TII DM6437 VPSS Drivers

Previewer Design Specifications

TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

1.1 Purpose & Scope

Video Processing Front End (VPFE) is a highly integrated, programmable module used for capture, preview, resize and analysis of video data.

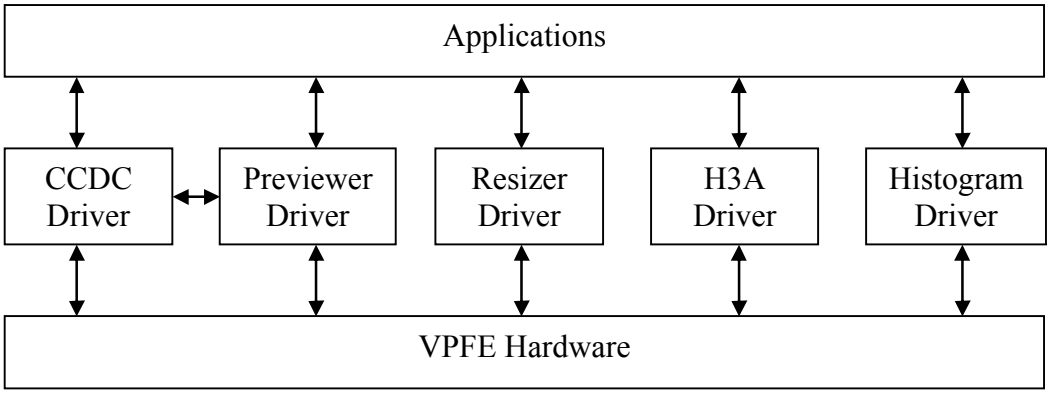

Figure 1. VPFE Driver Stack

Previewer driver facilitates an abstracted way of doing Bayer Pattern Conversion either for Raw Video available in RAM or for the Raw Data received directly from CCDC Hardware (on-the-fly). Previewed output image is always stored to the RAM.

Resizer driver facilitates an abstracted way of resizing YUV or color separate Images from RAM.

H3A driver facilitates an abstracted way of collecting statistical data from the AF & AEW Hardware for an input image, directly received from the CCDC Hardware.

Histogram driver facilitates an abstracted way of collecting histogram for an input image, directly received from the CCDC Hardware.

Scope of this document is to describe design of Previewer Driver.

1.2 Terms & Abbreviations

AF	Auto Focus
AEW	Auto Exposure & Auto White Balance
CCDC	Charged Couple Device Controller
CSL	Chip Support Library
H3A	Hardware Bases AF & AEW Modules
RAM	Random Access Memory
VPSS	Video Processing Subsystem
VPFE	Video Processing Front End

1.3 References

1.	Document 1	Peripheral Reference Guide for DM420 Subsystem Video Processing Front End Rev B06, dated SEP 29, 2005
2.	Document 2	VPFE Drivers Requirements Document.doc Draft 1.01 dated OCT 07, 2006

1.4 Overview

The Preview Engine accepts RAW image/video data in either the Bayer pattern (including movie mode patterns that are not strictly Bayer) and transforms it into YUV data in the 422 format. A number of image processing steps are necessary to achieve this conversion. The output of the Preview Engine is typically used for both video compression and display (via analog or digital interface). The Preview Engine is capable of processing up to 75 Mpixels/sec. The rest of this section describes, in detail, the functionality of each sub-block in the Preview Engine as shown in the figure below. In this block diagram, optional blocks (processing that can be bypassed) are distinguished by dashed, clear boxes and solid, shaded boxes denote mandatory blocks.

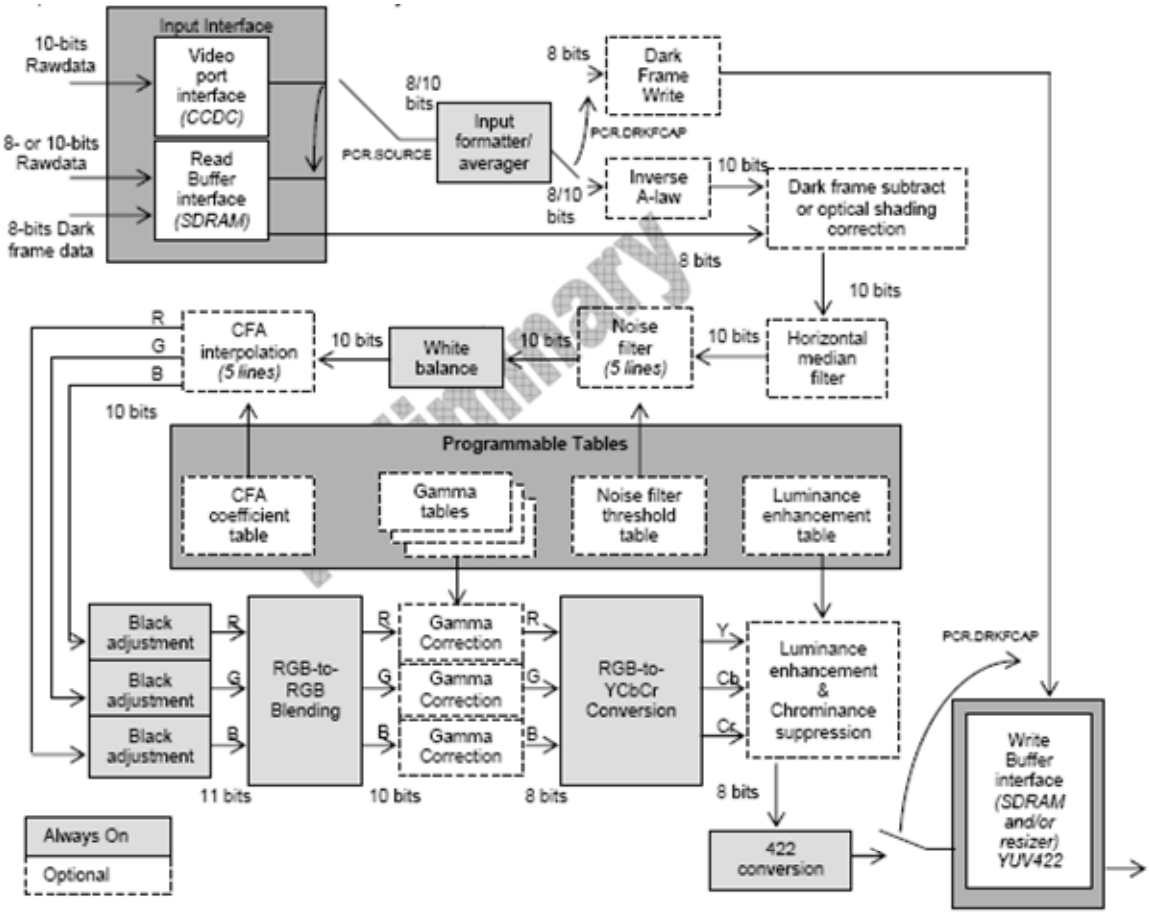

Figure 2. Block Diagram of Previewer HW

Input Interface: The Preview Engine receives RAW image/video data from either the video port interface via the CCD Controller module (which is interfaced to an external CCD/CMOS sensor) or from the read buffer interface via the SDRAM/DDRAM The input data is 10-bits wide if the source is the video port interface. When the input source is the read buffer interface, the data can either be 8-bit or 10-bits The 8-bit data can either be linear or non-linear. In addition, the Preview Engine can optionally fetch a dark frame from the SDRAM/DDRAM with each pixel being 8-bits wide

Input Formatter/Averager: The Preview Engine output is limited to 1280 pixels per horizontal line due to line memory width restrictions in the noise filter and CFA interpolation blocks. In order to support sensors that output greater than 1280 pixels per line, an 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. This must be configured to match the input pattern type. For example, a Bayer pattern has a horizontal distance of two pixels of the same color for both even and odd lines.. The valid output of the input formatter/averager is either 8- or 10-bits wide.

Dark Frame Write: The Preview Engine is capable of capturing and saving a dark frame to the SDRAM/DDRAM instead of performing the conventional processing steps This dark frame can later be subtracted from the RAW image data to eliminate the repeatable baseline noise level in the frame. Each input pixel is written out as an 8-bit value; if the input pixel value is greater than 255, it is saturated to 255. Note that if a dark pixel is greater than 255, it is more likely to be a fault pixel and can be corrected by the Fault Pixel Correction module in the CCDC. If properly corrected, the value should be less than 255 when it reaches the Preview Engine. The WSDR_ADDR and WADD_OFFSET registers should be used to indicate the output address and line offset respectively of the dark frame output in memory.

Inverse A-law: In order to save SDR/DDR capacity and bandwidth, the CCD Controller includes an option to apply 10-bit to 8-bit A-Law compression and to pack the sensor data to 1-byte per pixel. In order to process this data properly, the Inverse A-law block is provided to decompress the 8-bit non-linear data back to 10-bit linear data if enabled (PCR.INVALAW). Note that even if the Inverse A-law block is not enabled, but the input is still 8-bits (PCR.WIDTH), the data is left shifted by 2 to make it 10-bit data. If the input is 10-bits wide, no operation is performed on the data.

Darkframe Subtract or Shading Compensation The Preview Engine is capable of optionally fetching a dark frame containing 8-bit values from SDRAM/DDRAM and subtracting it, pixel-by-pixel, from the incoming input frame (PCR.DRKFEN). **Horizontal Median Filter** The Preview Engine contains a horizontal median filter that can be useful for reducing temperature induced noise effects. The horizontal median filtering operation**,** 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 specified threshold, and the sign of the differences is the same, then the average of pixel (i-X) and pixel (i+X) replaces pixel (i).

Noise Filter Following the Horizontal Median Filter, a programmable noise filter that operates on a 3x3 grid of same color pixels can be used to reduce the noise in the image/video data. This filter always operates (identifies neighborhood same-color pixels that are close in value) on nine pixels of the same color. 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 the 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.

White Balance The White Balance module has two gain adjusters, a digital gain adjuster and a white balance adjuster. In the digital gain adjuster (WB_DGAIN), the RAW data is multiplied by a fixed value gain regardless of the color pixel to be processed. In the white balance gain adjuster (WBGAIN), the RAW data is multiplied by a selected gain corresponding to the color of the processed pixel

CFA Interpolation In the generic mode, the CFA interpolation block is responsible for populating the 2 missing color pixels at a given location resulting in a 3-color RGB pixel. It does this by interpolating data from neighboring pixels of the same color.

Black Adjustment The output of the CFA interpolation is three pixels (red, blue, and green values) and this is fed as input to the black adjustment module. The black adjuster module performs the following calculation for an adjustment of each color level.
data_out = data_in + bl_offset

RGB Blending The RGB2RGB blending module has a general 3x3 square matrix and redefines the RGB data from the CFA interpolation module, which can be used as a function of a color correction. This is programmable (RGB_MAT1, RGB_MAT2, RGB_MAT3, RGB_MAT4, RGB_MAT5, RGB_OFF1, & RGB_OFF2) so that the color spectrum of the sensor can be adjusted to the human color spectrum.

Gamma Correction The gamma correction is performed on each of the R, G, and B pixels separately by indexing programmable gamma lookup tables. Each table has 1024 8-bit entries. The input data value is used to index into the table and the table content is the output.

RGB to YCbCr Conversion, Luminance Enhancement & Chrominance Suppression, Contrast & Brightness, and 4:2:2 Down Sampling & Output Clipping
RGB to YCbCr Conversion**:**
The RGB to YCbCr conversion module has a 3x3 square matrix and converts the RGB color space of the image data into the YCbCr color space. In this module, the following calculation is made using the contents of the CSC0, CSC1, CSC2, and CSC_OFFSET registers:
Non-linear Luminance Enhancement:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} CSCRY & CSCGY & CSCBY \\ CSCRCB & CSCGCB & CSCBCB \\ CSCRCR & CSCGCR & CSCBCR \end{bmatrix} \begin{bmatrix} R_{in} \\ G_{in} \\ B_{in} \end{bmatrix} + \begin{bmatrix} YOFST \\ OFSTCB \\ OFSTCR \end{bmatrix}$$

The non-linear luminance enhancement functions as an edge enhancer (crossed in the horizontal direction).
The non-linear luminance operation can be described as follows First a high-passed version of Y is computed as:
hpy(i) = y(i) – (y(i–1) + y(i+1))/2;
This value, divided by four, is fed to a lookup table which returns the offset and slope for the interpolation calculation:
offset(i) = offset_lut[hpy(i) >> 2];
slope(i) = slope_lut[hpy(i) >> 2];
interpolated(i) = offset(i) + (slope(i) * (hpy(i) & 0x3))>>2;
The interpolated output is then added to original Y to complete the luminance enhancement:
enh_y(i) = clip(y(i) + interpolated(i));
Chrominance Suppression:
Occasionally, in very bright portions of an image, only one or two of the color channels may be saturated but the remaining channel(s) may not be. This may lead to a false color effect. One common example would be the appearance of a pink color where white should be. Chrominance suppression can be used to correct this issue.
Line Width Reduction:

If the non-linear luminance enhancer or the chrominance suppression is enabled, the Preview Engine will reduce the output of this stage by 2 pixels (1 starting pixel – left edge and 1 ending pixel – right edge) in each line

Write Buffer Interface The output of the Preview Engine may be passed directly to the Previewer (PCR.RSZPORT) and/or written to SDRAM (PCR.SDRPORT). If the output is written to SDRAM, the write address (WSDR_ADDR) and line offset (WADD_OFFSET) should be on 32-byte boundaries. In this implementation output will be stored in SDRAM only.

2 Requirements

The DM6437 Previewer driver Functional requirements are as follows:

- Driver shall support input from CCDC and DDR
- Driver shall support output to DDR
- Driver shall support dark frame capturing and dark frame subtraction

2.1 Assumptions

None

2.2 Constraints

Driver will not support output to Resizer.

