TII DM6437 VPSS Drivers

Resizer 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.

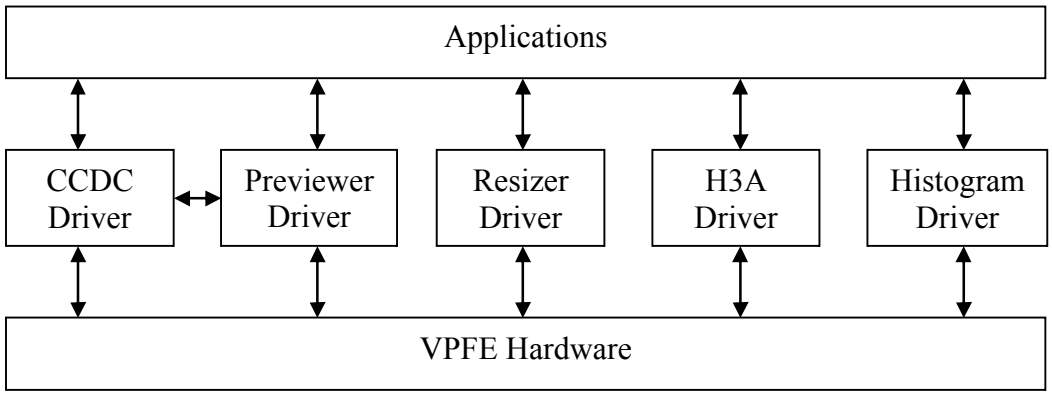

Figure 1. VPFE Driver Stack

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 Resizer 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 scope of Resizer Driver is to develop a Driver for Resizer module of DM6437. It should be fully programmable and configurable and all hardware features shall be supported. The basic functionality of Resizer driver is to upscale and/or downscale the image, provided by application.

It will support following features:

- Driver should support YUV422 color interleaved and 8-bit color separate data input formats.
- Resizer driver will support input from SDRAM or DDRAM. Driver will not support On-the-fly mode.
- Resizer driver should be a standalone module so that it can be used by multiple applications.
- Driver should support multiple access of the resizer by different application threads, but at low driver level the hardware needs to be protected from simultaneously accessing it.

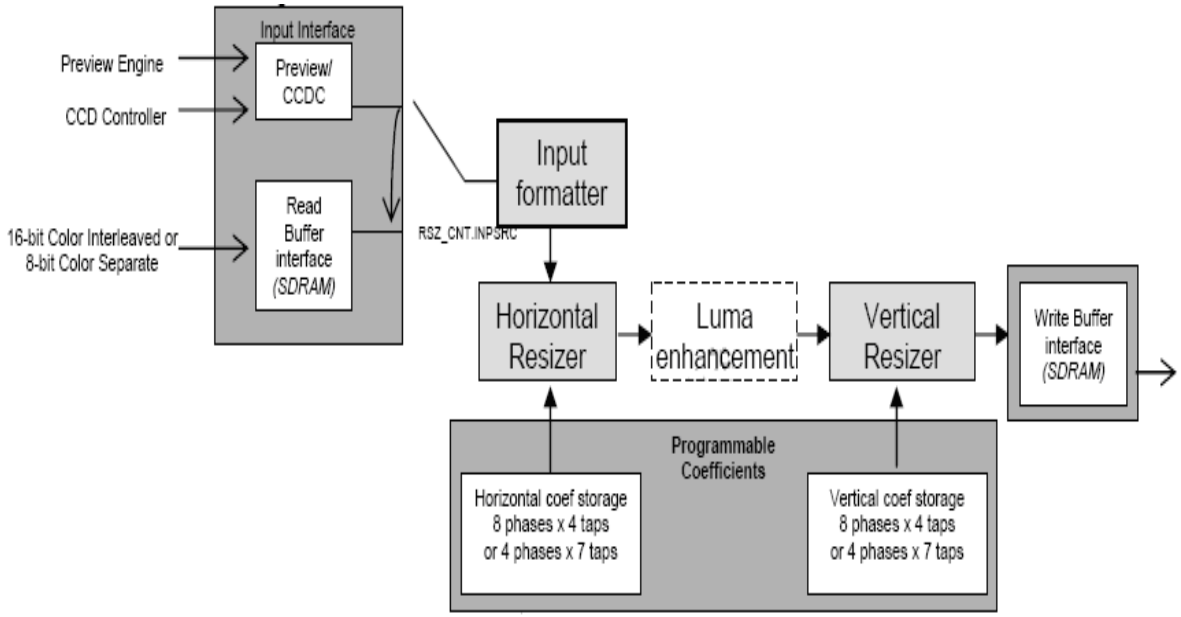

Figure 2. Block Diagram of Resizer HW

2 Requirements

The DM6437 resizer driver Functional requirements are as follows:

- Driver shall support input from DDR.
- Driver shall support output to DDR.
- Driver shall support Single pass resizing. The multiple passes for larger buffers should be handled in the application space.
- Driver shall support user configurable coefficient programming.
- Driver shall support Configuration of Luminance enhancement.
- Driver shall support multiple virtual channels with priority support.

2.1 Assumptions

None

2.2 Constraints

Resizer Output Width

- The output width should be even number of pixels. Minimum output width can be 16 pixels.
- The output width cannot be greater than 1280 pixels if the vertical resizing ratio is between 1/2x to 4x and 640 pixels wide if the vertical resizing ratio is between 1/2x to 1/4x.

Resizer Input Address

- SDRAM or DDRAM source address must be 32-byte aligned.

Resize Ratio

- As actual Resize Ratio is calculated using specific formula, actual value for maximum & minimum scaling factor are little less than 4X & little more than 0.25X.

3 Design Description

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

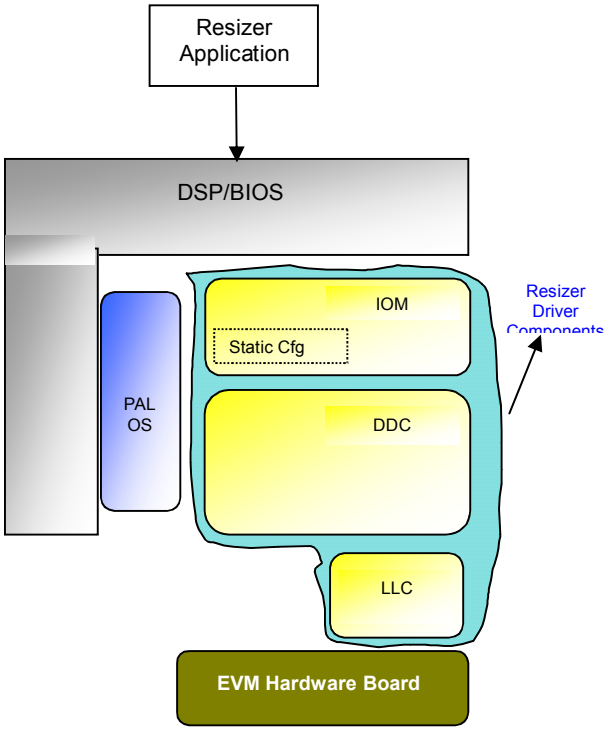

Figure 3. Overall Design of Resizer Driver

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

For further reference of PSP architecture, please refer to "DM6437_BIOS_PSP_User_Guide.pdf" document.

