixel locations have Cb and Cr components.

Figure 4 shows the block diagram of the Preview Engine. There are a number of processing sub-blocks and configurations can get fairly complicated. As discussed above, the primary function of the Preview Engine is to convert the image from RGB Bayer pattern to Y/Cb/Cr 4:2:2. The sub-blocks for doing that are CFA interpolation and color space conversion. The CFA performs interpolation to get the missing two color components in the Bayer pattern input. The full color RGB image is then converted to Y/Cb/Cr 4:2:2. The other blocks in Figure 4 are for image enhancement and can be turned off without affecting the basic functionality of the Preview Engine, however, using all of them gives the best image quality.

The following are the sub-functional blocks in the Preview Engine. Each is discussed in the next section.

- Input Formatter/Averager
- Inverse A-law
- Dark Frame Write
- Dark Frame Subtraction/Lens Shading Compensation
- Noise Filter
- White Balance
- CFA Interpolation
- Black Adjustment
- RGB-to-RGB Blending
- Gamma Correction
- RGB-to-YCbCr Conversion
- Luminance Enhancement and Chrominance Suppression

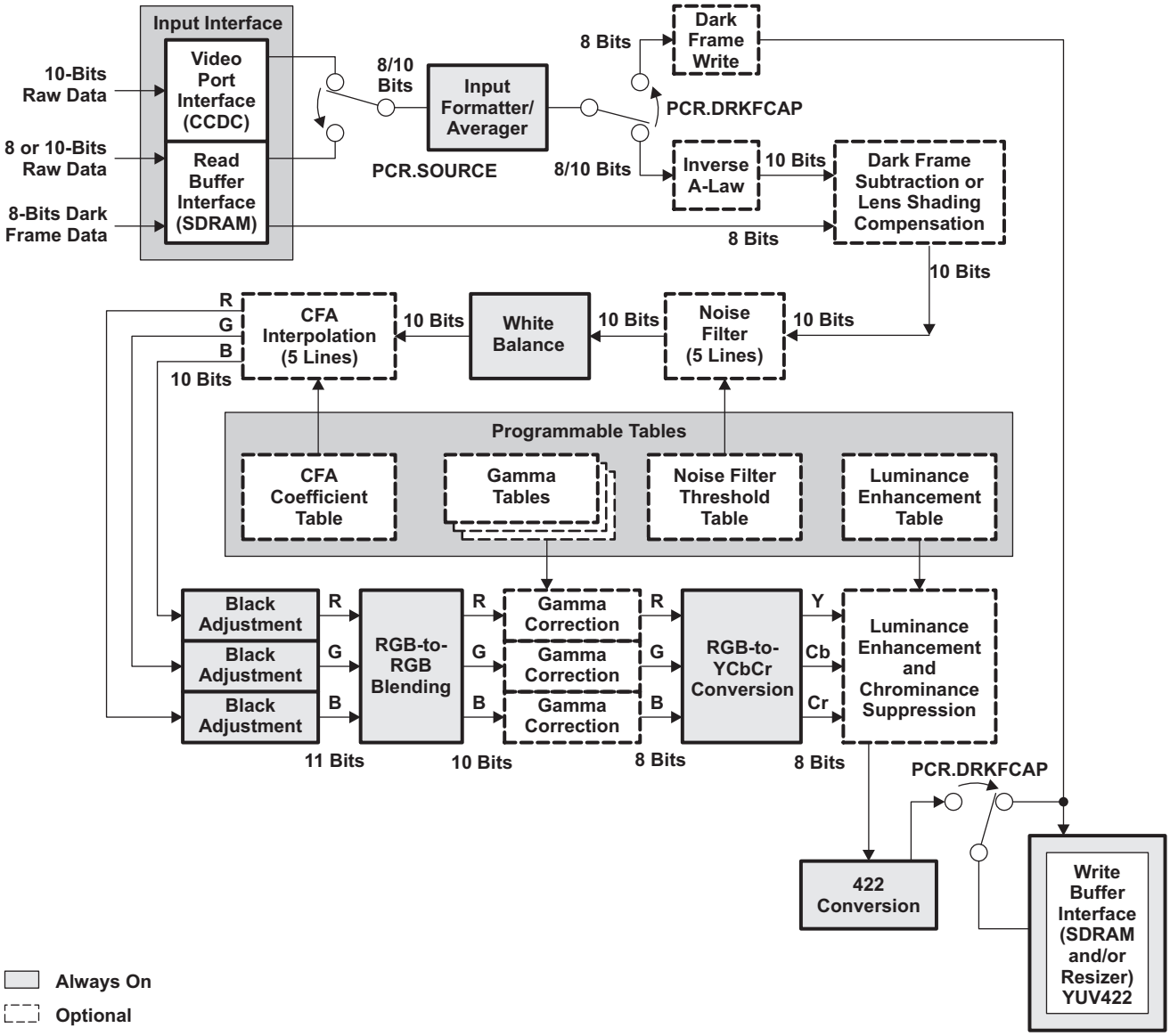

Figure 4. Preview Engine Block Diagram

2 Functional Blocks

In this section, the supported and non-supported features are discussed first. Then, each of the functional sub-blocks are discussed in more detail. For each block, we discuss its primary functionality, as well as the algorithms behind the hardware implementation.

2.1 Supported Features and Non-Supported Features

The following are the supported features. Most features are performed by an underlying functional sub-block. Please refer to the *TMS320DM644x DMSoC Video Processing Front End (VPFE) User's Guide* (SPRUE38) for more details.

- Support for conventional Bayer pattern color sensors
- Support for accepting the input image/video data from either the CCDC or the DDRAM
- Support for an output width up to 1280 pixels per line
- Simple horizontal averaging (by factors of 2, 4, or 8) to handle input widths greater than 1280 (plus the cropped number) pixels wide