3 Design Description

This section gives detail on the overall architecture of TI DM6437 Previewer device driver. This includes the static view explaining the functional decomposition and dynamic view explaining the deployment scenario of the Previewer driver.

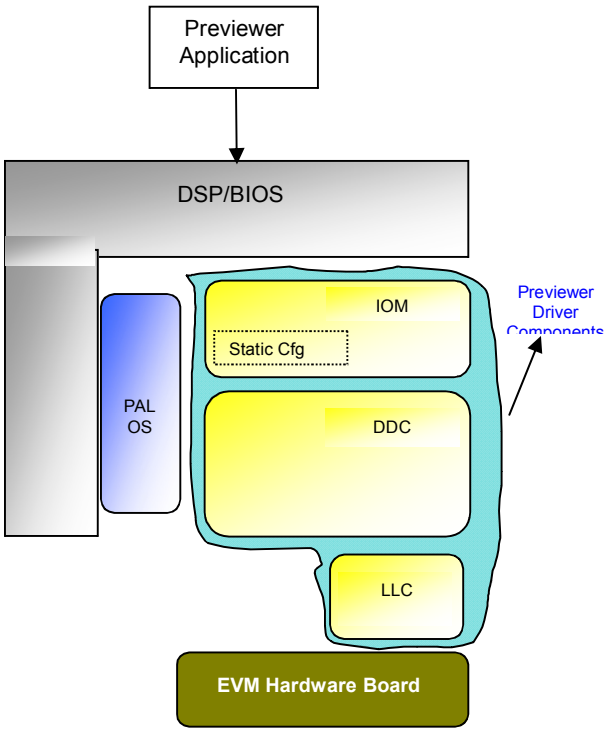

Figure 3. Overall Design of Previewer Driver

3.1 Component Interaction

This Section demonstrates component level interactions – static and dynamic. These diagrams help in presenting the data/ message exchanges, events etc. in the component/ system under design.

3.1.1 Static view

The central portion (IOM-DDC-LLC) shown in above constitutes mainline previewer driver component. The surrounding module PAL-OS constitute the supporting system components that facilitates the interfaces between the OS and above mentioned driver components. These modules do not specifically deal with previewer but assist the driver by providing OS abstraction.

H/W Device Specific Layer (LLC)

Please refer section 1.2.1 of DM6437_BIOS_PSP_User_Guide.pdf for Diagrammatic Explanation

Device Driver Core functionality (DDC)

Please refer section 1.2.1 of DM6437_BIOS_PSP_User_Guide.pdf for Diagrammatic Explanation

OS Specific Device Driver Adaptation (IOM)

Please refer section 1.2.1 of DM6437_BIOS_PSP_User_Guide.pdf for Diagrammatic Explanation

Platform Abstraction Layer for OS services (PALOS)

Please refer section 1.2.1 of DM6437_BIOS_PSP_User_Guide.pdf for Diagrammatic Explanation

3.1.2 Dynamic view

This sub-section describes the interaction between different layers and functionalities of the Previewer driver. Figure below represents detailed design for Previewer driver.

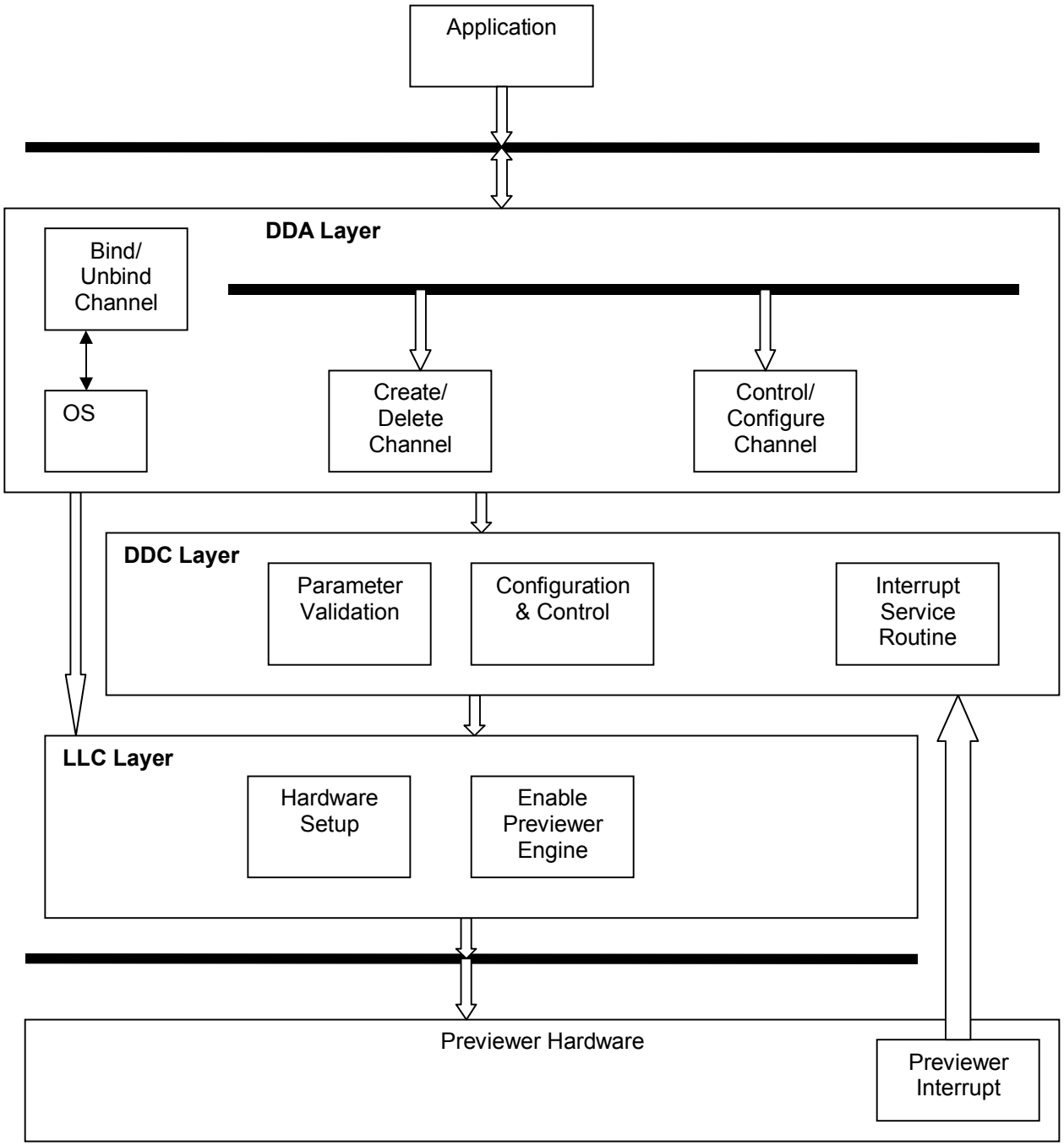

Figure 4. Detailed Design Diagram for Previewer

Various functionalities of different layers in Previewer driver are as below:

DDA (IOM)layer

- Bind/Unbind
- Create/Delete
- Various controls to configure HW

DDC Layer

- Multiple Application Handling
- Parameters Validation
- Interrupt Service Routine

LLC Layer

- Hardware setup
- Enabling previewer engine

3.1.2.1 Driver Creation/deletion

Driver Creation

The sequence diagram below depicts the creation phase of the BIOS Previewer driver. User is expected to invoke PREV_mdBindDev (),in the application startup phase.

The PREV_mdBindDev () performs register overlaying of the device driver. It registers the interrupt handler of the driver. It attaches the DDC functions for use later during actual initialization of each device instance.

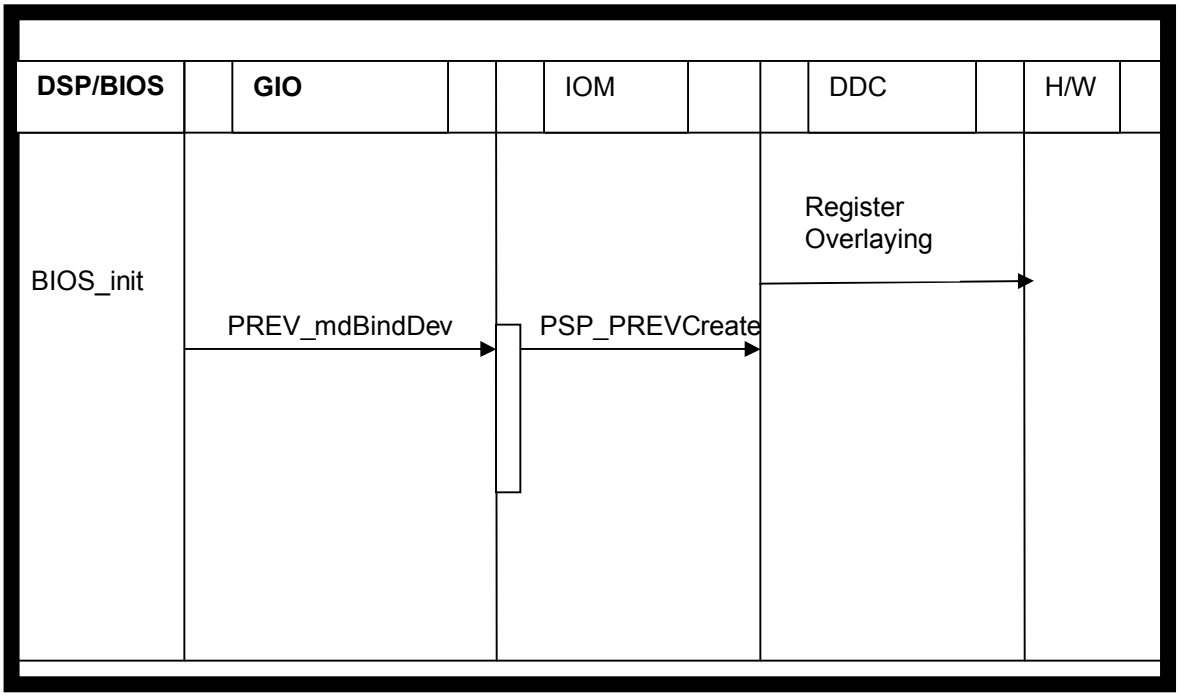

Figure 5. Driver Creation overview

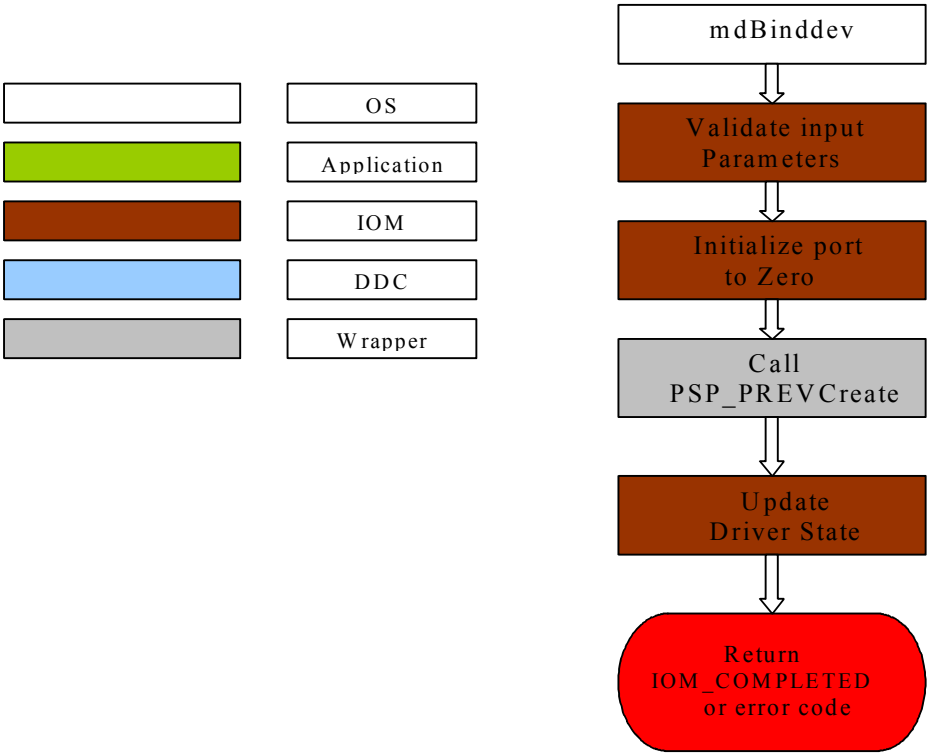

Figure 6. Driver Creation detail flow diagram-1

PSP_PREVCreate

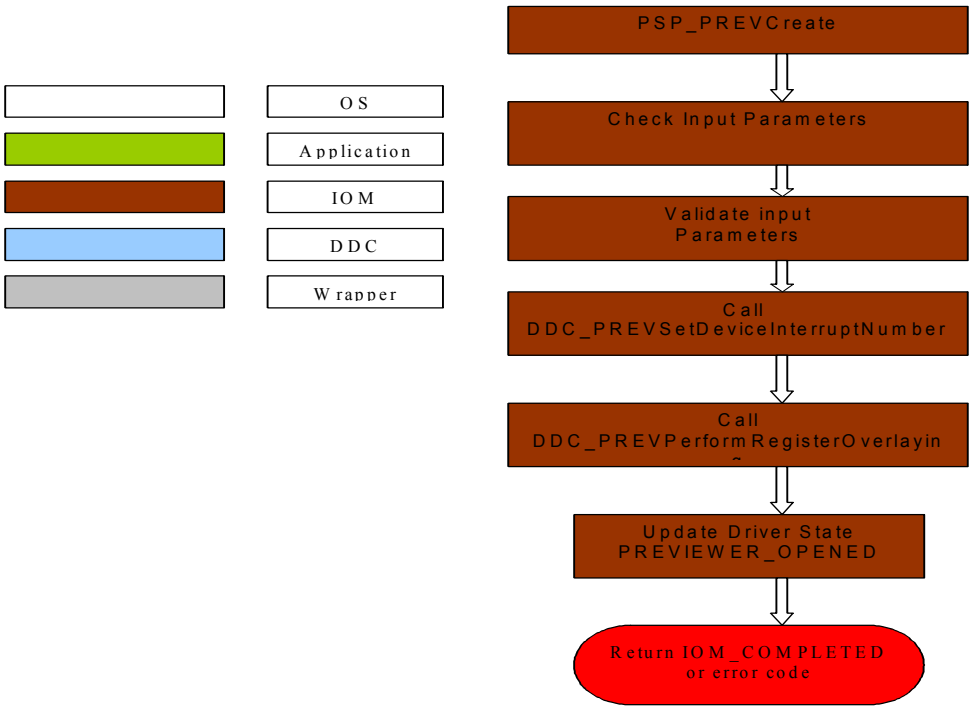

Figure 7. Driver Creation Detail flow diagram -2

Driver Deletion

The driver de-initialize and delete functions de-initialize the Previewer DDC and deallocate OS resources allocated through PREV_mdBindDev ().