3.1.2 Dynamic view

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

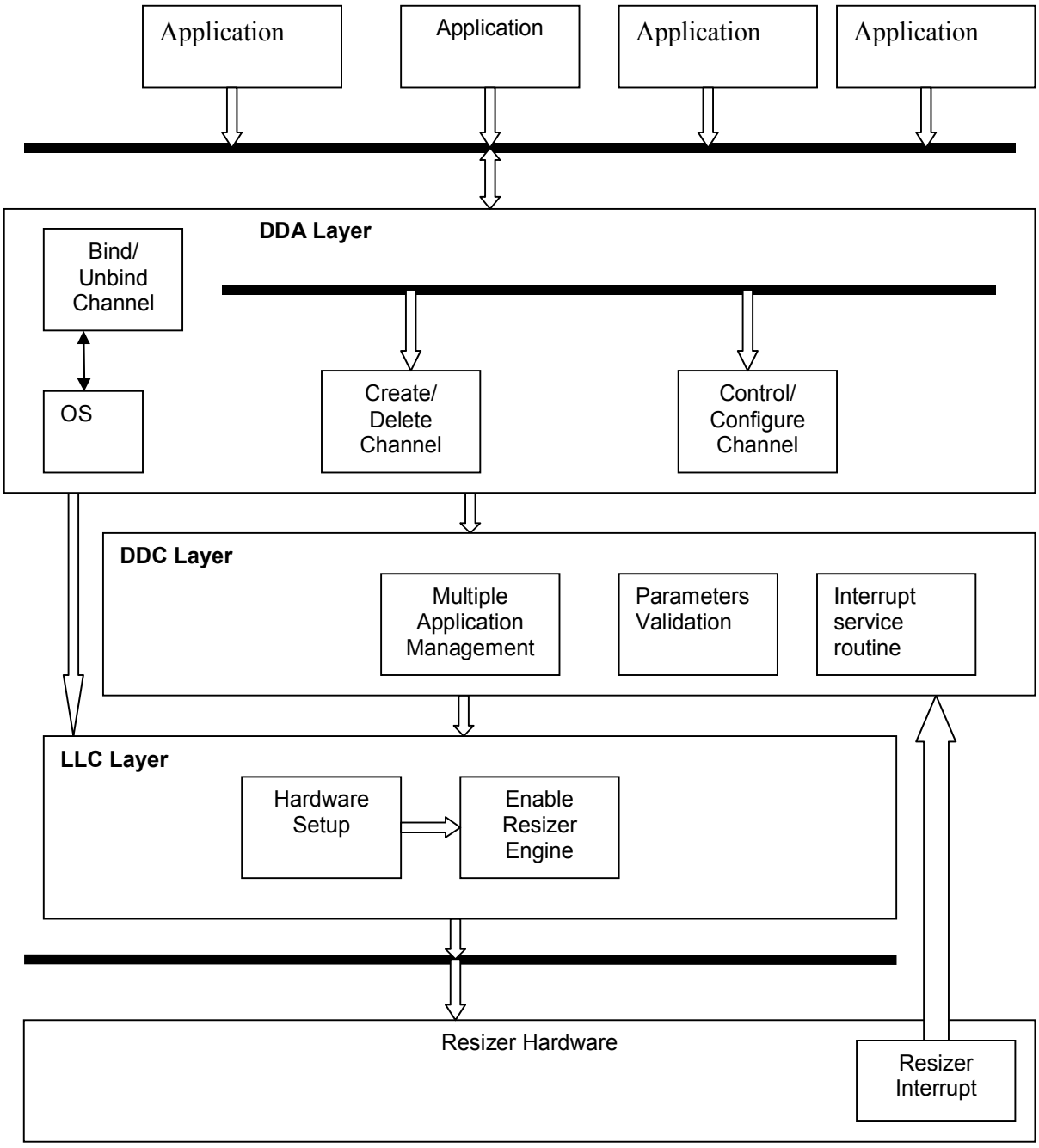

Figure 4. Detailed Design Diagram for Resizer

Various functionalities of different layers in Resizer 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 resizer engine

3.1.2.1 Driver Creation/deletion

Driver Creation

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

The RSZ_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.