- Dark frame capture and subtraction or lens shading compensation
- A-law decompression to transform non-linear 8-bit data to 10-bit linear data
- A programmable noise filter that operates on a 3×3 grid of the same color
- Digital gain and white balance (color separate gain for white balance).
- Programmable CFA interpolation that operates on a 5×5 grid
- Programmable RGB-to-RGB blending matrix (9 coefficients for the 3×3 matrix)
- Fully programmable gamma correction
- Programmable color conversion (RGB to YUV) coefficients and offsets
- Luminance enhancement (non-linear) and chrominance suppression and offset

The following are non-supported features. Please refer to the *TMS320DM644x DMSoC Video Processing Front End (VPFE) User's Guide* (SPRUE38) for more detail.

- Edge interpolation is not performed in any of the sub-functional blocks. As a result, certain pixels/lines are automatically/mandatorily cropped when the corresponding block is enabled. If all blocks are enabled, a total of 14 pixels per line (7 left most and 7 right most) and 8 lines (4 top most and 4 bottom most) are not output.
- Does not support output width of more than 1280 pixels per line. This is due to the size limitation of internal line buffers.
- Does not support an output format other than YCbCr 4:2:2 format
- Does not support input format in the YCbCr domain

2.2 Input Averager

To support sensors that output greater than 1280 pixels per line, an input averager is incorporated to down-sample by factors of 1 (no averaging), 2, 4, or 8 in the horizontal direction. The horizontal distance between two consecutive pixels of the same color to be averaged is selectable between 1, 2, 3, or 4 for both even and odd lines. For Bayer pattern input, the distance must be configured as 2 for both even and odd lines. The valid output of the input averager is either 8 or 10 bits wide.

CAUTION

This block precedes all other functional blocks in the Preview Engine. However, because it does not make sense to down-sample the A-law compressed data by simple averaging, this block must be disabled when the Inverse A-law block is enabled, or vice versa.