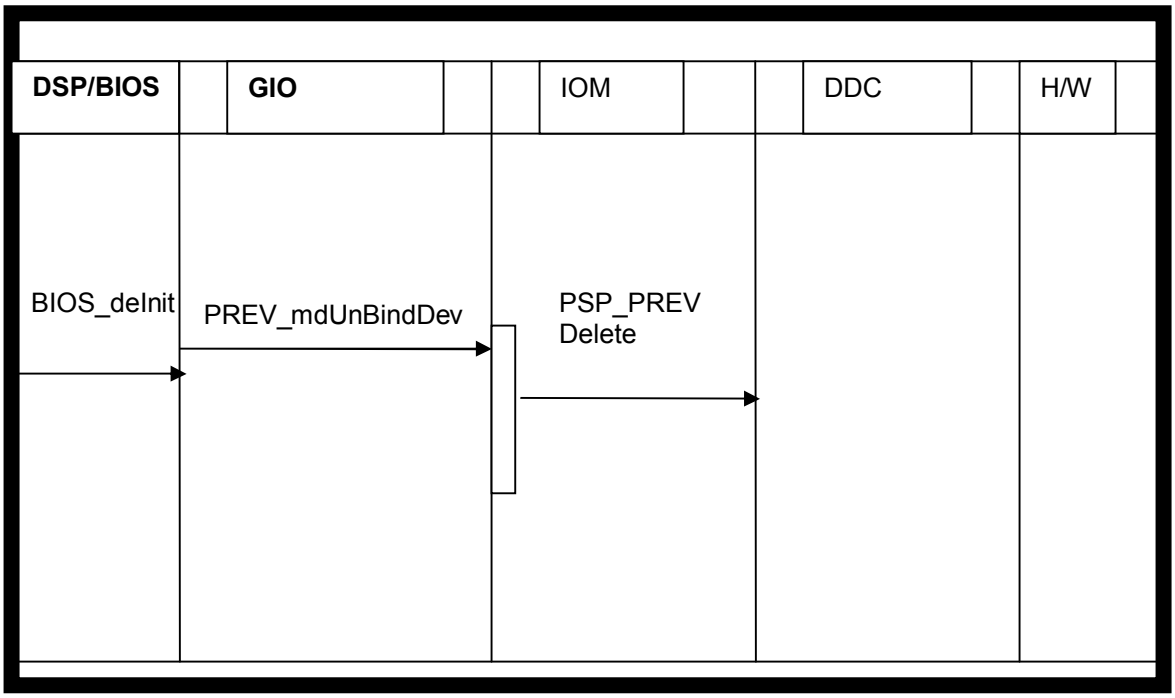

Figure 8. Driver deletion overview

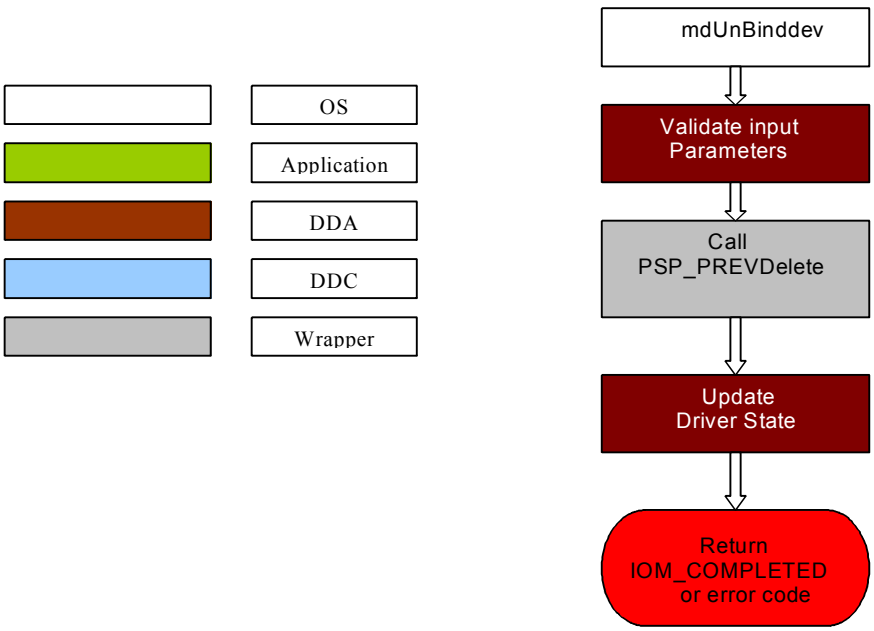

Figure 9. Driver Deletion detail Flow diagram -1

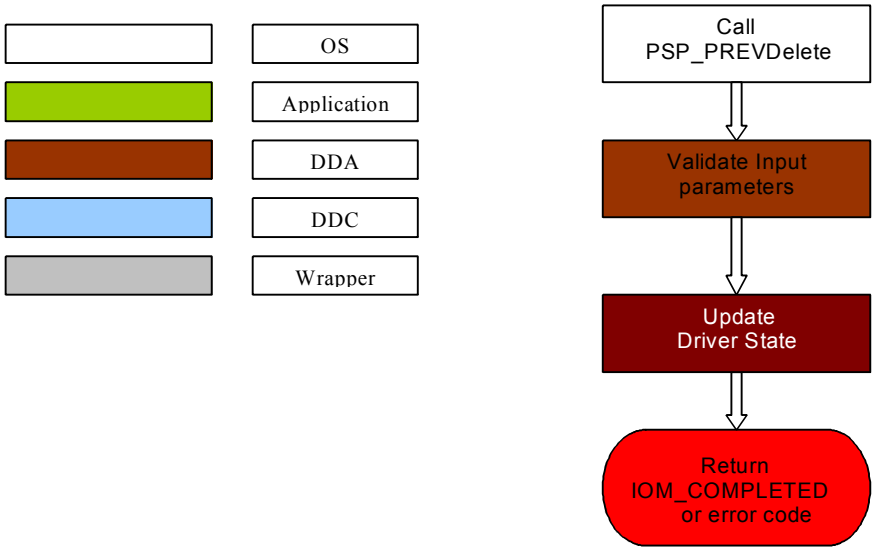

Figure 10. Driver deletion detail flow diagram -2

3.1.2.2 Driver Open/Close

Driver Open

When the application calls the GIO_create() function, internally PREV_mdCreateChan () will be called, driver creates logical channel and initializes channel parameters.