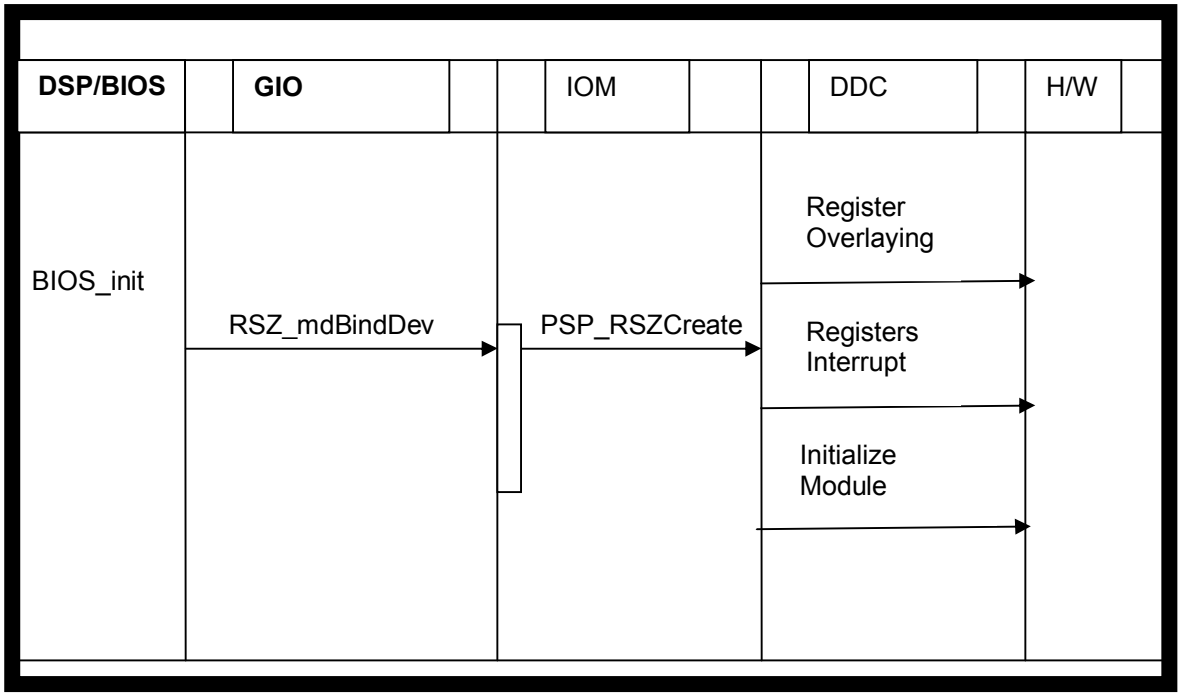

Figure 5. Driver Creation overview

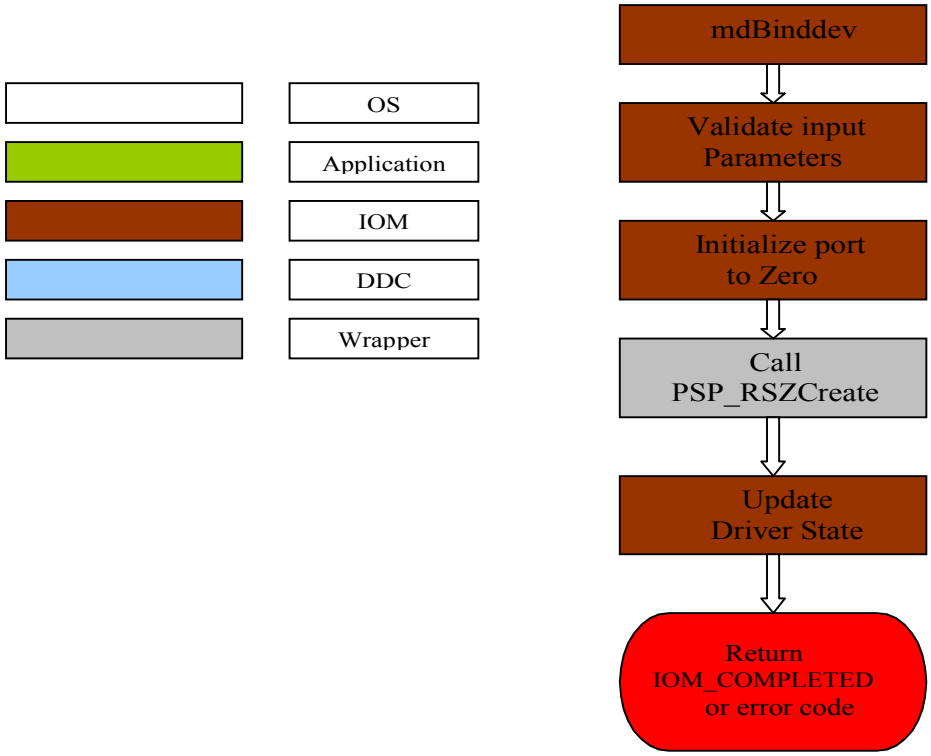

Figure 6. Driver Creation detail flow diagram-1

PSP_RSZCreate

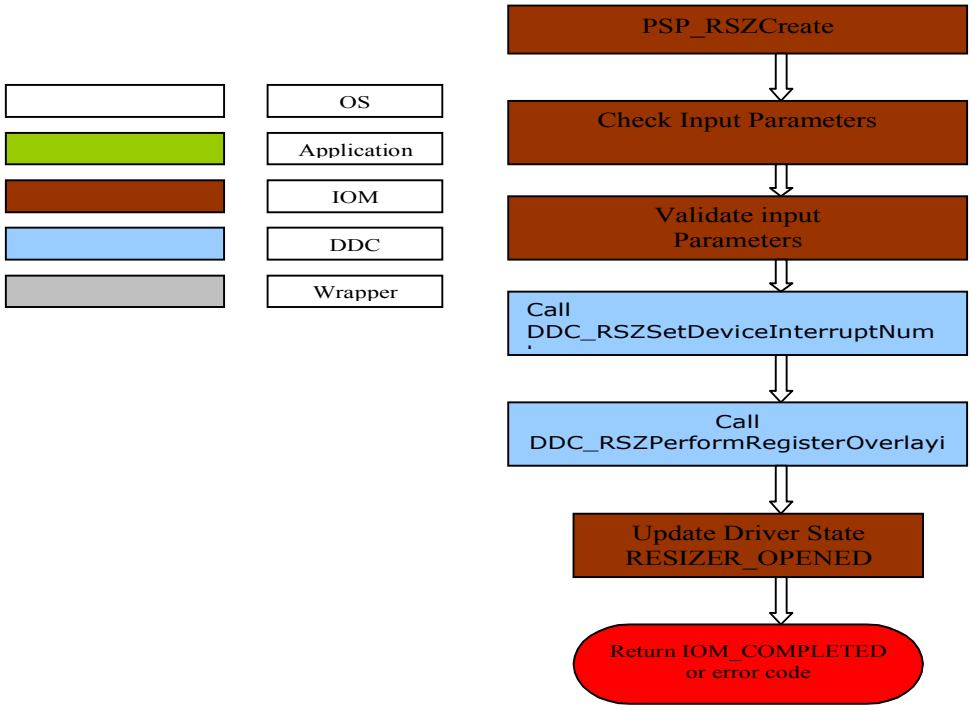

Figure 7. Driver Creation Detail flow diagram -2

Driver Deletion

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

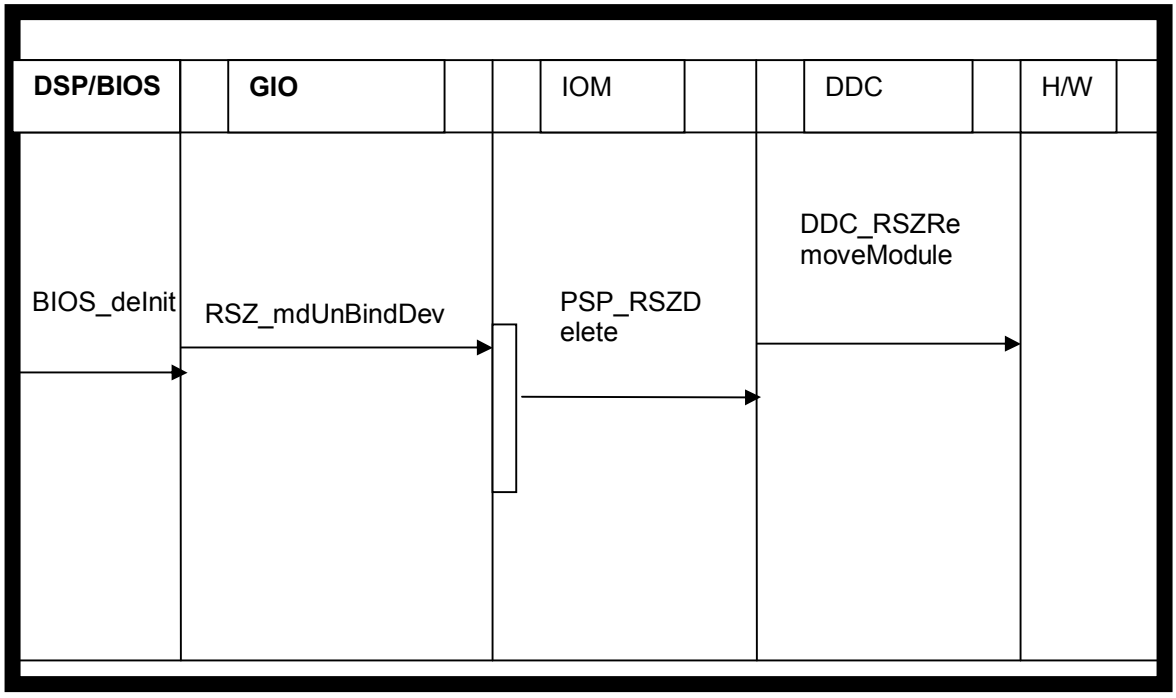

Figure 8. Driver deletion overview

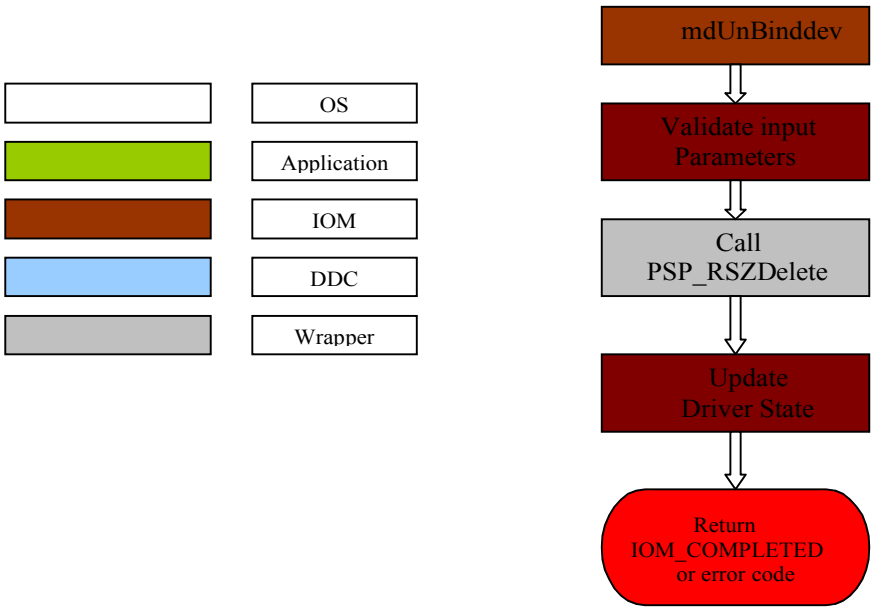

Figure 9. Driver Deletion detail Flow diagram -1

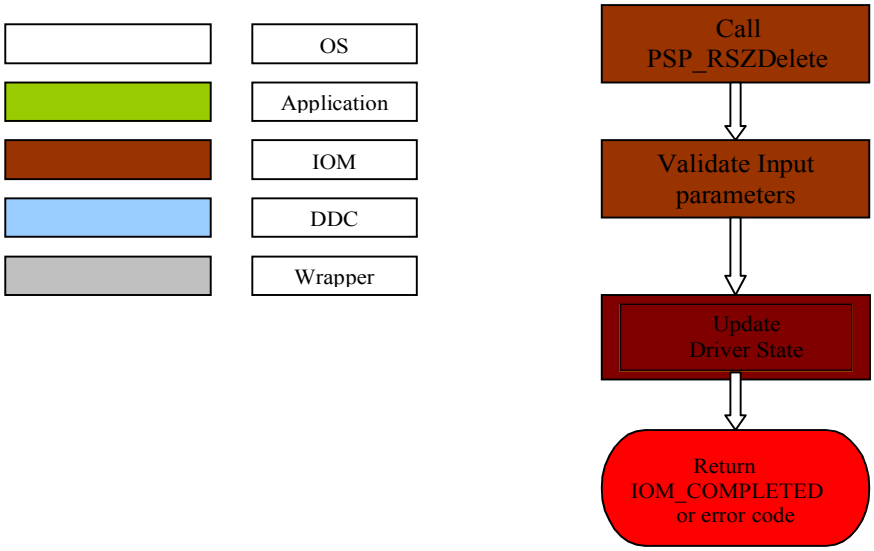

Figure 10. Driver deletion detail flow diagram -2

3.1.2.2 Driver Open/Close

Driver Open

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

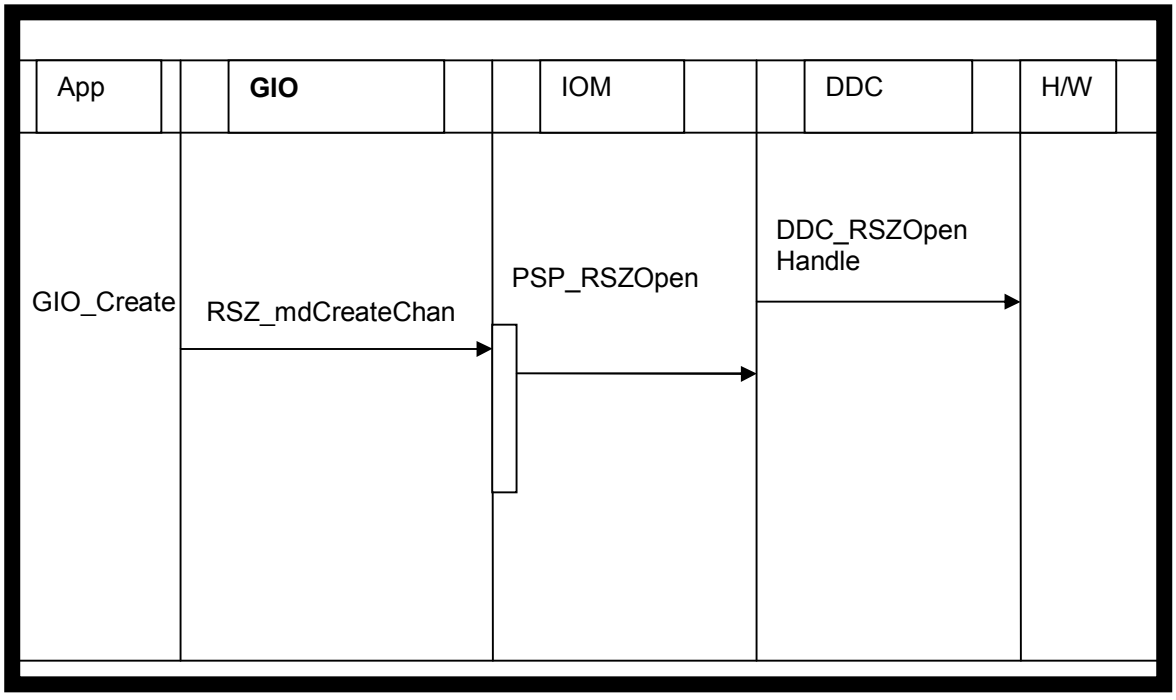

Figure 11. Driver create channel overview

RSZ_mdCreateChan

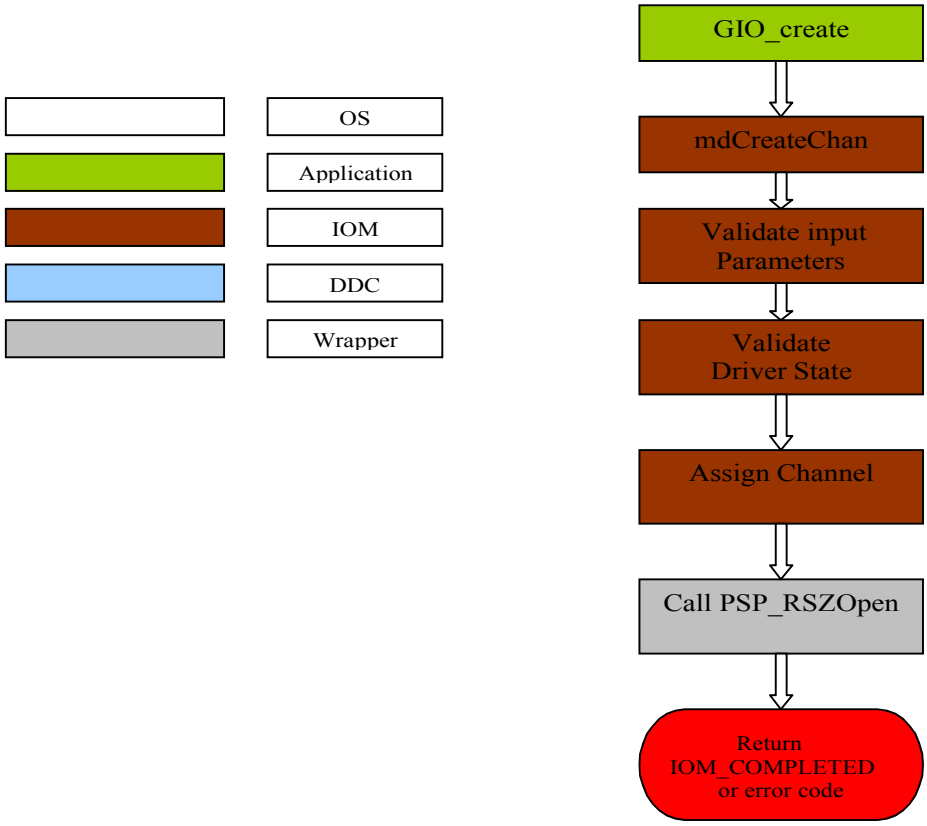

Figure 12. Driver Open Detail flow diagram -1

PSP_RSZOpen

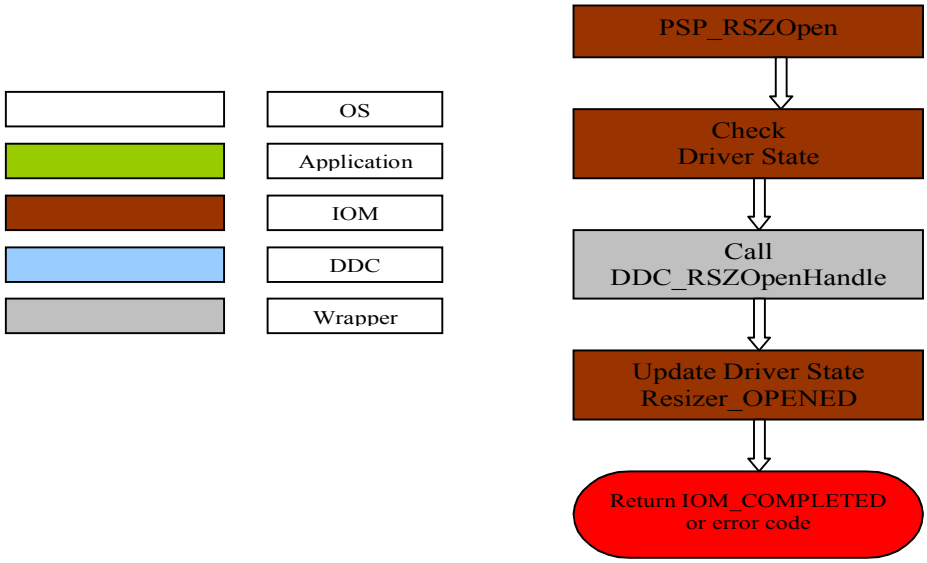

Figure 13. Driver Open detail flow diagram -2