2.3 Inverse A-law

The inverse A-law block performs inverse A-law transform to convert 8-bit A-law compressed data back to 10-bit. This feature is useful when the Preview Engine gets input from DDR memory. When data is captured, CCDC uses A-law to compress 10-bit data to 8-bit before storing it to memory. This results to 50% memory savings because each 10-bit data takes two bytes for storage. When this block is disabled, 8-bit input data was shifted left by 2 to make 10-bit data. 10-bit input data simply passes through.

CAUTION

This block can not be enabled while the input averager is also enabled.

2.4 Dark Frame Write

The black level input from a typical CMOS/CCD sensor may not be truly black (zero) and may vary from pixel to pixel. To compensate, most sensors can output a dark frame, which then can be subtracted from the actual image. The dark frame write module bypasses all other sub-blocks except the input averager and stores the input data directly to DDR memory in 8-bit values. When the input value is greater than 255, it is saturated to 255. Although black level input can be as big as 1023 for 10-bit input, it should be close to zero. Big black level input usually signals a bad pixel, which can be corrected in the CCDC by the fault pixel correction block.

2.5 Dark Frame Subtraction/Lens Shading Compensation

If enabled, the dark frame subtraction/lens shading compensation block fetches a dark frame from DDR memory subtracting it pixel-by-pixel to the incoming input video frame. The output of the dark frame subtract is 10 bits wide (U10Q0).

Optionally, this block can perform lens shading compensation. In this case, the 8-bit value fetched from DDR memory is multiplied with the incoming pixel and the result is right shifted. The number of shifts (0-7) is programmable.

Lens shading compensation can be used to correct the intensity fall-off at the edges of the image sensor due to the optical lens system.

This block can perform either dark frame subtraction or lens shading compensation, but not both.

2.6 Horizontal Mean Filter

The horizontal mean filter block is useful for reducing temperature-induced noise effects. It tries to smooth out those pixels that are far brighter or darker than its neighbors. If enabled, it calculates the absolute difference between the current pixel (I) and pixel (i-X) and between the current pixel (i) and pixel (i+X). If the absolute difference exceeds a threshold, and the sign of the differences is the same, then the average of pixel (i-X) and pixel (i+X) replaces pixel (i). The horizontal distance (X) between two consecutive pixels must be configured as 2 for Bayer pattern input for both even and odd lines.

2.7 Noise Filter

The purpose of the noise filter block is to smooth regions that appear to be uniform.

This is basically a programmable filter that operates on a 3×3 grid of **same-color** pixels to reduce the noise in the image data. This filter always operates on nine pixels of the **same color**. An 8-bit threshold is obtained on indexing the current pixel into a 256-entry table. If the absolute difference of the current pixel and each of its eight neighbors is less than the threshold, that neighboring pixels are used in computing a weighted average as shown in Figure 4.The threshold should ideally be set to exclude the far-apart-value neighbors and average the noise among the remaining same-color pixels. The average is then weighted with the current pixel to obtain the replacement noise-filtered pixel.

2.8 White Balance

The purpose of the white balance is to adjust the gains for each color component so that a white color object, such as a wall, is captured and displayed as **WHITE**, rather than blue or orange in certain lighting conditions, especially indoor.

2.9 CFA Interpolation

The term CFA refers to the color filter array that is positioned on top of the CCD or CMOS sensor to filter out red, green, and blue components of light at each pixel position. This is a common cost-reduction technique used by CCD and CMOS sensor manufactures. The most commonly used CFA pattern is Bayer pattern, as shown below. An image sensor with Bayer pattern CFA captures image in Bayer pattern format, as shown in Figure 2.

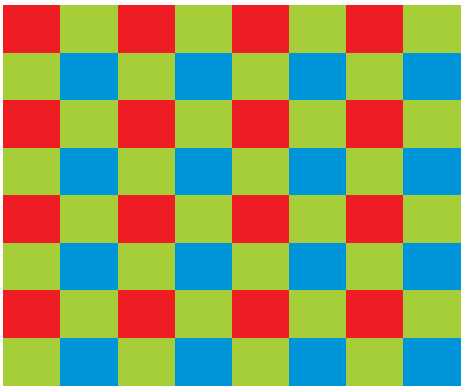

Figure 5. Bayer Pattern CFA

Because the Bayer pattern CFA outputs one color at each pixel location, the purpose of the CFA Interpolation block is to interpolate the missing two color components at each pixel to output an RGB image with fully populated colors. This processing is also widely referred to as CFA demosaicing.