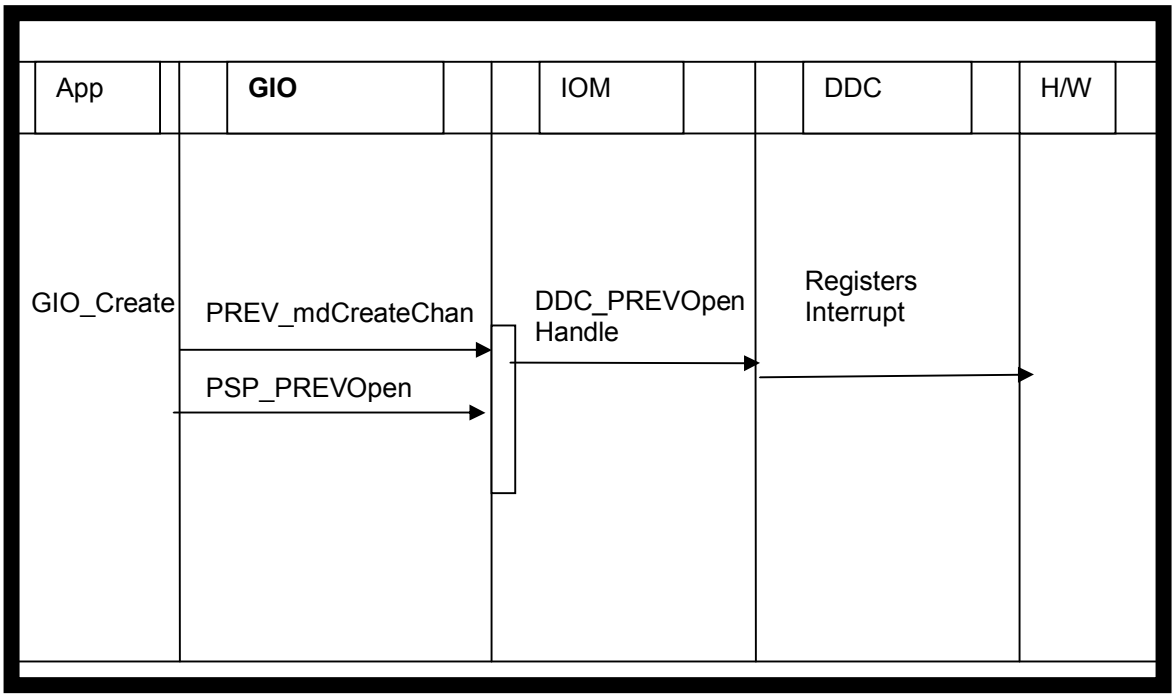

Figure 11. Driver create channel overview

PREV_mdCreateChan

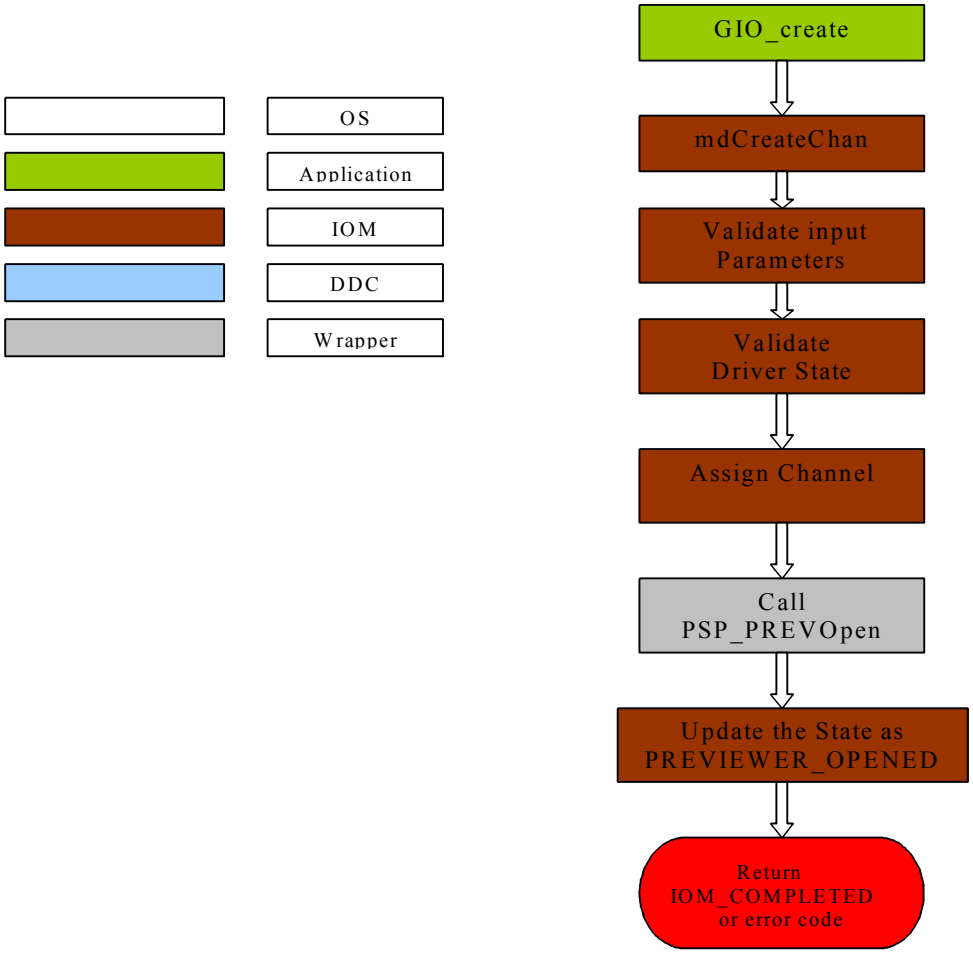

Figure 12. Driver Open Detail flow diagram -1

PSP_PREVOpen

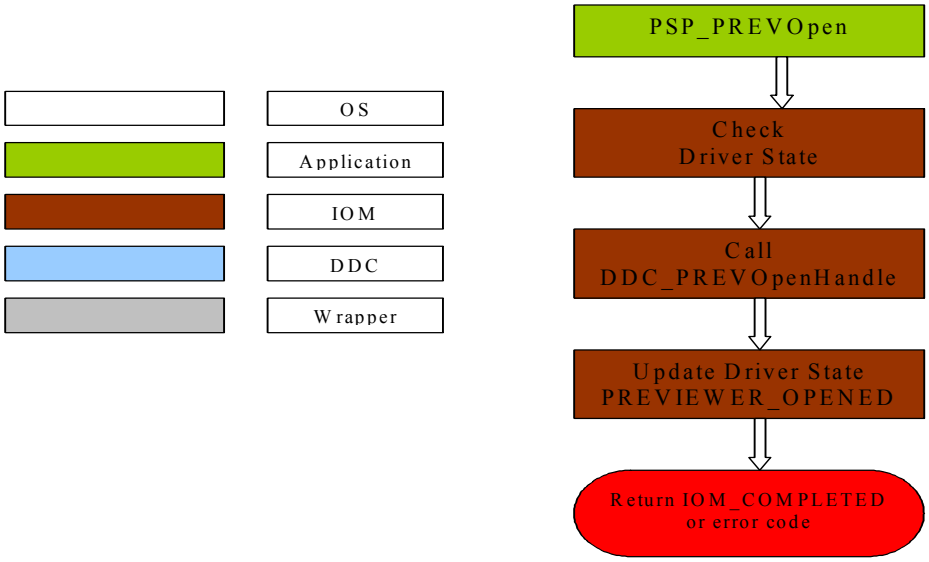

Figure 13. Driver Open detail flow diagram -2

DDC_PREVOpenHandle

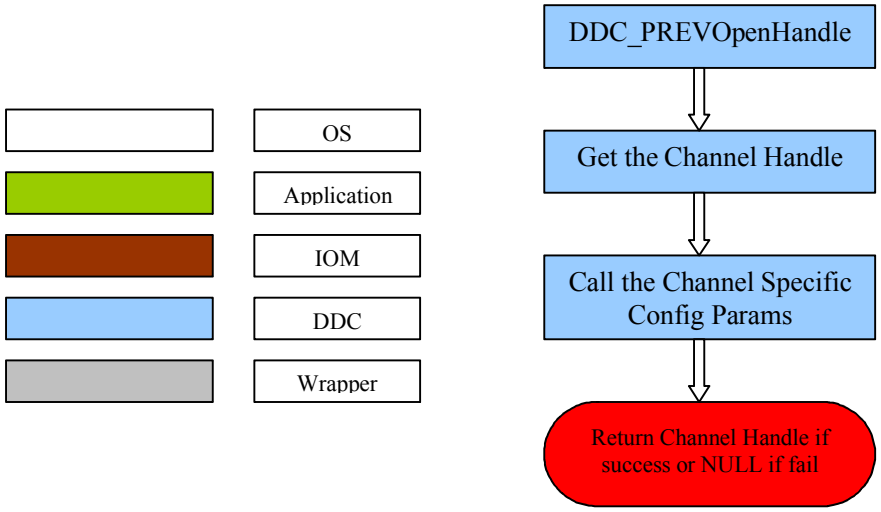

Figure 14. Driver Open detail flow diagram -3

Driver Close

When the application calls the GIO_delete() function, internally PREV_mdDeleteChan() will be called to delete driver logical channel.

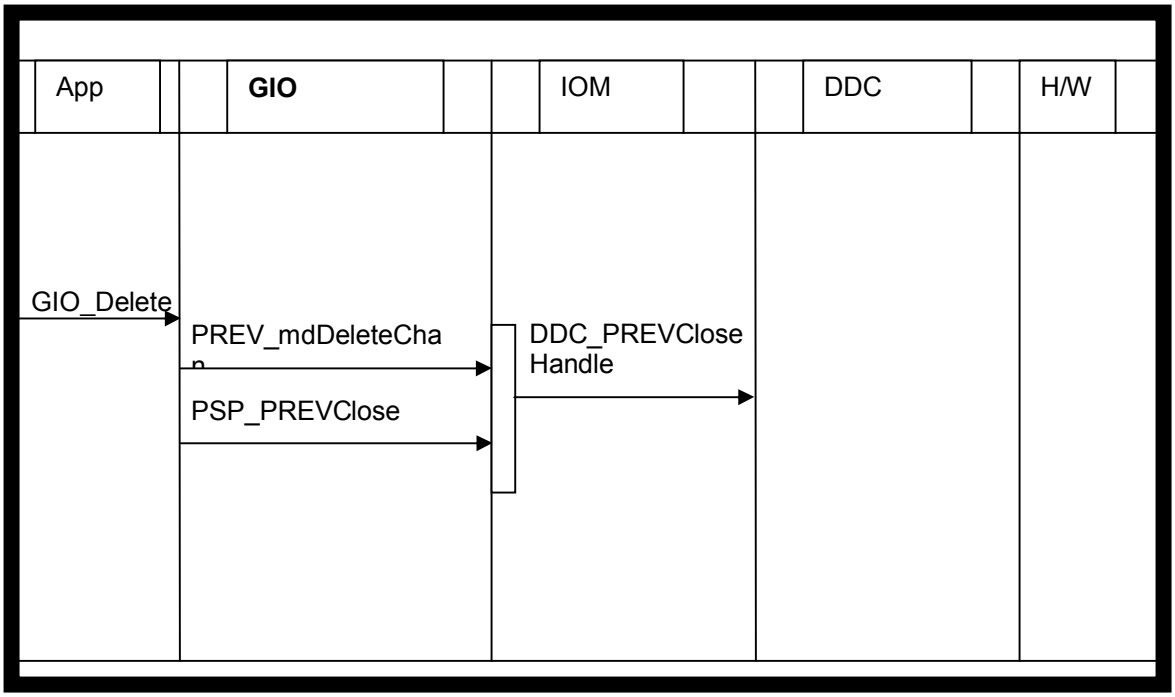

Figure 15. Driver channel close overview

PREV_mdDeleteChan

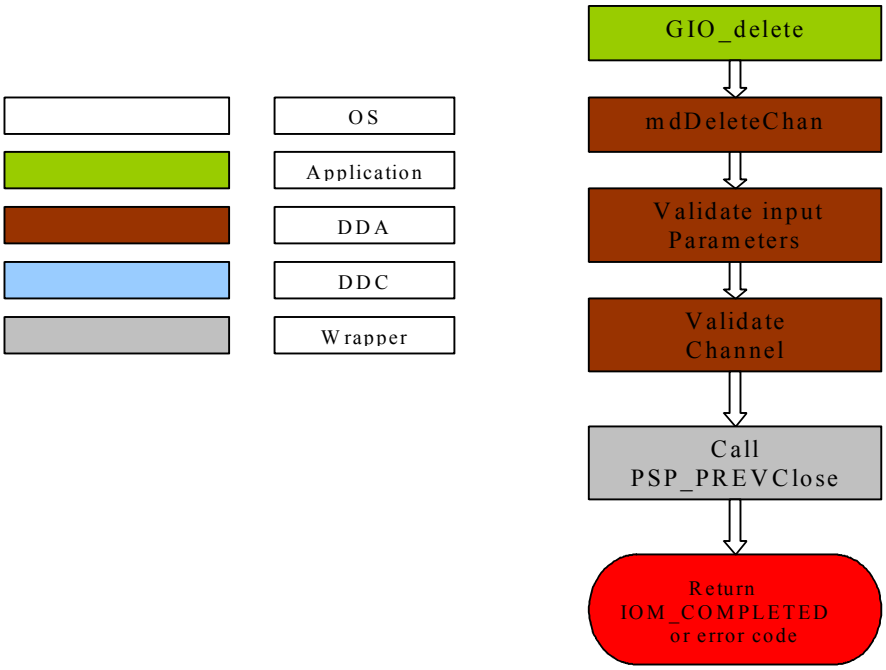

Figure 16. Driver close channel detail flow diagram -1

PSP_PREVClose

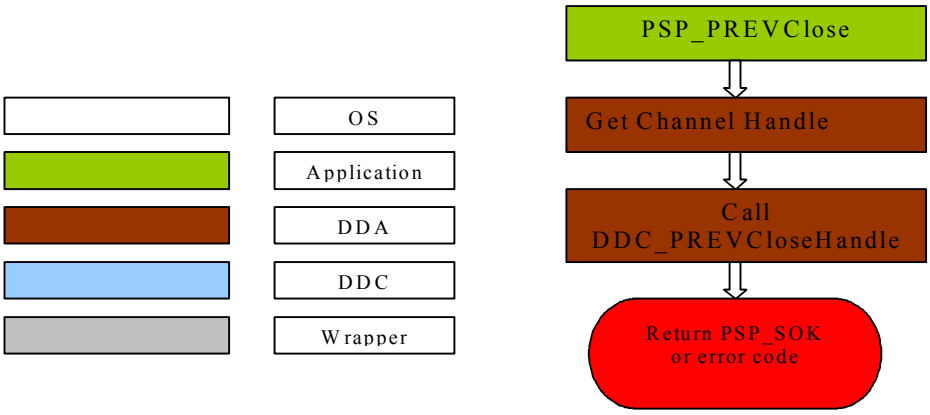

Figure 17. Driver close channel detail flow diagram -2

DDC_PREVCloseHandle

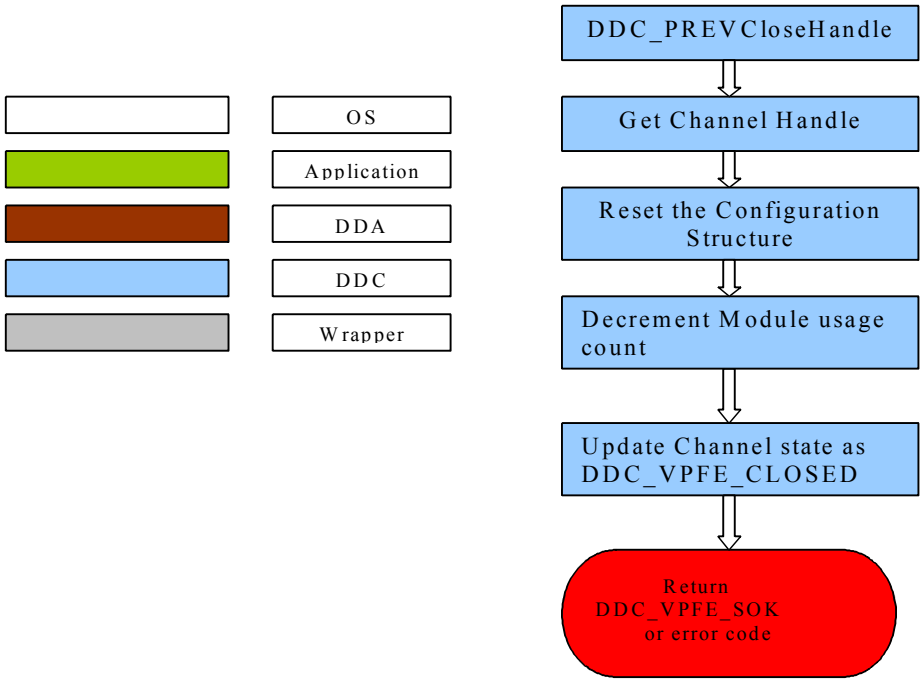

Figure 18. Driver close Channel detail flow diagram -3

3.1.2.3 Various Controls

When the application calls GIO_control internally PREV_mdControlChan() is called to set/get common configuration parameters and perform resizing of the input image..