DDC_RSZOpenHandle

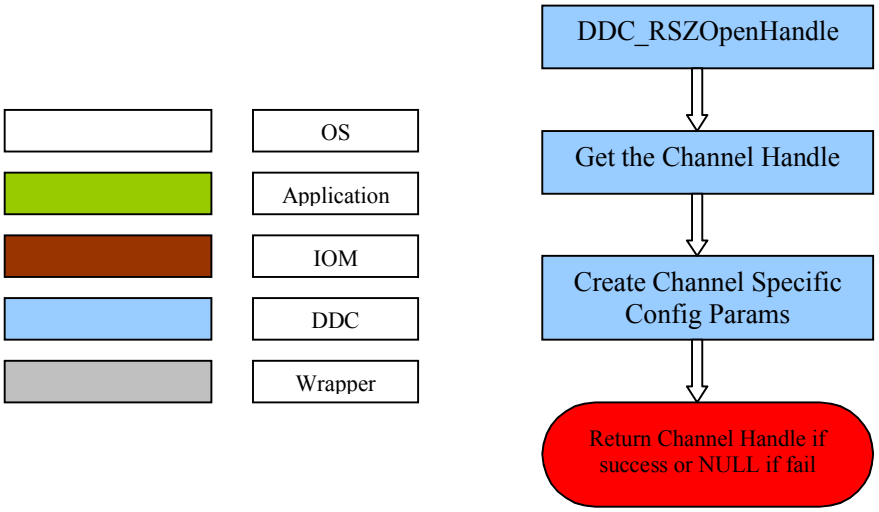

Figure 14. Driver Open detail flow diagram -3

Driver Close

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

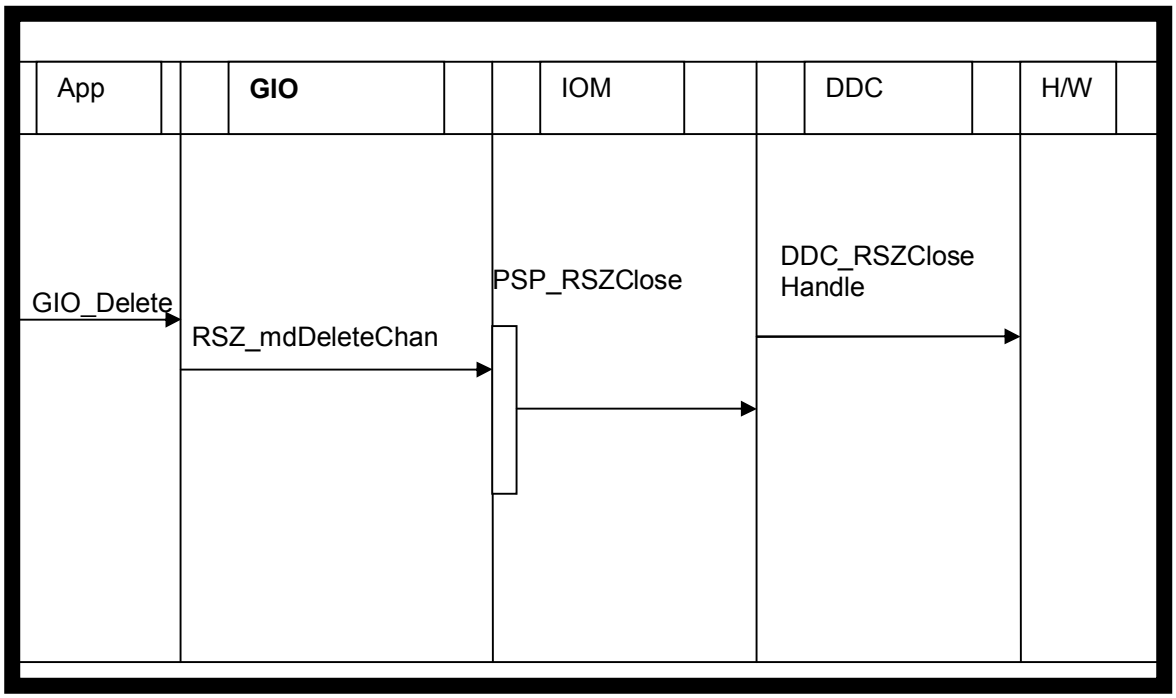

Figure 15. Driver channel close overview

Rsz_mdDeleteChan

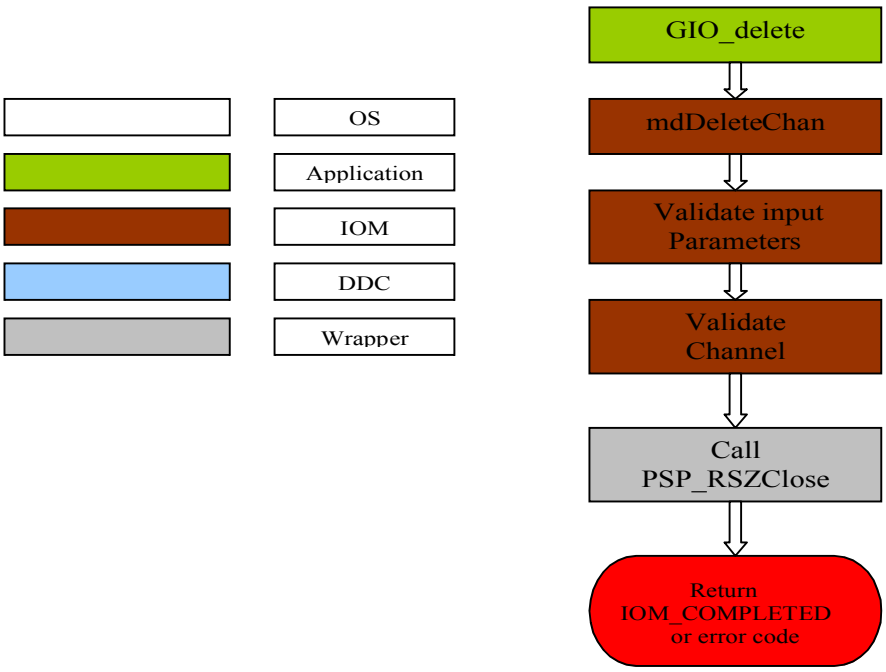

Figure 16. Driver close channel detail flow diagram -1

PSP_RSZClose

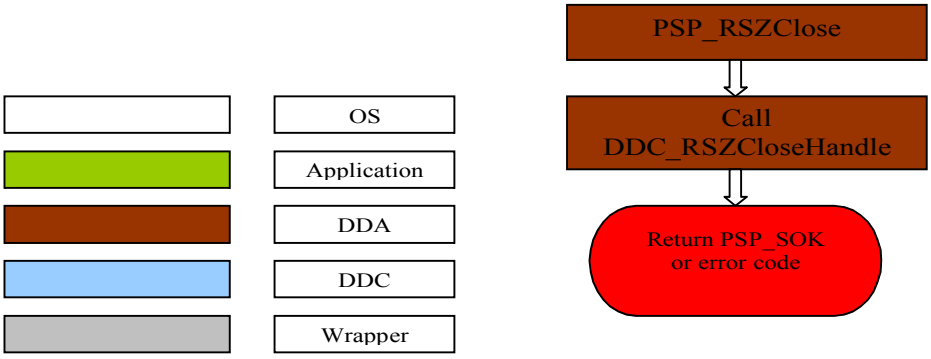

Figure 17. Driver close channel detail flow diagram -2

DDC_RSZCloseHandle

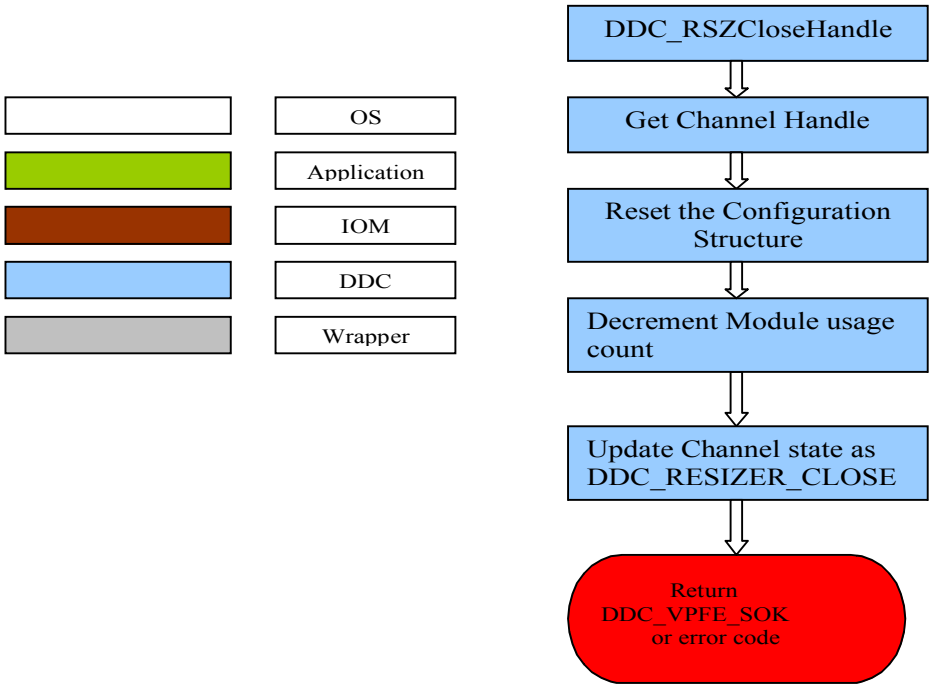

Figure 18. Driver close Channel detail flow diagram -3

3.1.2.3 Various Controls

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

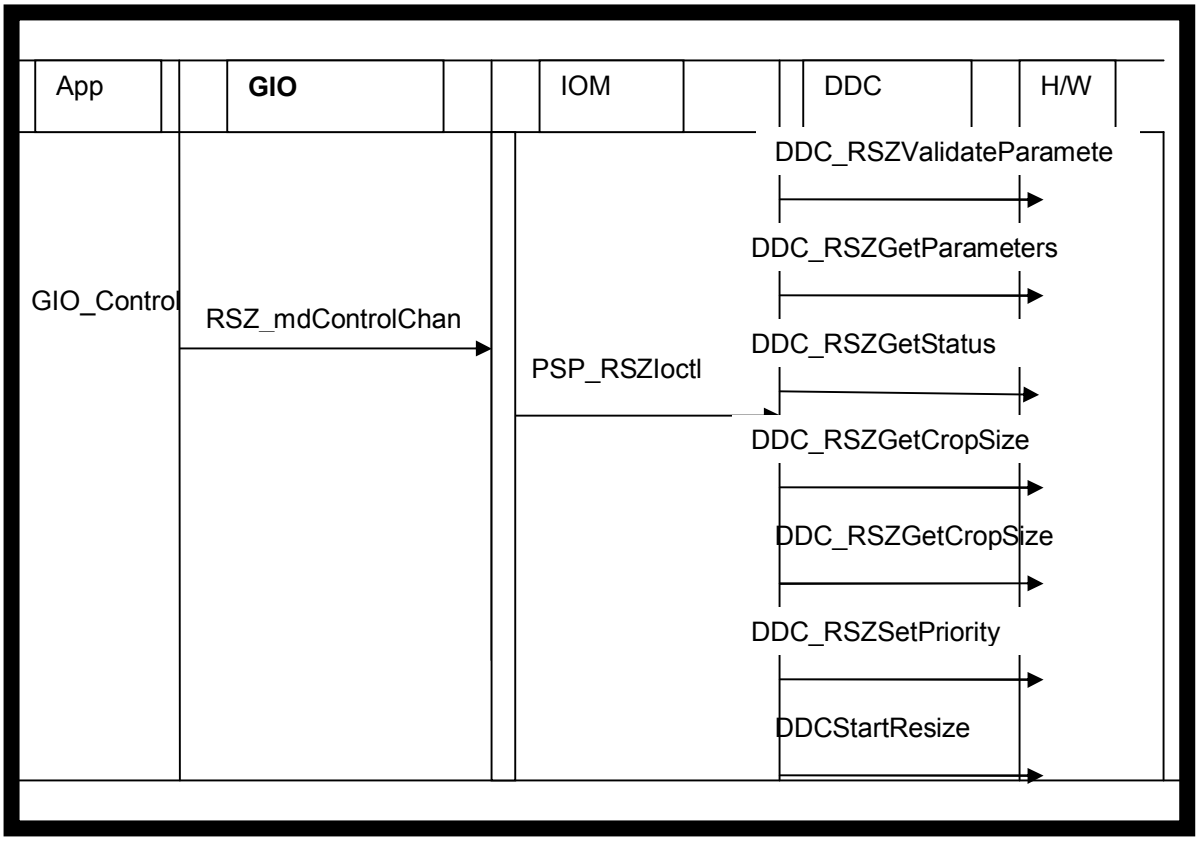

Figure 19. Control Command overview

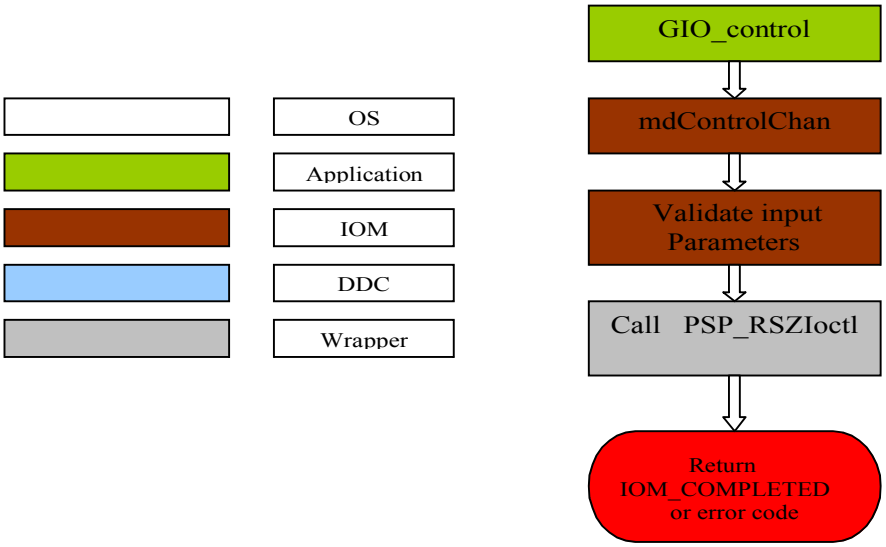

Figure 20. Control Command detail flow diagram -1

DDC Layer Functions

3.1.2.4 Multiple Channel Handling