Scientific research has been conducted on CFA demosaicing, but there currently is no industry standard on this process. The most advanced algorithms use gradient-based adaptive interpolation techniques to minimize artifacts around object edges. The algorithm implemented in the Preview Engine is TI proprietary.

2.10 Black Level Adjustment

The purpose of black level adjustment is to make black appear **BLACK** in the final image. A fixed offset is subtracted from each of the R, G, and B color components.

2.11 RGB-to-RGB Blending

The purpose of the RGB-to-RGB blending block is to adjust the colors of the input image so that the R, G and B colors are in the standard sRGB color space. The implementation is done by multiplying a 3×3 RGB blending matrix with the three RGB values of each pixel. Note that this block can also be used for white balancing adjustment.

2.12 Gamma Correction

The purpose of the gamma correction block is to compensate the non-linearity of the display device (for example, a CRT monitor) as shown in Figure 6.

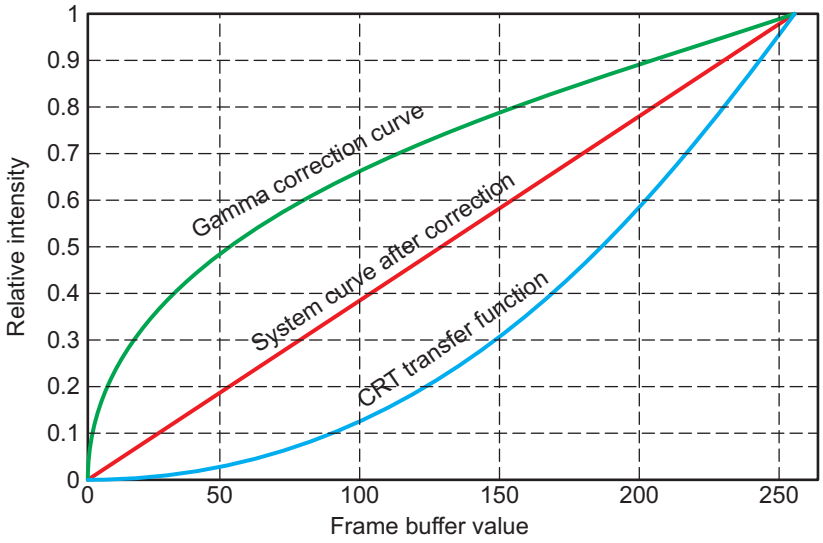

Figure 6. Gamma Correction

2.13 RGB-to-YCbCr Conversion

The RGB-to-YCbCr conversion block performs color space conversion from sRGB to YCbCr using standard coefficients. The output of this block is interleaved YCbCr 4:2:2 data, as shown in Figure 3. The YCbCr data format is commonly used by video codecs, such as MPEG-2 and H.264. At first, 10-bit RGB data is converted to 8-bit Y/Cb/Cr 4:4:4 data. Then the chrominance data is 2:1 down-sampled by averaging the neighboring 2 pixels to generate 8-bit 4:2:2 data.

2.14 Luminance Enhancement/Chrominance Suppression

The luminance enhancement/chrominance suppression blocks reside in the middle of the RGB-YCbCr color space conversion step.

The purpose of luminance enhancement is to sharpen the edges blurred by some of the previous processing steps, such as the CFA interpolation.

The purpose of chrominance suppression is to reduce the chrominance values cb and cr in very bright areas, i.e., when the luminance value is bigger than a pre-defined threshold. This is intended to correct occasional false color effect in very bright areas and also can be used to correct color errors introduced by the CFA interpolation in the horizontal direction.