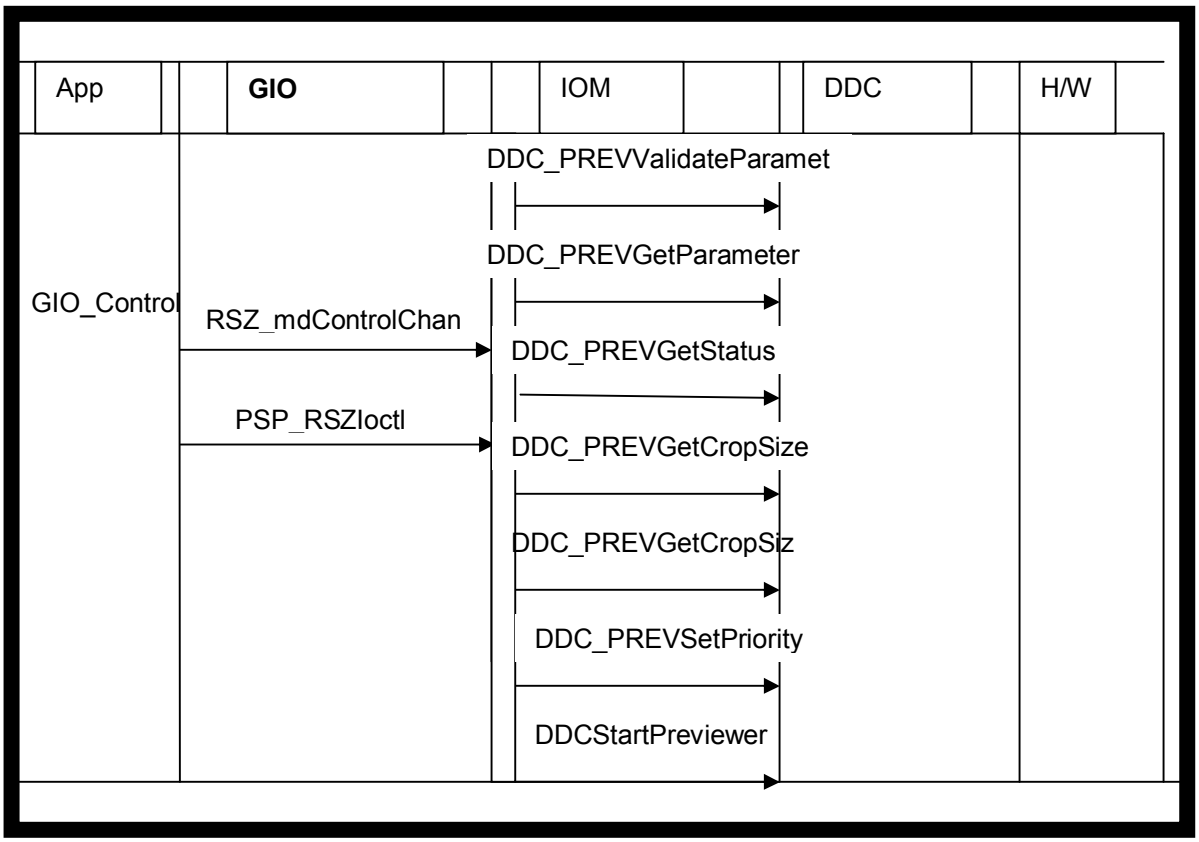

Figure 19. Control Command overview

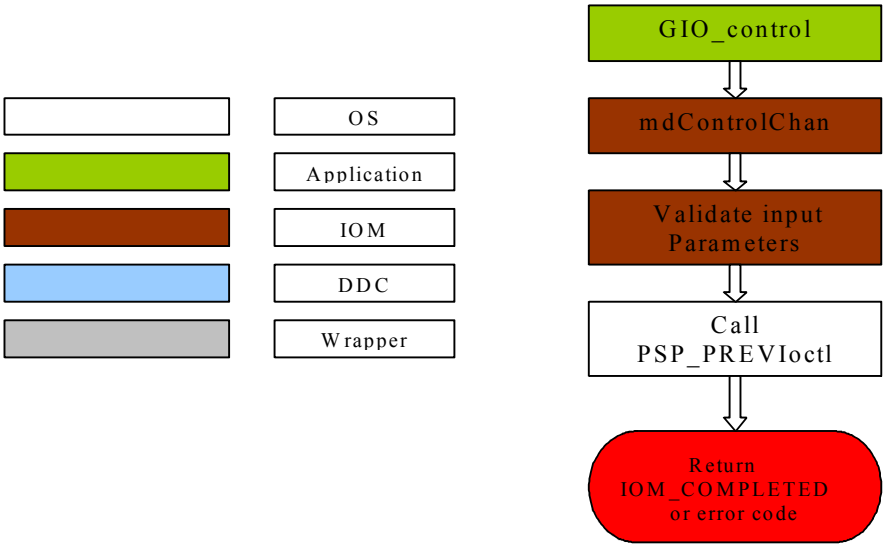

Figure 20. Control Command detail flow diagram -1

3.1.2.4 Interrupt Service Routine

Previewer Driver Interrupt Handles will be attached with the Previewer Hardware Interrupt of DM6437. Previewer driver will implement interrupt dispatcher which will handle interrupts generated for the completion of previewing activity. On receipt of interrupt, this interrupt dispatcher will read the status register to verify the source of interrupt and post the Mutex. This Mutex, which is a member of the Global Device structure, is used to signal the preview completion to the preview function of the configuration & control module.

3.1.2.5 Parameters Validation

This sub module will check the validity of every parameters .It will return an error if the parameter is not valid. This will be tightly coupled with the ioctl function for PSP_PREV_IOCTL_SET_PARAMS command.

4 Low Level Definitions

This section describes low level details of Previewer Driver.

4.1 Constants & Enumerations

CONSTANTS:

4.1.1 Feature List

It describes the Various Feature List of the previewer.

Definition

```
#define PSP_PREVIEWER_INVERSE_ALAW
#define PSP_PREVIEWER_HMF
#define PSP_PREVIEWER_NOISE_FILTER
#define PSP_PREVIEWER_CFA
#define PSP_PREVIEWER_GAMMA
#define PSP_PREVIEWER_LUMA_ENHANCE
#define PSP_PREVIEWER_CHROMA_SUPPRESS
#define PSP_PREVIEWER_DARK_FRAME_SUBTRACT
#define PSP_PREVIEWER_LENS_SHADING
```

Comments

None

Constraints

None

4.1.2 Down Sampling Rate

It describes the Down sampling rate of the previewer.

Definition

```
#define PSP_PREVIEWER_DOWN_SAMPLE_RATE1
#define PSP_PREVIEWER_DOWN_SAMPLE_RATE2
#define PSP_PREVIEWER_DOWN_SAMPLE_RATE4
#define PSP_PREVIEWER_DOWN_SAMPLE_RATE8
```

Comments

None

Constraints

None

4.1.3 Table Size of Modules

Describe the table size require for the different modules of the previewer

Definition

```
#define PSP_PREVIEWER_LUMA_TABLE_SIZE            128
#define PSP_PREVIEWER_GAMMA_TABLE_SIZE           1024
#define PSP_PREVIEWER_CFA_COEFF_TABLE_SIZE       576
#define PSP_PREVIEWER_NOISE_FILTER_TABLE_SIZE    256
```

Comments

None

Constraints

None

4.1.4 White Balance Gain

Describe the number of coefficient for the white balance gain

Definition

```
#define PSP_PREVIEWER_WB_GAIN_MAX     4
```

Comments

None

Constraints

None

4.1.5 RGB2RGB Blending

Describes the matrix size of the RGB2RGB Blending

Definition

```
#define PSP_PREVIEWER_RGB_MAX         3
```

Comments

None

Constraints

None

4.1.6 Image Width

This describes the max image width in pixels

Definition

```
#define PSP_PREVIEWER_MAX_IMAGE_WIDTH   1280
```

Comments

None

Constraints

None

4.1.7 Previewer Source

This describes the previewer source

Definition

```
#define PSP_PREVIEWER_CHANNEL_CCDC
#define PSP_PREVIEWER_CHANNEL_SDRAM
```

Comments

None

Constraints

None

4.1.8 Previewer Width

This describes the previewer with 8bit or 10bit

Definition

```
#define PSP_PREVIEWER_INWIDTH_8BIT
#define PSP_PREVIEWER_INWIDTH_10BIT
```

Comments

None

Constraints

None

4.1.9 Device instance Status

This describes the device instance status of the previewer.

Definition

```
#define DDA_PREVIEWER_DEVICE_CREATED
#define DDA_PREVIEWER_ DEVICE_DELETED
```

Comments

None

Constraints

None

4.1.10 Device Channel Status

This describes the device channel status of the previewer.

Definition

```
#define PSP_PREVIEWER_CHANNEL_FREE
#define PSP_PREVIEWER_CHANNEL_BUSY
```

Comments

None

Constraints

None

4.1.11 Dark Frame Status

This describes the dark frame status of the previewer.

Definition

```
#define PSP_PREVIEWER_DARK_FRAME_FAILED
#define PSP_PREVIEWER_DARK_FRAME_WORKING
```

Comments

None

Constraints

None

4.1.12 Channel Configure State

This describes state of channel configuration

Definition

```
#define DDC_PREVIEWER_CHANNEL_NOT_CONFIGURED
#define DDC_PREVIEWER_CHANNEL_CONFIGURED
```

Comments

None

Constraints

None

4.1.13 Control Command

This enum describes various control commands

Definition

```
typedef enum _PSP_previewerControlCmd
{
    PSP_PREVIEWER_IOCTL_SET_PARAMS = IOM_USER,
    PSP_PREVIEWER_IOCTL_GET_PARAMS,
    PSP_PREVIEWER_IOCTL_GET_STATUS,
    PSP_PREVIEWER_IOCTL_GET_DARK_FRAME_STATUS,
    PSP_PREVIEWER_IOCTL_PREVIEW,
    PSP_PREVIEWER_IOCTL_GET_CROPSIZE,
```

```
    PSP_PREVIEWER_IOCTL_SET_EXP,
    PSP_PREVIEWER_GET_INFO_FOR_CCDC,
    PSP_PREVIEWER_SET_DARK_FRAME_CAPTURE,
    PSP_PREVIEWER_GET_DARK_FRAME_CAPTURE
} PSP_previewerControlCmd;
```

Comments

None

Constraints

None

4.1.14 Output Pixel Order

This enum describes Data type for Output pixel Order

Definition

```
typedef enum _PSP_previewerOutPixelOrder
{
    PSP_PREVIEWER_PIXELORDER_YCBYCR,
    PSP_PREVIEWER_PIXELORDER_YCRYCB,
    PSP_PREVIEWER_PIXELORDER_CBYCRY,
    PSP_PREVIEWER_PIXELORDER_CRYCBY
} PSP_previewerOutPixelOrder;
```

Comments

None

Constraints

None

4.1.15 Interrupt mapping needed status

This macro defines interrupt mapping is needed or not.

Definition

```
#define DDC_PREVIEWER_INTERRUPT_MAPPING_NOT_NEEDED
#define DDC_PREVIEWER_INTERRUPT_MAPPING_NEEDED
```

Comments

None

Constraints

None

4.1.16 DDC error codes

This macros describes error code used at DDC layer.

Definition

```
#define  DDC_PREV_SOK
#define  DDC_PREV_E_FAIL
#define  DDC_PREV_INVALID_STATE
#define  DDC_PREV_INVALID_PARAM
#define  DDC_PREV_MEMORY_ERROR
#define  DDC_PREV_RESOURCE_ERROR
#define  DDC_PREV_OVERFLOW_ERROR
#define  DDC_PREV_NOT_SUPPORTED
```

Comments

None

Constraints

None

4.2 Data Structures

4.2.1 Previewer Channel source Structure

This structure is used as a member element of _PSP_previewerChannelCreateMode structure.

Definition

```
typedef struct _PSP_previewerChannelSource
{
    Uint8 source;
}PSP_previewerChannelSource;
```

Fields

source — PSP_PREVIEWER_CHANNEL_CCDC or PSP_PREVIEWER_CHANNEL_SDRAM

Comments

None

Constraints

None

4.2.2 Previewer Create Channel Structure

This structure is used at the time of channel creation.

Definition

```
typedef struct _PSP_previewerChannelCreateMode
{
    PSP_previewerChannelSource chanSource;
    Int32 segId;
}PSP_previewerChannelCreateMode;
```

Fields

source — channel source

segId — segment id passed by application, will be used to allocate memory

Comments

None

Constraints

None

4.2.3 Previewer Get Info For CCDC

This structure is used to get an info of previewer, required by CCDC.

Definition

```
typedef struct PSP_previewerGetInfoForCCDC
{
```

```
    Uint8 source;
    Uint8 intNo;
} PSP_previewerGetInfoForCCDC;
```

Fields

Source	inetrrupt source
intNo	interrupt number

Comments

None

Constraints

None

4.2.4 Previewer Size Parameter

This structure is used to configure Previewer size parameters

Definition

```
typedef struct _PSP_previewerSize
{
    Uint16 inPitch;
    Uint16 outPitch;
    Uint8 sph;
    Uint8 eph;
    Uint8 slv;
    Uint8 elv;
    Uint8 pixelSize;
}PSP_previewerSize;
```

Fields

inPitch	Represents line offset of input image - used when source is SDRAM
outPitch	Represents line offset of output image
sph	For specifying start pixel horizontal
eph	For specifying end pixel horizontal
slv	For specifying start line vertical
elv	For specifying end line vertical
pixelSize	pixel size of the image in terms of bits - used when source is SDRAM

Comments

All parameters should be configured before previewing.

Constraints

None

4.2.5 Previewer White Balance

This structure is used for white balancing parameters

Definition

```
typedef struct _PSP_previewerWhiteBalance
{
    Uint8 position[PSP_PREVIEWER_WB_GAIN_MAX][ PSP_PREVIEWER_WB_GAIN_MAX];
    Uint16 wbDgain;
    Uint8 wbGain[PSP_PREVIEWER_WB_GAIN_MAX];
} PSP_previewerWhiteBalance;
```

Fields

position	Represents 16 position, out of 4 values
wbDgain	Represents white balance common(digital) gain
wbGain[PSP_PREVIEWER_WB_GAIN_MAX]	individual color gains

Comments

None

Constraints

None

4.2.6 Previewer Black Adjustment

This structure is used for black adjustment for colors.

Definition

```
typedef struct _PSP_previewerBlackAdj
{
    Int8 redAdj;
    Int8 greenAdj;
    Int8 blueAdj;
} PSP_previewerBlackAdj;
```

Fields

redAdj	Represents black adjustment offset for red color
greenAdj	Represents black adjustment offset for green color
blueAdj	Represents black adjustment offset for blue color

Comments

None

Constraints

None

4.2.7 Previewer RGB2RBG Blending

This structure is used for RBG2RGB blending.

Definition

```
typedef struct _PSP_previewerRgbBlending
{
    Uint16 blending[PSP_PREVIEWER_RGB_MAX][PSP_PREVIEWER_RGB_MAX];
    Int16 offset[PSP_PREVIEWER_RGB_MAX];
} PSP_previewerRgbBlending;
```

Fields

blending[PSP_PREVIEWER_RGB_MAX][PSP_PREVIEWER_RGB_MAX]	Represents color correlation 3x3 matrix
offset[PSP_PREVIEWER_RGB_MAX];	Represents color correlation offsets

Comments

None

Constraints

None

4.2.8 Previewer Rgb2Ycbcr

This structure is used for Rgb2Ycbcr parameters.

Definition

```
typedef struct _PSP_previewerRgb2ycbcrCoeffs
{
    Uint16 coeff[PSP_PREVIEWER_RGB_MAX][PSP_PREVIEWER_RGB_MAX];
    Uint16 yOffset;
    Int8 cbOffset;
    Int8 crOffset;
}PSP_previewerRgb2ycbcrCoeffs;
```

Fields

coeff[PSP_PREVIEWER_RGB_MAX][PSP_PREVIEWER_RGB_MAX];	Represents color conversion gains in 3x3 matrix
yOffset	Represents y color conversion offsets
cbOffset	Represents Cb color conversion offsets
crOffset	Represents Cr color conversion offsets

Comments

None

Constraints

None

4.2.9 Previewer CFA coefficients

This structure is for CFA coefficient.