This sub module handles multiple requests to access the hardware. As shown in the below, this will be tightly coupled with the ioctl function for PSP_RSZ_IOCTL_RESIZE command. IOCTL function used three main functionalities of this module:

Request Queuing:

When application issues a resizing request then this module first checks the hardware. If hardware is free then it allows the application to access the hardware. If some other application is using the hardware then it submits the request to the pending queue and pends on the channel special Mutex (ChannelSem). At the time of adding the channel object handle to the pending queue this module also sorts all requests, i.e. existing plus new, in descending order of their priorities. It also changes channel status to busy.

De-queuing: After the resizing is completed, interrupt handling module signals to ioctl using a mutex. The ioctl function then requests to remove current request from the pending queue. De-queuing function then, removes the specified channel object handle from the pending queue. This module also returns the channel object handle for the next highest priority request in the pending queue to the ioctl function. IOCTL functions uses this channel object handle to post the channel specific Mutex (ChannelSem) to transfer the control of Resizer hardware to its kernel task.

Priority Management: Application can set the priority of the resizing channel. Possible priority values are in between 0 to 5. If there are multiple applications waiting to get access of the Resizer engine, application with highest priority will get access to the engine.

For Example, let's take three applications with each configuring low (1), medium (2) and high (3) priority for their resizing task. Assume that low priority task has submitted first request to the resizing driver, and then medium and high priority tasks submits their requests before the low priority request is completed. Hence Medium and high priority tasks will be blocked and kept in a pending queue, which is sorted in descending order of priority