2.15 4:2:2 Conversion

The final stage of the Image Pipe is the brightness and contrast adjustment. This function deals only with luminance data. While brightness adjustment adds a fixed offset to the luminance (Y) data, contrast adjustment multiplies a fixed gain to the luminance data.

3 Programming the Preview Engine

For programming details of the Preview Engine, please refer to the DM6446 VPFE PRG.

3.1 Input and Output Sizing

Because no edge interpolation is performed, certain filtering operations in the Preview Engine crop several pixels and lines at the boarders of the image. Table 1 summaries the cropping. As a result, if the output image is required to be at a certain standard size, the input image must be larger to allow for the extra cropping. For example, for the Preview Engine to output HD 720p resolution image, which has a size of 1280×720, the input image must be at least 1290×726 if all the processing functions in Table 1 are enabled. Because the input image is normally captured by the CCDC, the CCDC parameters must be set to accommodate this requirement.

Table 1. Image Cropping by Preview Engine

Image Cropping by Preview Functions		
Function	**Pix/Line**	**Lines**
Noise Filter	4	4
CFA (Bayer pattern)	4	4
CFA (2x down-sampling)	0	2
Color suppression or luminance enhancement	2	0
Maximum total	10	8

3.2 Dark Frame Subtraction

To use the dark frame subtraction function, a dark frame must first be captured and stored to DDR memory using the dark frame write function of the Preview Engine. This can be done by enabling the *PCR.DRKFCAP* bit. Once the dark frame is captured, the *PCR.DRKFCAP* bit needs to be cleared. The dark frame subtraction can be enabled by setting the *PCR.DRKFEN* bit.

The Preview Engine pre-fetches dark frame data to avoid potential underflow of its internal dark frame buffer. The pre-fetch starts when the *PCR.DRKFEN* bit is set, even when the Preview Engine is not enabled (*PCR.ENABLE* bit is zero) or as soon as it has finished processing the previous frame while the *PCR.DRKFEN* bit is set.

CAUTION

Caution must be taken when the dark frame memory address needs to be changed by writing to the *DSDR_ADDR* register. If *DSDR_ADDR* is updated directly with the *PCR.DRKFEN* bit set, then the Preview Engine does not perform the pre-fetch from the newly updated address and its internal buffer is loaded with data from the old dark frame. As a result, the first several lines of the next output image are incorrect until the internal dark frame buffer is depleted with the old data.

To ensure correct operation, every time before the *DSDR_ADDR* register is changed, the *PCR.DRKFEN* bit must be disabled. Then, this bit must be enabled after the address change is complete. This ensures the Preview Engine pre-fetch from the correct memory location. Figure 7 compares the incorrect and correct ways of setting up the dark frame subtraction function.

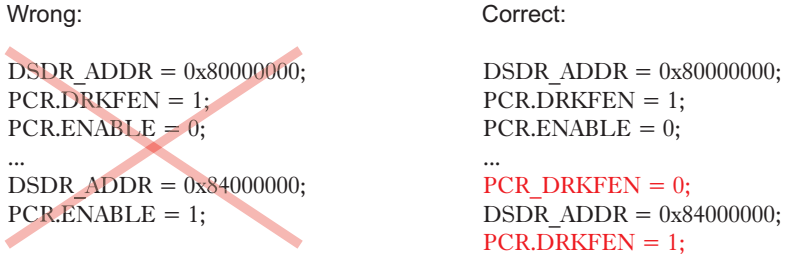

Figure 7. CORRECT Ways of Setting Up Dark Frame Subtraction Function

The discussion above on dark fram subtraction also applies to the lens shading compensation. To perform this function, the *PCR.SHADE_COMP* bit must be set in addition to the *PCR.DRKFEN bit*.