Definition

```
typedef struct _PSP_previewerCfaCoeffs
{
    Uint8 hThreshold;
    Uint8 vThreshold;
    Int32 coeffs[PSP_PREVIEWER_CFA_COEFF_TABLE_SIZE];
}PSP_previewerCfaCoeffs;
```

Fields

hThreshold	Represents horizontal threshold
vThreshold	Represents vertical threshold
coeffs[PSP_PREVIEWER_CFA_COEFF_TABLE_SIZE]	Represents cfa coefficients

Comments

None

Constraints

None

4.2.10 Previewer Gamma Coeffs

This structure is for previewer Gamma coeffs.

Definition

```
typedef struct _PSP_previewerGammaCoeffs
{
    Uint8 red[PSP_PREVIEWER_GAMMA_TABLE_SIZE];
    Uint8 green[PSP_PREVIEWER_GAMMA_TABLE_SIZE];
    Uint8 blue[PSP_PREVIEWER_GAMMA_TABLE_SIZE];
} PSP_previewerGammaCoeffs;
```

Fields

red[PSP_PREVIEWER_GAMMA_TABLE_SIZE]	Represents table of gamma correction values for red
green[PSP_PREVIEWER_GAMMA_TABLE_SIZE]	Represents table of gamma correction values for green
blue[PSP_PREVIEWER_GAMMA_TABLE_SIZE]	Represents table of gamma correction values for blue

Comments

None

Constraints

None

4.2.11 Previewer Noise Filter Coeffs

This structure is for noise filter coeffs.

Definition

```
typedef struct _PSP_previewerNoiseFilterCoeffs
{
    Uint8 noise[PSP_PREVIEWER_NOISE_FILTER_TABLE_SIZE]
    Uint8 strength;
}PSP_previewerNoiseFilterCoeffs;
```

Fields

noise[PSP_PREVIEWER_NOISE_FILTER_TABLE_SIZE]	Represents noise filter table
strength	Represents to find out weighted average

Comments

None

Constraints

None

4.2.12 Previewer Chroma Suppression

This structure is used for chroma suppression.

Definition

```
typedef struct _PSP_previewerChromaSuppression
{
    Uint8 hpfy;
    Int8 threshold;
    Uint8 gain;
} PSP_previewerChromaSuppression;
```

Fields

hpfy	Represents whether to use high passed version of Y or normal Y
threshold	Represents threshold for chroma suppress
gain	Represents chroma suppression gain

Comments

None

Constraints

None

4.2.13 Previewer Structure

This structure is used for all parameters configuration of Previewer

Definition

```
typedef struct _PSP_previewerParams
{
    Uint32 features;
    PSP_previewerSize sizeParam;
    PSP_previewerWhiteBalance whiteBalanceParam;
    PSP_previewerBlackAdj  blackAdjParam;
    PSP_previewerRgbBlending  rgbBlendingParam;
    PSP_previewerRgb2ycbcrCoeffs rgb2ycbcrParam;
    PSP_previewerCfaCoeffs  cfaCoeffsParam;
    PSP_previewerGammaCoeffs  gammaCoeffsParam;
    PSP_previewerNoiseFilterCoeffs  noiseFilterCoeffsParam;
    PSP_previewerChromaSuppression  chromaSuppressionParam;
    PSP_previewerOutPixelOrder  outPixelOrderParam;
    Uint32  lumaEnhance[PSP_PREVIEWER_LUMA_TABLE_SIZE];
    Ptr darkFrameAddr;
    Uint16  darkFrameOffset;
    Int16  hmfThreshold;
    Int8  contrast;
    Int8  brightness;
    Uint8  downSampleRate;
    Uint8  lensShadingShift;
}PSP_previewerParams;
```

Fields

Features	Set of features enabled
sizeParam	size parameters
whiteBalanceParam	white balancing parameters
blackAdjParam	black adjustment parameters
rgbBlendingParam	rgb blending parameters
rgb2ycbcrParam	rgb to ycbcr parameters
cfaCoeffsParam	CFA coefficients
gammaCoeffsParam	gamma coefficients
noiseFilterCoeffsParam	noise filter coefficients
chromaSuppressionParam	chroma suppression coefficients
outPixelOrderParam	output pixel format
lumaEnhance[PSP_PREVIEWER_LUMA_TABLE_SIZE]	luma enhancement coeffs
darkFrameAddr	dark frame address
darkFrameOffset	Dark frame offset
hmfThreshold	hmfThreshold
contrast	constrast
brightness	brightness
downSampleRate	down sampling rate for averager

lensShadingShift	number of bits to be shifted for lens shading

Comments
None

4.2.14 Previewer Status

This structure is used to know the previewer status.

Definition

```
typedef struct _PSP_previewerStatus
{
    Uint8 channelStatus;
}PSP_previewerStatus;
```

Fields

channelStatus	Represents Status of previewer

Comments
none

Constraints
None

4.2.15 Previewer Darkframe Status

This structure is used to know the dark frame status.

Definition

```
typedef struct _PSP_previewerDarkFrameStatus
{
    Uint8 status;
}PSP_previewerDarkFrameStatus;
```

Fields

status	Represents Status of Dark frame fail

Comments
none

Constraints
None

4.2.16 Previewer preview

This structure is used to pass parameters while triggering previewer for preview operation.

Definition

```
typedef struct _PSP_preview
{
    Ptr inBuf;
    Ptr outBuf;
```

```
    int inBufSize;
    int outBufSize;
}PSP_preview;
```

Fields

inBuf	Represents address of the input buffer
outBuf	Represents address of the output buffer
inBufSize	Represents input buffer size
outBufSize	Represents output buffer size

Comments

none

Constraints

None

4.2.17 Previewer Crope Size

This structure is used to receive a crop size while calling get_crop_size ioctl.

Definition

```
typedef struct _PSP_previewerCropSize
{
    Int32 hCrop;
    Int32 vCrop;
}PSP_previewerCropSize;
```

Fields

hCrop	Represents number of pixels per line cropped in output image
vCrop	Represents number of lines cropped in output image

Comments

None

4.2.18 Previewer Read Request Expand

This structure is used to pass a preview read request expand.

Definition

```
typedef struct _PSP_previewerReadReqExp
{
    Int16 prevExp;
}PSP_previewerReadReqExp;
```

Fields

prevExp	a preview read request expand

Comments

none

Constraints

None

4.2.19 Previewer Dark Frame Capture

This structure is used to get/set dark frame capture feature..

Definition

```
typedef struct _PSP_previewerDarkFrameCapture
{
    Uint8 darkFrameState;
    Uint16 outPitch;
} PSP_previewerDarkFrameCapture;
```

darkFrameState	Indicates whether to enable dark frame capture or not. Value can be PSP_PREVIEWER_DARK_FRAME_CAPTURE_DISABLE or PSP_PREVIEWER_DARK_FRAME_CAPTURE_ENABLE
outPitch;	offset in dark image for each row. Value is only relevant when flag is PSP_PREVIEWER_DARK_FRAME_CAPTURE_ENABLE

Comments

none

Constraints

None

4.2.20 IOM Port structure

This structure contains information related to per device instance. For each device instance one instance will be there of this structure. It contains information related to device in reference with IOM layer.

Definition

```
typedef struct _DDA_prevPortObject
{
    struct _DDA_prevChannelObject* channelHandle;
    Uint32 state;
    PSP_Handle ddcDeviceHandle;
} DDA_prevPortObject, *DDA_prevPortHandle;
```

Fields

channelHandle	Represents single channel instance, for this port.

state	Represents state of device instance – it can be DDA_PREVIEWER_DEVICE_CREATED orDDA_PREVIEWER_DEVICE_DELETED.
ddcDeviceHandle	Pointer to device specific DDC structure, which will be used to invoke proper DDC object. At the time of device binding it will be initialized.

Comments

One object of this structure will be taken at DDA layer. The name of that object is prevPortObj.

Constraints

None

4.2.21 IOM Channel structure

This structure contains per channel information. For each channel instance one instance will be there of this structure. It contains information related to channel in reference with IOM layer.

Definition

```
typedef struct _DDA_prevChannelObject
{
    DDA_prevPortHandle portHandle;
    PSP_Handle ddcChanHandle;
} DDA_prevChannelObject, *DDA_prevChannelHandle;
```

Fields

portHandle	Represents handle to port structure.
ddcChanHandle	Pointer to channel specific DDC structure, which will be used to invoke proper DDC object. At the time of channel creation it will be initialized.

Comments

None

Constraints

None

4.2.22 Previewer Device structure

This structure contains information related to per device instance for DDC layer. For each device instance one instance will be there of this structure. It contains information related to device in reference with DDC layer.

Definition

```
typedef struct _DDC_prevDeviceObject
{
    Uint8 intNum;
```

```
    Ptr regs;
    PAL_OsSemHandle semIsr;
}DDC_prevDeviceObject, *DDC_prevDeviceHandle;
```

Fields

intNum	Interrupt number on which interrupt for previewer is captured.
regs	Handle for previewer registers
semIsr	This is the semaphore that will be posted at the end of ISR and in preview IOCTL, driver will pend on this semaphore.

Comments

One object of this structure will be taken at DDC layer. The name of that object is prevDeviceObj.

Constraints

None

4.2.23 Previewer Channel structure

This structure contains per channel information. For each channel instance one instance will be there of this structure. It contains information related to channel in reference with DDC layer.

Definition

```
typedef struct _DDC_prevChannelObject
{
    PSP_previewerChannelCreateMode channelMode;
    PSP_previewerParams *DDC_prevParamHandle;
    PAL_OsSemHandle  channelSem;
    PSP_previewerDarkFrameCapture DFCState;
    Int8 configState;
} DDC_prevChannelObject, *DDC_prevChannelHandle;
```

Fields

channelMode	Indicates channel source is SDRAM or CCDC.
DDC_prevParamHandle	Handle to previewer parameters structure.
channelSem	If preview ioctl is running and request to change config params comes(like set_params) or ioctl for changing config params is running and request for preview ioctl is coming at that time , on this semaphore, second request will pend.
configState	When channel will be created at that time,channel will not be configured, this state will represent that and before calling preview ioctl, appli. needs to configure channel.otherwise error will return. Values can be DDC_PREVIEWER_CHANNEL_NOT_CONFIGURED or DDC_PREVIEWER_CHANNEL_CONFIGURED

DDC_previewHandle	Handle to preview request parameters structure.
DFCState	dark frame capture state structure. Contains information that dark frame capture is enabled or not, and if enabled, what is the outpitch for it.

Comments

None

Constraints

None

Constraints

None

4.3 API Definition

4.3.1 GIO_CREATE

Syntax

GIO_Handle GIO_create(String name, int mode, int* status, Ptr chanParams, GIO_Attrs * attrs);

Arguments

IN String name

The name argument is the name specified for the device when it was created in the configuration or at runtime. It is used to find a matching name in the device table. The name generally will be "/previewer".

Int Mode

The mode argument specifies the mode in which the device is to be opened. This may be IOM_INPUT, IOM_OUTPUT or IOM_INOUT.In case of previewer, IOM_INOUT will be used

Ptr chanParams

The chanParams parameter is a pointer that may be used to pass device or domain-specific arguments to the mini-driver. The contents at the specified address are interpreted by the mini-driver in a device-specific manner. It is optional.

In case of previewer object of PSP_previewerChannelCreateMode structure is passed.which specified what will be the source of channel.(DDR or CCDC.)

Return Value

GIO_Handle — Handle to an instance of the device if device is successfully opened. It returns NULL if the device could not be opened.

Comments

It will open a logical channel. Multiple channel will not be supported by the previewer driver.

The PSP_previewerChannelCreateMode structure is as shown below:

/*previewer channel structure - filled at the time of channel creation*/

```
typedef struct _PSP_previewerChannelCreateMode
{
    Uint8 source;  /* SDRAM or CCDC */
}PSP_previewerChannelCreateMode;
```

The GIO_Attrs structure is as shown below:

```
typedef struct GIO_Attrs
{
   Int nPackets;       /* number of I/O packets */
   Uns timeout;        /* for blocking calls    */
} GIO_Attrs;
```

Default for nPackets is 2 & for timeout is SYS_FOREVER if attrs is NULL.

Constraints

This function can be called only after the device has been loaded and initialized.

4.3.2 GIO_DELETE

Syntax

int GIO_delete(GIO_Handle gioChan);

Arguments

IN	GIO_Handle	gioChan

The gioChan parameter is the handle returned by GIO_create.

Return Value

Int	This function returns IOM_COMPLETED if the channel is successfully closed. If an error occurs, the device returns a negative value.

Comments

An application calls GIO_delete to close a communication channel associated with gioChan.

Constraints

This function can be called only after the device has been loaded and initialized. The handle supplied should have been obtained with a prior call to GIO_create.

4.3.3 GIO_CONTROL

Syntax

int GIO_control(GIO_Handle gioChan, int cmd, int args);

Arguments

IN	GIO_Handle	gioChan

The gioChan parameter is the handle returned by GIO_create.

int	cmd

Specified mini-driver command to perform functionality.

IN OUT	int		args

The args parameter points to a data structure defined by the device to allow control information to be passed between the device and the application.

Return Value

Int	IOM_COMPLETED on success and negative value if error.

Comments

An application calls GIO_control to configure or perform control functionality on the communication channel.

Constraints

- This function can be called only after the device has been loaded and initialized. The handle supplied should have been obtained with a prior call to GIO_create.
- GIO_control cannot be called from a SWI or HWI unless the underlying mini-driver is a non-blocking driver and the GIO Manager properties are set to use non-blocking synchronization methods.