i.e. higher priority task remains at the top of the queue. When low priority task is completed, it will take the highest priority task entry from the queue and unblock that task. Hence high priority request will be executed by the Resizer hardware. When high priority request is completed, it will take entry for next highest priority task, which is medium priority task, from the pending queue and unblock that task. So now, medium priority task is submitted to the Resizer hardware.

Figure below explains the Multiple Channel Handling

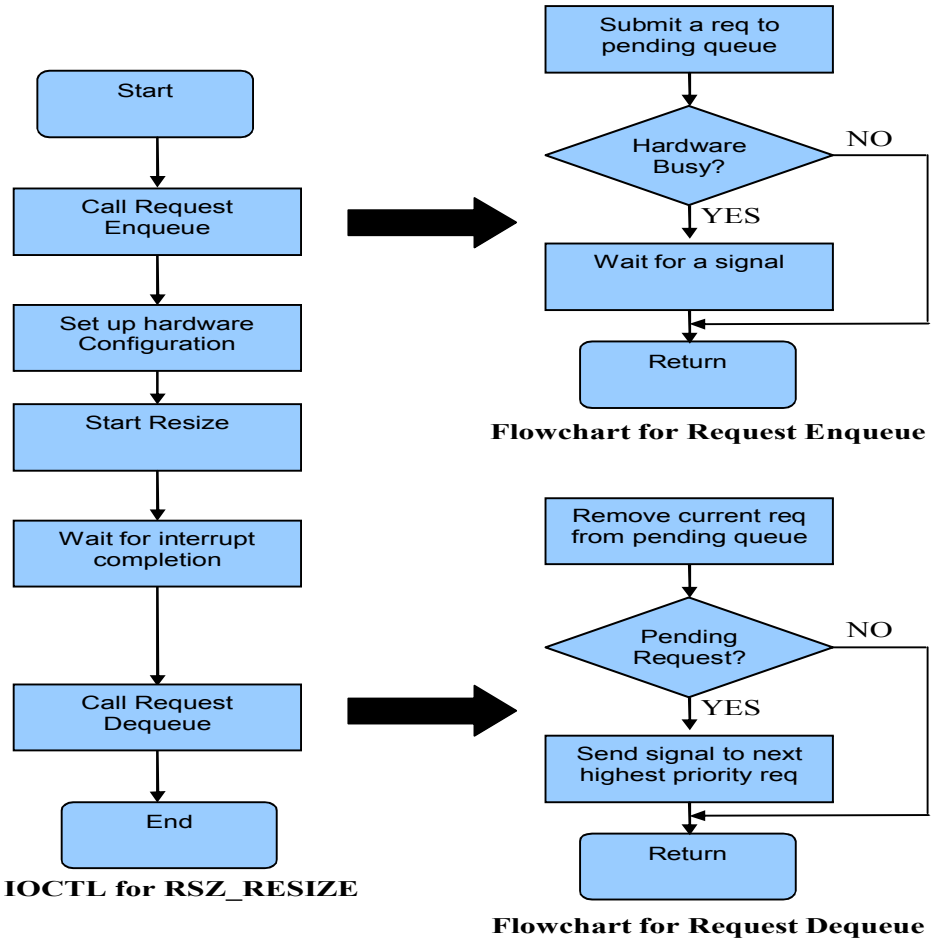

Figure 21. Multiple Channel Handling

3.1.2.5 Interrupt Service Routine

Resizer Driver Interrupt Handles will be attached with the Resizer Hardware Interrupt of DM6437. Resizer driver will implement interrupt dispatcher which will handle interrupts generated for the completion of Resizing 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 Resize completion to the Resize function of the configuration & control module.