3.3 Avoiding Write Buffer Overflow

The Preview Engine hardware is designed in such a way that its operation is regulated by the input rate, without regard to the status of its write buffer. This is not a problem when the input is coming from the CCDC and the data rate has already been regulated and restricted. However, when input data is coming from DDR memory, the input data rate can be as high as 400 Mbytes/second, putting a huge pressure on the system DDR bandwidth. This also can cause overflow of its write buffer, where processed data is temporarily stored before being written to the DDR memory.

To alleviate this problem, the input rate must be reduced significantly. The SDR_REQ_EXP register is there for this purpose.

The SDR_REQ_EXP register can be used to insert delays between consecutive read requests from the Resizer, the Preview Engine and the Histogram module. For the Preview Engine, the field is *SDR_REQ_EXP.PRV_EXP* and the value ranges from 0 – 0x3FF, with 0 as no delay and 0x3FF as maximum delay. The actual delay is *SDR_REQ_EXP.RESZ_EXP*32 cycles*. The maximum clock is 200 MHz if the device is running full-speed at 600 MHz.

When overflow occurs in the Preview Engine write buffer memory, the VPSS_PCR.PRV_WBL_O field is set to indicate this condition. This field can be cleared by writing a 1 to them.

3.4 Tuning Parameters for a Particular Sensor

The Preview Engine parameter settings are sensor dependent and require tuning for a particular CCD or CMOS sensor. The following are functions with parameters that require tuning.

- Black Level Adjustment
- Noise Reduction
- White Balance
- RGB to RGB Blending

The following are functions with standard parameters that are sensor independent.

- Gamma Correction
- CFA Interpolation
- RGB-to-Y/Cb/Cr Conversion

Sensor tuning is out of scope of this application report. Please refer to [2] for more information.

4 Preview Engine Driver

4.1 Overview

The Preview Engine driver is a Linux character driver. It functions with the Linux 2.6.10 systems. The Preview Engine driver is a loadable module, which can be loaded/unloaded at run-time. The driver gets its major number from the kernel at run-time. The Preview Engine driver supports the following features:

- Bayer pattern RGB data input
- Either 8-bit or 10-bit data input
- Input from the DDR memory controller
- Both application-allocated and driver-allocated buffers, as long as they are physically contiguous, as required by hardware

Here are the features that the driver does not support:

- The Preview Engine driver does not support the *on-the-fly* mode, which gets data directly from CCDC.
- The Preview Engine driver does not support multi-passing operations for output size greater than 1280. The multi-passing operations are supported at the application level. Please refer to the application example on multi-passing.

Basically, the driver supports all available hardware features with the exception of accepting input directly from CCDC. The reason CCDC does not support this input is that it supports the multi-pass application scenario for frames that are bigger than 1280 pixels/line, such as 1080p frames which has 1920 pixels/line. Figure 8 shows the three-layer architecture of the driver.

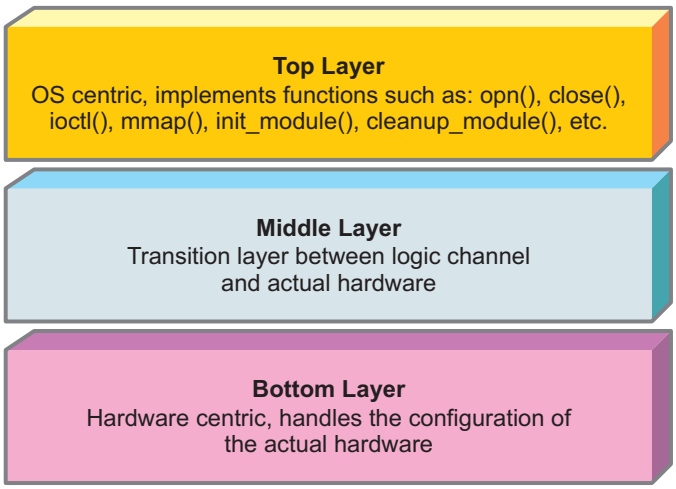

Figure 8. Three-Layer Architecture of the Preview Engine Driver

Top layer: This layer handles all operating system (OS) level driver application programming interface (API) implementations and operations associated with logic channel. This layer should be OS centric and hardware agnostic.