4.4 DDA Layer Functions

4.4.1 PREV_mdBind

Function	PREV_mdBindDev()
Function Prototype	Int PREV_mdBindDev (Ptr *devp, Int devid, Ptr devParams)
Input Parameters	devid – device id to distinguish instance of devices(incase of multiple instances).in previewer case only one instance will be there. devParams – would be the H/W configuration information pointer variable. It will be NULL in case of previewer.
Output Parameters	devp – void pointer to be updated once instance is created Int returning the IOM Status
Description	This function would call PSP_prevCreate function to create device instance at DDA layer.
Preconditions	None
Design	Logic in steps. 1) call PSP_prevCreate function

4.4.2 PREV_mdCreateChan

Function	PREV_mdCreateChan
Function Prototype	Int PREV_mdCreateChan (Ptr *chanp, Ptr devp, String name, Int mode, Ptr chanParams, IOM_TiomCallback cbFxn, Ptr cbArg)
Input Parameters	chanp – void pointer to be updated after the channel has been created. devp – pointer to the port structure name – name of the driver.Name will differentiate between the drivers. mode – mode of the driver. This may be IOM_INPUT, IOM_OUTPUT or IOM_INOUT.In case of previewer, IOM_INOUT will be used. chanParams – additional parameters needed to initialize the channel cbFxn – callback function to the GIO

	cbArg – callback arguments
Output Parameters	Int according to the IOM errors
Description	This function would call PSP_prevOpen to open a channel at DDA layer.
Preconditions	None
Design	Logic in steps 1) Call PSP_prevOpen function.

4.4.3 PREV_mdControlChan

Function	PREV_mdControlChan
Function Prototype	Int PREV_mdControlChan(Ptr chanp, Uns cmd, Ptr arg)
Input Parameters	chanp – pointer to the channel object. cmd – control command to be executed arg – argument needed to execute the command
Output Parameters	Int according to the IOM error codes
Description	This function would call PSP_prevIoctl to execute different IOCTLs.
Preconditions	None
Design	Logic in steps 1) Call a function PSP_prevIoctl to Execute the IOCTL command.

4.4.4 PREV_mdDeleteChan

Function	PREV_mdDeleteChan
Function Prototype	int PREV_mdDeleteChan (Ptr chanp)
Input Parameters	Chanp – pointer to chan object
Output Parameters	Int according to the IOM error codes
Description	This function would call PSP_prevClose

	function to close channel at DDA layer.
Preconditions	None
Design	Logic in steps 1) Call PSP_prevClose Function

4.4.5 PREV_mdUnbindDev

Function	PREV_mdUnbindDev
Function Prototype	Int PREV_mdUnbindDev(Ptr devp)
Input Parameters	Devp – Handle to PREV device
Output Parameters	Int according to the IOM error code
Description	This function would call PSP_prevDelete function to delete a device instance at DDA layer.
Preconditions	None
Design	Logic in steps. 1) Call the PSP_prevDelete function.

4.4.6 PSP_prevCreate

Function	PSP_prevCreate
Function Prototype	PSP_Result PSP_prevCreate(PSP_Handle* devHandle)
Input Parameters	NULL.
Output Parameters	Int according to the PSP errors DevHandle - void pointer to be updated by port structure once device instance is created
Description	This function initializes the device.
Preconditions	The status in IOM port structure should be DDA_PREVIEWER_ DEVICE_DELETED . The driver should be opening for the first time at the OS initialization time.
Design	Logic in steps

	1) Check the Input Parameters 2) Check the state of the device instance 3) Initialize IOM port structure 4) Call previewer DDC_prevDeviceCreate 5) Update the parameters of the driver port structure 6) Assign IOM port structure instance to devHandle.

4.4.7 PSP_prevOpen

Function	PSP_prevOpen
Function Prototype	PSP_Result PSP_Handle PSP_prevOpen (PSP_Handle *chanHandle, PSP_Handle devHandle, PSP_previewerChannelCreateMode chanParams)
Input Parameters	devHandle - device instance port structure. chanParams – additional parameters needed to initialize the channel here it will be passed to DDC_prevOpenHandle.
Output Parameters	Int according to the PSP errors chanHandle – pointer to channel object that is being created will be assigned to this pointer
Description	This function creates a channel.
Preconditions	Device must be in DDA_PREVIEWER_DEVICE_CREATED state and channel should not be created yet.
Design	Logic in steps 1) Check the Input Parameters 2) Check the State of the Device 3) Check that the channel is not in use 4) Allocate memory for the channel 5) Point portHandle to portObject. 6) Call DDC_prevOpenHandle Function. 7) Update the state of the Channel. 8) Call DDA_prevSetInterrupt to bind H/W interrupt to ISR routine, if source for previewer is DDR

4.4.8 PSP_prevIoctl

Function	PSP_prevIoctl

Function Prototype	PSP_Result PSP_prevIoctl(PSP_Handle chanHandle,PSP_previewerControlCmd cmd,Ptr cmdArg)
Input Parameters	chanHandle – Handle of the Channel Cmd – Submit command to be passed CmdArg – Pointer to parameter structure passed as an argument for cmd
Output Parameters	Int according to the PSP errors
Description	It acts as wrapper, which calls DDC_prevIoctl function to execute this IOCTL.
Preconditions	Device must be in DDA_PREVIEWER_DEVICE_CREATED state and channel should be in created state.
Design	Logic in steps 1) Check Channel is Closed or not. 2) Read the ioctl command, and call DDC_prevIoctl.

4.4.9 PSP_prevClose

Function	PSP_prevClose
Function Prototype	PSP_Result PSP_prevClose(PSP_Handle chanHandle)
Input Parameters	chanHandle – Handle to close the channel
Output Parameters	Int according to the PSP errors
Description	This function would remove the channel instance.
Preconditions	Device must be in DDA_PREVIEWER_DEVICE_CREATED state and channel should be in created state.
Design	Logic in steps 1) Check Channel is Closed or not. 2) Call DDC_prevCloseHandle Function of the corresponding channel. 3) Update portHandle structure 4) Call DDA_prevUnsetInterrupt to unbind H/W interrupt to ISR routine,

	if source for previewer is DDR 5) Free memory allocated for channel object.

4.4.10 PSP_prevDelete

Function	PSP_prevDelete
Function Prototype	PSP_Result PSP_prevDelete(PSP_Handle devHandle)
Input Parameters	devp – handle to port device structure
Output Parameters	Int according to the PSP errors
Description	This function initializes the device.
Preconditions	Device must be in DDA_PREVIEWER_DEVICE_CREATED state and channel should not be in created state.
Design	Logic in steps 1) Check the Device handle 2) Check the state of the driver 3) Call DDC_prevDeviceDelete to delete device instance at DDC layer 4) Update the state of the driver port structure – make ddcDeviceHandle to NULL.

4.4.11 PSP_prevGetPSPHandle

Function	PSP_prevGetPSPHandle
Function Prototype	PSP_Result PSP_prevGetPSPHandle(PSP_Handle *chanHandle)
Input Parameters	NULL
Output Parameters	Int according to the PSP errors chanHandle – Handle to the channel
Description	It returns PSP handle of channel object..
Preconditions	Device must be in DDA_PREVIEWER_DEVICE_CREATED state and channel should not be in created state.
Design	Logic in steps

	1) Return the PSP handle.

4.4.12 DDA_prevSetInterrupt

Function	DDA_prevSetInterrupt
Function Prototype	void DDA_prevSetInterrupt(Uint8 intNum, ECM_Fxn isrRoutine)
Input Parameters	intNum - interrupt number on which previewer gives interrupt isrRoutine – ISR function which will be mapped
Output Parameters	None
Description	This function maps isrRoutine to previewer interrupt.
Preconditions	None
Design	Logic in steps 2) call PAL_osProtectEntry to disable interrupt 3) plug the isrRoutine to proper interrupt. 4) Call PAL_osProtectExit to enable interrupt. 5) Call HWI_disable to disable interrupts. 6) Call HWI_dispatchplug to plug interrupts. 7) Call HWI_restore to enable interrupts.

4.4.13 DDA_prevUnsetInterrupt

Function	DDA_prevUnsetInterrupt
Function Prototype	void DDA_prevUnsetInterrupt(Uint8 intNum)
Input Parameters	intNum - interrupt number on which previewer gives interrupt
Output Parameters	None

Description	This function un maps isrRoutine to previewer interrupt.
Preconditions	None
Design	Logic in steps 1) Disable previewer interrupt

4.5 DDC Layer Functions

4.5.1 DDC_prevDeviceCreate

Function	DDC_prevDeviceCreate
Function Prototype	DDC_prevResult DDC_prevDeviceCreate(PSP_Handle *ddcDeviceHandle)
Input Parameters	NULL
Output Parameters	Return DDC error status ddcDeviceHandle – handle for this device instance at DDC layer will be initialized here.
Description	It will open device at DDC layer.
Preconditions	None
Design	Logic in steps 1) Create semaphore for ISR. 2) Take object of previewer device structure. Assign it to the ddcDeviceHandle. 3) Call DDC_prevPerformRegOverlaying

4.5.2 DDC_prevDeviceDelete

Function	DDC_prevDeviceDelete
Function Prototype	DDC_prevResult DDC_prevDeviceDelete(PSP_Handle ddcDeviceHandle)
Input Parameters	ddcDeviceHandle - handle for this device instance at DDC layer
Output Parameters	Return DDC error status
Description	It will close device at DDC layer.
Preconditions	None
Design	Logic in steps 1) delete ISR semaphore. 2) Reset device handle parameters.

4.5.3 DDC_prevPerformRegOverlaying

Function	DDC_prevPerformRegisterOverlaying
Function Prototype	DDC_prevResult DDC_prevPerformRegisterOverlaying()
Input Parameters	NULL
Output Parameters	Return DDC error status
Description	It will Perform register overlaying.
Preconditions	None
Design	Logic in steps 1) Get the previewer device handle. 2) Assign the previewer base address to the regs member of the device handle.

4.5.4 DDC_prevOpenHandle

Function	DDC_prevOpenHandle
Function Prototype	DDC_prevResult DDC_prevOpenHandle(PSP_previewerChannelCreateMode chanParams, PSP_Handle *ddcChanHandle, Uint32 *interruptMapRequired, Uint8 *interruptMapInfo, ddcISRType *isrFunc)
Input Parameters	chanParams – pointer to `PSP_previewerChannelCreateMode structure`
Output Parameters	Return DDC error status ddcChanHandle – handle to channel structure at DDC level. interruptMapRequired - indicates interrupt mapping is required or not. interruptMapInfo - indicates on which event mapping is needed. isrFunc - pointer to ISR function
Description	It will return the Channel handle and update the state of previewer channel.
Preconditions	None
Design	Logic in steps 1) Allocate memory for DDC channel object.

	2) Take an object of DDC_prevChannelObject structure and keep configState to NOT_CONFIGURED state. 3) channelSem will be created. 4) Fill interrupt related information in interruptMapRequired, isrFunc and interruptMapInfo. 5) Return the channel handle

4.5.5 DDC_prevCloseHandle

Function	DDC_prevCloseHandle
Function Prototype	DDC_prevResult DDC_prevCloseHandle(PSP_Handle ddcChanHandle)
Input Parameters	prevChannelHandle – handle to channel structure at DDC level.
Output Parameters	Return DDC error status
Description	It will close the channel at DDC level and return error code
Preconditions	None
Design	Logic in steps 1) Take channel semaphore. 2) channelSem will be deleted. 3) Free memory for channel object. 4) Update the port structure

4.5.6 DDC_prevIoctl

Function	DDC_prevIoctl
Function Prototype	DDC_prevResult DDC_prevValidateParameters(PSP_Handle ddcChannelHandle,PSP_previewerControlCmd cmd,Ptr cmdArg)
Input Parameters	ddcChannelHandle – DDC level channel object Cmd – Submit command to be passed CmdArg – Pointer to parameter structure passed as an argument for cmd
Output	Return DDC error status

Parameters	
Description	It will call respective function to execute proper IOCTL.
Preconditions	None
Design	Logic in steps: Validate cmdtype. I Call corresponding function as per type of IOCTL If cmd type is PSP_PREVIEWER_IOCTL_SET_PARAMS, pend on channel semaphore;call DDC_prevSetParameters;post channel semaphore If cmd type is PSP_PREVIEWER_IOCTL_GET_PARAMS,call DDC_prevGetParameters. If cmd type is PSP_PREVIEWER_IOCTL_GET_STATUS,call DDC_prevGetStatus. If cmd type is PSP_PREVIEWER_IOCTL_GET_DARK_FRAME_STATUS, pend on channel semaphore;call DDC_prevGetDarkFrameStatus;post channel semaphore If cmd type is PSP_PREVIEWER_IOCTL_PREVIEW, And already one preview request is already running then return an error, Else pend on channel semaphore;call DDC_prevPreview;post channel semaphore; If cmd type is PSP_PREVIEWER_IOCTL_GET_CROPSIZE,call DDC_prevgetCropSize.

4.5.7 DDC_prevValidateParameters

Function	DDC_prevValidateParameters
Function Prototype	DDC_prevResult DDC_prevValidateParameters(DDC_prevChannelHandle ddcChannelHandle, PSP_previewerParams * DDC_prevParamHandle)
Input Parameters	ddcChannelHandle – DDC level channel object

	DDC_prevParamHandle - Pointer to PSP_prevparams according to which values will be filled in previewer registers
Output Parameters	Return DDC error status
Description	It will Validate the previewer channel parameters passed by the user.
Preconditions	None
Design	Check parameters validity of DDC_prevParamHandle object.

4.5.8 DDC_prevSetParameters

Function	DDC_prevSetParameters
Function Prototype	DDC_prevResult DDC_prevSetParameters(DDC_prevChannel Handle ddcChannelHandle, PSP_previewerParams * DDC_prevParamHandle)
Input Parameters	ddcChannelHandle – DDC level channel object DDC_prevParamHandle - Pointer to PSP_prevparams according to which values will be filled in previewer registers
Output Parameters	Return DDC error status
Description	It will get the channel specific parameters.
Preconditions	None
Design	Logic in steps Check that previewer is disabled. Validate values of passed parameter structure,by calling DDC_prevValidateParameters() take values from the structure passed and filled these values in channel structure. Call LLC_preHardwareSetup

4.5.9 DDC_prevGetParameters

Function	DDC_prevGetParameters

Function Prototype	DDC_prevResult DDC_prevGetParameters(DDC_prevChannelHandle ddcChannelHandle, PSP_previewerParams * DDC_prevParamHandle)
Input Parameters	NULL
Output Parameters	Return DDC error status DDC_prevParamHandle - Pointer to PSP_prevparams which will be filled.
Description	It will get the channel specific parameters.
Preconditions	Channel handle should be in configures state.
Design	Logic in steps 1) Get the previewer parameters from the channel specific structure and fill these values in passed structure.

4.5.10 DDC_prevGetCropSize

Function	DDC_prevGetCropSize
Function Prototype	DDC_prevResult DDC_prevGetCropSize(DDC_prevChannelHandle ddcChannelHandle,PSP_previewerCropSize *)
Input Parameters	ddcChannelHandle – DDC level channel object
Output Parameters	Return DDC error status Pointer to PSP_previewerCropSize
Description	It will get the information regarding number of horizontal and vertical cropping pixels.
Preconditions	Channel handle should be in configures state.
Design	Logic in steps 1) Get the number of horizontal and vertical pixels to be cropped by using channel object.

4.5.11 DDC_prevGetStatus

Function	DDC_prevGetStatus
Function Prototype	DDC_prevResult DDC_prevGetStatus(DDC_prevChannelHandle ddcChannelHandle, PSP_previewerStatus *prevStatus)
Input Parameters	ddcChannelHandle – DDC level channel object
Output Parameters	Return DDC error status prevStatus – hardware status will be stored in this field.
Description	It will get the information regarding the Status of the hardware and channel.
Preconditions	Channel handle should be in configures state.
Design	Logic in steps 1) validate input parameters and check channel is configured or not. Get the status of channel by calling LLC_prevChannelStatus.

4.5.12 DDC_prevGetDarkFrameStatus

Function	DDC_prevGetDarkFrameStatus
Function Prototype	DDC_prevResult DDC_prevGetDarkFrameStatus(DDC_prevChannelHandle ddcChannelHandle, PSP_previewerDarkFrameStatus *prevDarkFrameStatus)
Input Parameters	ddcChannelHandle – DDC level channel object
Output Parameters	Return DDC error status prevDarkFrameStatus – dark frame subtract failure status will be stored in this field.
Description	It will get the information regarding the dark frame Status.
Preconditions	Channel handle should be in configures state. Dark Frame subtraction operation should be

	enabled
Design	Logic in steps 2) validate input parameters and check channel is configured or not. 3) Get the status of darkframe subtraction by calling LLC_prevDarkFrameStatus, it will also clear the status in hardware.

4.5.13 DDC_prevPreview

Function	DDC_prevPreview
Function Prototype	DDC_prevResult DDC_prevPreview(DDC_prevChannelHandle ddcChannelHandle, PSP_preview *DDC_previewHandle)
Input Parameters	ddcChannelHandle – DDC level channel object DDC_previewHandle - Pointer to PSP_preview structure
Output Parameters	Return DDC error status
Description	This function will trigger previewer.
Preconditions	Channel handle should be in configures state.
Design	Logic in steps: 1) Validate PSP_preview parameters by calling DDC_prevValidateInputParams and DDC_prevValidateOutputParams. 2) check the status of previewer by calling DDC_prevGetStatus, if it is busy, return an error. 3) Call PrevOneShotPreviewer 4) Wait on SemIsr 5) Call LLC_prevGetWriteBufMemOverflow to get Write buffer memory overflow. and in case it is error return an error.

4.5.14 DDC_prevValidateInputParams

Function	DDC_prevValidateInputParams
Function	DDC_prevResult

Prototype	DDC_prevValidateInputParams(DDC_prevChannelHandle ddcChannelHandle, Int32 inBufSize)
Input Parameters	ddcChannelHandle – handle to channel object inBufSize – input buffer size
Output Parameters	Return DDC error status
Description	This function will validate input buffer size as per configuration parameter.
Preconditions	None
Design	Logic in steps: 1) calculate number of bytes needed for input image. 2) Consider down sample rate also while calculating it. 3) Check that calculated size is less then given size.

4.5.15 DDC_prevValidateOutputParams

Function	DDC_prevValidateOutputParams
Function Prototype	DDC_prevResult DDC_prevValidateOutputParams(DDC_prevChannelHandle ddcChannelHandle, Int32 outBufSize, Ptr outBuf)
Input Parameters	ddcChannelHandle – handle to channel object outBufSize – input buffer size outBuf – pointer to output buffer
Output Parameters	Return DDC error status
Description	This function will validate output buffer size and out buffer pointer as per configuration parameter.
Preconditions	None
Design	Logic in steps:

	1) calculate number of bytes needed for output image. 2) Consider crop size also while calculating it. 3) Check that calculated size is less then given size check also that out buffer pointer is 32 bytes aligned.

4.5.16 DDC_prevSetReadReqExpand

Function	DDC_prevSetReadReqExpand
Function Prototype	DDC_prevResult DDC_prevSetReadReqExpand(DDC_prevChannelHandle ddcChannelHandle, PSP_previewerReadReqExp *prevReadReqExp)
Input Parameters	ddcChannelHandle – handle to channel object prevReadReqExp – pointer to PSP_previewerReadReqExp structure to store prevExp value
Output Parameters	Return DDC error status
Description	This function will call LLC_prevSetReadReqExpand to set prevEXp value.
Preconditions	None
Design	Logic in steps: 1) Validate parameters 2) Call LLC_prevSetReadReqExpand

4.5.17 DDC_prevGetInfoForCCDC

Function	DDC_prevGetInfoForCCDC
Function Prototype	DDC_prevResult DDC_prevGetInfoForCCDC(DDC_prevChannelHandle, ddcChannelHandle, PSP_previewerGetInfoForCCDC *prevInfo)
Input Parameters	ddcChannelHandle – handle to channel object

	prevInfo – pointer to PSP_previewerGetInfoForCCDC
Output Parameters	Return DDC error status
Description	It is used to pass info required by CCDC..
Preconditions	None
Design	Logic in steps: 1) Validate parameters

4.5.18 DDC_prevISR

Function	DDC_prevISR
Function Prototype	Void DDC_prevISR()
Input Parameters	
Output Parameters	NULL
Description	It will be called when Previewer completes processing of one frame.
Preconditions	None
Design	Logic in steps 1) Post the SemIsr of the global structure.

4.5.19 DDC_prevSetDFC

Function	DDC_prevSetDFC
Function Prototype	DDC_prevResult DDC_prevSetDFC(DDC_prevChannelHandle ddcChannelHandle, PSP_previewerDarkFrameCapture *prevSetDFC);
Input Parameters	ddcChannelHandle – handle to channel object prevSetDFC – parameter structure passed by application, according to that dark frame capture feature will be enabled or disabled.
Output Parameters	Return DDC error status

Description	It will set dark frame capture parameter..
Preconditions	None
Design	Logic in steps: Validate parameters. If darkFrameCapture needs to be enable, validate outpitch. And call LLC_prevSetDFC to enable dark frame capture. If dark frame capture needs to be disabled, call LLC_prevSetDFC to disable it.

4.5.20 DDC_prevGetDFC

Function	DDC_prevGetDFC
Function Prototype	DDC_prevResult DDC_prevGetDFC(DDC_prevChannelHandle ddcChannelHandle, PSP_previewerDarkFrameCapture *prevGetDFC);
Input Parameters	ddcChannelHandle – handle to channel object prevGetDFC – parameter structure passed by application, according to that dark frame capture feature will be enabled or disabled.
Output Parameters	Return DDC error status
Description	It will get dark frame capture parameter..
Preconditions	None
Design	Logic in steps: Validate parameters Copy darkframecapture structure from driver channel structure to passed structure.

4.6 LLC Layer Functions

4.6.1 LLC_prevResetPreviewer

Function	LLC_prevResetPreviewer
Function Prototype	CSL_Status LLC_prevResetPreviewer(Ptr regs);
Input Parameters	regs – pointer to previewer register base address.
Output Parameters	Return CSL error code or CSL_SOK
Description	It will reset all register values in previewer.
Preconditions	None
Design	Logic in steps 1) through regs pointer set all registers to their default values.

4.6.2 LLC_prevHardwareSetup

Function	LLC_prevHardwareSetup
Function Prototype	CSL_Status LLC_prevHardwareSetup(PSP_previewerChannelCreateMode prevChannelCreateMode, PSP_previewerParams *prevConfigParams, Ptr regs)
Input Parameters	prevChannelCreateMode – it tells that whether source is CCDC or SDRAM. DDC_prevParamHandle - Pointer to PSP_prevparams according to which values will be filled in previewer registers. regs – pointer to previewer register base address.
Output Parameters	Return CSL error code or CSL_SOK
Description	This Function will write all the parameters into the hardware registers except enable bit and source dest address.
Preconditions	channel handle should not be NULL.

Design	Logic in steps 1) From the channel config parameter structure fill every value into the hardware register as per source by doing appropriate masking.

4.6.3 LLC_prevOneShotPreviewer

Function	LLC_PrevOneShotPreviewer
Function Prototype	CSL_Status LLC_PrevOneShotPreviewer(PSP_previewerChannelCreateMode prevChannelCreateMode, PSP_preview *prevPreviewParams, Ptr regs)
Input Parameters	prevPreviewParams - Pointer to PSP_preview structure regs – pointer to previewer register base address
Output Parameters	Return CSL error code or CSL_SOK
Description	This Function will enable the previewer engine by writing the enable bit to hardware register.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps Set source dest registers as per values in PSP_preview structure. If source is CCDC, source address register will not be written. clear buffer memory overflow bit. Set enable bit into the previewer PCR register.

4.6.4 LLC_prevChannelStatus

Function	LLC_prevChannelStatus
Function Prototype	void LLC_prevChannelStatus(Uint8 *channelStatus, Ptr regs)
Input Parameters	regs – pointer to previewer register base address

Output Parameters	channelStatus - Pointer to capture channel status Return CSL error code or CSL_SOK
Description	This Function will get the previewer hardware status by reading from register.
Preconditions	
Design	Logic in steps Read register to get the status of previewer hardware.

4.6.5 LLC_prevDarkFrameStatus

Function	LLC_prevDarkFrameStatus
Function Prototype	CSL_Status LLC_prevDarkFrameStatus(Uint8 *darkFrameStatus, Ptr regs)
Input Parameters	regs – pointer to previewer register base address
Output Parameters	darkFrameStatus - Pointer to capture dark frame status Return CSL error code or CSL_SOK
Description	This Function will get the dark frame status by reading from register and also will clear that bit in register.
Preconditions	
Design	Logic in steps Read register to get the status of dark frame subtract status. Clear the register bit.

4.6.6 LLC_prevSetReadReqExpand

Function	LLC_prevSetReadReqExpand
Function Prototype	CSL_Status LLC_prevSetReadReqExpand(Uint16 prevExp,Ptr regs)
Input Parameters	prevExp - read request expand value regs – pointer to vppss register base address

Output Parameters	Return CSL error code or CSL_SOK
Description	This Function will set the read request expand value
Preconditions	
Design	Logic in steps Set value in prevExp in SDR_REQ_EXP register.

4.6.7 LLC_prevGetWriteBufMemOverflow

Function	LLC_prevGetWriteBufMemOverflow
Function Prototype	CSL_Status LLC_prevGetWriteBufMemOverflow(Uint8 *prevWBLO,Ptr regs)
Input Parameters	regs – pointer to vppss register base address
Output Parameters	prevWBLO – to take overflow status Return CSL error code or CSL_SOK
Description	This Function will get the overflow status
Preconditions	
Design	Logic in steps get value of previewer overflow bit from VPSS PCR register.

4.6.8 LLC_prevPreviewerStatus

Function	LLC_prevPreviewerStatus
Function Prototype	CSL_Status LLC_prevPreviewerStatus(Uint8 *previewerStatus, Ptr regs)
Input Parameters	regs – pointer to previewer register base address
Output Parameters	previewerStatus - Pointer to previewer Status Return CSL error code or CSL_SOK

Description	This Function used to read whether previwer is enabled or not.
Preconditions	
Design	Logic in steps Read register to get the status of Previewer enable bit.

4.6.9 LLC_prevSetDFC

Function	LLC_prevPreviewerStatus
Function Prototype	CSL_Status LLC_prevSetDFC(Uint8 DFCState, Uint16 outPitch, Ptr regs)
Input Parameters	regs – pointer to previewer register base address outPitch : OutPitch
Output Parameters	DFCState – Dark Frame Capture State Return CSL error code or CSL_SOK
Description	This Function used to set dark frame capture parameter.
Preconditions	
Design	Logic in steps Enable or disable dark frame capture in PCR register. Set outpitch value in register.

5 Decision Analysis & Resolution

5.1 DAR Criteria

None

5.2 Available Alternatives

None

5.3 Decision

None

6 Revision History

Version #	Date	Author Name	Revision History
Draft 1.01	16 OCT 2006	EI2	Initial Draft Created
Draft 1.02	19 OCT 2006	EI2	Updated for technical review comments.
Draft 1.03	25 OCT 2006	EI2	Updated for QA review comments.
Issue 1.00	25 OCT 2006	EI2	Issued to TII
Issue 1.01	19 NOV 2006	EI2	Updated after incorporating Changes found during coding.
Pre-silicon Release 0.3.0	20 NOV 2006	EI2	Release to TI
Post-silicon Release 0.3.0	30 NOV2006	EI2	Release to TI
Post-silicon Release 0.3.0	21 Dec 2006	EI2	Updated for dark frame capture feature
1.00.02	29 June 2007	Amit Chatterjee	Modified Release Version
1.00.03	18 July 2007	EI2	Modified Release Version
1.00.04	29th November, 2007	Sivaraj R	PSP merge package changes - directory structure changes

««« § »»»