Bottom layer: This layer is responsible for the actual configuration of the Preview Engine hardware through writing to the Preview Engine memory mapped registers (MMRs). It should be OS agnostic and hardware centric.

Middle layer: This layer is primarily responsible for the transition between logic channel and physical hardware. It also handles the ISR.

4.2 The Preview Engine Driver API

At the top level, the Preview Engine driver implements the usual Linux driver APIs, namely open, close, mmap, and ioctl. However, it does not implement read and write to avoid the need to implement memory copy between the user and kernel space.

Please refer to the *TMS320DM644x DMSoC Video Processing Front End (VPFE) User's Guide* (SPRUE38) for the complete description of the Preview Engine driver API.

All the Preview Engine specific APIs are defined in the davinci_previewer.h header file, including ioctls and parameters.

The following are all Preview Engine specific ioctls:

- PREV_REQBUF: request buffer(s) to be allocated by driver
- PREV_QUERYBUF: query the physical memory of driver allocated buffer
- PREV_SET_PARAM: set Preview Engine parameters
- PREV_GEG_PARAM: get Preview Engine parameters
- PREV_PREVIEW: perform the preview operation
- PREV_GET_STATUS: query the status of the Preview Engine hardware
- PREV_GET_CROPSIZE: get the number of pixels and lines that are cropped in the output image given the current configuration of the Preview Engine hardware
- PREV_S_EXP: set the delays inserted between consecutive Preview Engine reads from DDR

The following is the definition for the prev_reqbufs structure passed as the parameter of the PREV_REQBUFS ioctl. This provides information on the type of buffers, size of buffers and number of buffers to be allocated.

```
typedef struct prev_reqbufs
{
        int buf_type             // buffer type: PREV_BUF_IN/PREV_BUF/OUT
        int size;                // size of the buffer in bytes
        int count                // # of buffers to be allocated
}prev_reqbufs_t;
```

The following is the definition of the Preview Engine parameter structure used in the PREV_SET_PARAM and PREV_GET_PARAM ioctls:

```
struct prev_params {
       unsigned short features;  /* Set of features enabled */
       struct prev_size_params size_params;                  /* size parameters */
       struct prev_white_balance white_balance_params;       /* white balancing
                                                                parameters */
       struct prev_black_adjst black_adjst_params;           /* black adjustment
                                                                parameters */
       struct prev_rgbblending rgbblending_params;           /* rgb blending
                                                                parameters */
       struct prev_rgb2ycbcr_coeffs rgb2ycbcr_params;        /* rgb to ycbcr
                                                                parameters */
       unsigned char sample_rate;  /* down sampling rate for averager */
       short hmf_threshold;        /* horizontal median filter threshold */
       struct prev_cfa_coeffs cfa_coeffs;           /* CFA coefficients */
       struct prev_gamma_coeffs gamma_coeffs;       /* gamma coefficients */
       struct prev_noiseflt_coeffs nf_coeffs;       /* noise filter coefficients */
       unsigned int luma_enhance[LUMA_TABLE_SIZE];  /* luma enhancement coeffs */
       struct prev_chroma_spr chroma_suppress_params;/* chroma suppression
                                                        coefficients */
       void *dark_frame_addr;                   /* dark frame address */
       unsigned short dark_frame_pitch;         /* dark frame lineoffset */
       unsigned char lens_shading_sift;         /* number of bits to be shifted
                                                   for lens shading */
       enum prev_pixorder pix_fmt;              /* output pixel format */
       int contrast;                            /* contrast */
       int brightness;                          /* brightness */
};
```

These parameters are closely mapped to the underneath Preview Engine memory mapped registers.

The following is the definition of the prev_buffer structure and the prev_convert structure used in the PREV_PREVIEW ioctl:

```
typedef struct prev_buffer
{
        int index;               // input buffer descriptor
        int buf_type;            // output buffer descriptor
        int offset;              // physical address of buffer
        int size;                // size of buffer
};

typedef struct prev_convert
{
        prev_buffer_t in_buf;    // input buffer descriptor
        prev_buffer_t out_buf;   // output buffer descriptor
};
```

The index field in the prev_buffer structure indicates which buffer to use when it is allocated by the Preview Engine driver using the PREV_REQBUFS ioctl. In this case, the offset field is ignored by the driver. If a buffer was allocated outside of the Preview Engine driver, for example, by the V4L2 capture driver, then this field should be -1 and the offset field should be the physical address for the buffer.

5 Programming Examples

Several application examples are provided with the application report showing some of the typical usage of the Preview Engine driver. These examples use the Preview Engine driver with the V4L2 video capture (CCDC) driver to pre-process and convert the captured Bayer pattern input data into Y/Cb/Cr 4:2:2 data that is ready for the next level of processing. The examples provided are for the Micron® MT9T001 CMOS image sensor.

5.1 30 fps VGA Capture and Conversion Example

This example captures input from the MT9T001 image sensor in the VGA format with a resolution of 640×480. Because the native resolution is 2048×1520, the sensor is configured to use the skipping and binning mode that results in averaging a 3×3 native pixel block for every output pixel. This configuration is supported by the V4L2 video capture driver when the video standard V4L2_STD_MT9T001_VGA_30FPS is selected by the VIDIOC_S_STD ioctl. The purpose of this example is to showcase a complete usage example of the Preview Engine driver with most of the functional blocks turned on, including:

- Black Level Adjustment
- Noise Reduction
- White Balance
- RGB-to-RGB Blending
- Gamma Correction
- CFA Interpolation
- RGB-to-YCbCr Conversion
- Brightness and Contrast Adjustment
- Luminance Enhancement and Chrominance Suppression

According to Table 1, due to pixel/line cropping at the noise filter and CFA stages of the Preview Engine, its input resolution must be 650×488 to achieve an output resolution of 640×480.

This example can be invoked by typing:./prev_vga_capture at the command line.

5.2 HD-720p 30fps Capture and Conversion Example

This example is similar to the VGA capture example with a different resolution. To display captured 1280×720 resolution video in a 720×480 resolution display, the image is panned from top to bottom and from left to right.

5.3 Multi-Pass HD-1080p 18 fps Capture and Conversion Example

This example is similar to the two capture examples except the resolution is 1920×1080. Since the maximum horizontal output size of the Preview Engine is 1280 pixels, the input image is partitioned into two slices, as shown in Figure 9.

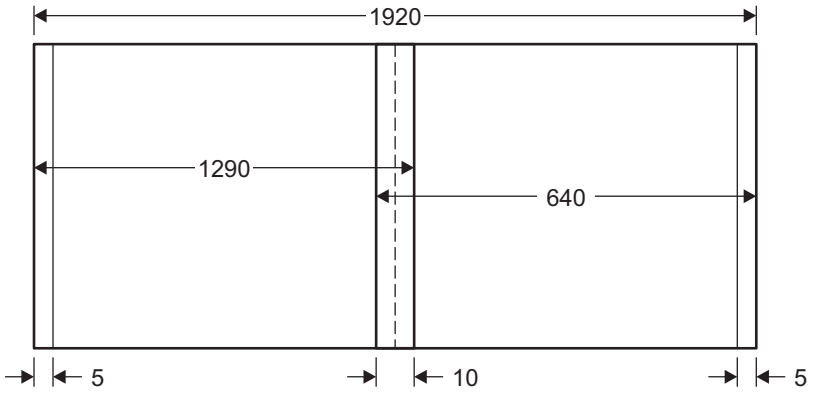

Figure 9. Slice Partitioning for 1080p Input

6 Reference

1. *TMS320DM644x DMSoC Video Processing Front End (VPFE) User's Guide*, SPRUE38
2. *A Look at the Algorithms and Tuning That Enable the DaVinci Image Pipeline*, Jianping Zhou, TIDC 2007 presentation.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265