3.1.2.6 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_RSZ_IOCTL_SET_PARAMS command.

4 Low Level Definitions

This section describes low level details of Resizer Driver.

4.1 Constants & Enumerations

CONSTANTS:

4.1.1 Maximum Channel

It describes the maximum number of channels that can be created.

Definition

```
#define PSP_RESIZER_NUM_CHANNELS    16u
```

Comments

It can be changed via changing the value of the constant

Constraints

None

4.1.2 Maximum Devices

It describes the maximum number of resizer devices that can be created.

Definition

```
#define PSP_RSZ_NUM_INSTANCES 1u
```

Comments

None

Constraints

Only one resizer device can be created.

4.1.3 Input Type

It describes the input image type.

Definition

```
#define PSP_RSZ_INTYPE_YCBCR422_16BIT      0
#define PSP_RSZ_INTYPE_PLANAR_8BIT         1
```

Comments

The input type does not describe the position of cb and cr component.

Constraints

None

4.1.4 Chrominance algorithm

It describes that chrominance algorithm processing is bilinear or same as luma processing.

Definition

```
#define PSP_RSZ_CBILIN_DISABLE      0
#define PSP_RSZ_CBILIN_ENABLE       1
```

Comments

If chrominance algorithm is disabled than processing is same as luma processing.

Constraints

None

4.1.5 Luma Configuration

Describe the luma edge enhancement with 3-tap or 5-tap if luma option is enabled

Definition

```
#define PSP_RSZ_YENH_DISABLE         0
#define PSP_RSZ_YENH_3TAP_HPF        1
#define PSP_RSZ_YENH_5TAP_HPF        2
```

Comments

Performs luma enhancement with 3 or 5 tap.

Constraints

None

4.1.6 Input src

Describe that input image if not from previewer or CCDC

Definition

```
#define PSP_RSZ_SRC_RAM             1
```

Comments

None

Constraints

None

4.1.7 Priority Settings

Describes the maximum, minimum and default priority of the channel

Definition

```
#define DDC_RSZ_MAX_PRIORITY          5
#define DDC_RSZ_MIN_PRIORITY          0
```

```
#define DDC_RSZ_DEFAULT_PRIORITY      0
```

Comments

None

Constraints

None

ENUMARATORS:

4.1.8 Controls Commands

This enum describes various control commands.

Definition

```
typedef enum
{
    PSP_RSZ_IOCTL_SET_PARAMS = 128,
    PSP_RSZ_IOCTL_GET_PARAMS,
    PSP_RSZ_IOCTL_RESIZE,
    PSP_RSZ_IOCTL_SET_PRIORITY,
    PSP_RSZ_IOCTL_GET_PRIORITY,
    PSP_RSZ_IOCTL_GET_STATUS,
    PSP_RSZ_IOCTL_GET_CROPSIZE,
    PSP_RSZ_IOCTL_SET_EXP

} PSP_RSZIoctlCmd;
```

Comments

None

Constraints

None

4.1.9 Parameters State

This enum describes resizer configuration status.

Definition

```
enum DDC_ParamsConfigDone
{
    STATE_CONFIGURED,
    STATE_NOT_CONFIGURED
};
```

Comments

None

Constraints

None

4.1.10 Resizer device and channel state

This enum describes various states for resizer driver.

Definition

```
typedef enum RESIZER_STATE
{
      PSP_RESIZER_DELETED,
      PSP_RESIZER_CREATED,
      PSP_RESIZER_OPENED,
      PSP_RESIZER_CLOSED
} resizerState;
```

Comments

None

Constraints

None

4.1.11 Resizer Channel state

This enum describes status for resizer channel.

Definition

```
typedef enum _DDC_RSZState
{
      DDC_RSZCHANCLOSED,
      DDC_RSZCHANOPENED,
      DDC_RSZChannelFREE,
      DDC_RSZChannelBUSY
} DDC_RSZChanState;
```

Comments

None

Constraints

None

4.1.12 Resizer return values

This enum describes return values.

Definition

```
typedef enum _DDC_RSZStatus
{
      DDC_RSZ_E_FAIL = 0,
      DDC_RSZ_SOK,
      DDC_RSZ_INVALID_STATE,
      DDC_RSZ_INVALID_PARAM,
      DDC_RSZ_NOT_SUPPORTED
} DDC_RSZStatus;
```

Comments

None

Constraints

None

4.2 Data Structures

Interface level structures

4.2.1 Luma enhancement parameters

This structure is used to configure the luma edge enhancement parameters

Definition

```
typedef struct _PSP_RSZyenhParams
{
    Int type;
    Uint8 gain;
    Uint8 slop;
    Uint8 core;
}PSP_RSZyenhParams;
```

Fields

type	Represents luma enable or disable
gain	Represents gain on the edges
slop	Represents slop
core	Represents core value

Comments

If luma type is disable than all others fields are ignored.

Constraints

None

4.2.2 Resizer configuration parameters

This structure is used to configure the various resizer parameters

Definition

```
typedef struct _PSP_RSZparams
{
      Int inHsize;
      Int inVsize;
      Int inPitch;
      Int inptyp;
      Int vertStartingPixel;
      Int horzStartingPixel;
      Int cbilin;
      Int pixfmt;
      Int outHsize;
      Int outVsize;
      Int outPitch;
```

```
        Int hstph;
        Int vstph;
        Int16 hfiltCoeffs[32];
        Int16 vfiltCoeffs[32];
        PSP_RSZyenhParams yenhParams;
}PSP_RSZparams;
```

Fields

inHsize	Represents input frame horizontal size
inVsize	Represents input frame vertical size
inPitch	Represents offset between two rows of input frame
Inptyp	For determining 16 bit or 8 bit data
vertStartingPixel	For specifying vertical starting pixel in input
horzStartingPixel	For specifying horizontal starting pixel in input
Cbilin	Defined, filter with luma or bi-linear Interpolation
Pixfmt	Defined, UYVY or YUYV image format
outHsize	Represents output frame horizontal size
outVsize	Represents output frame vertical size
outPitch	Represents offset between two rows of output frame
Hstph	For specifying horizontal starting phase
Vstph	For specifying vertical starting phase
hfiltCoeffs	Horizontal filter coefficients
vfiltCoeffs	Vertical filter coefficients
yenhParams	Represents luma enhancement parameters

Comments

All parameters should be configured before resizing.

Constraints

None

4.2.3 Resize

This structure is used to pass the address of input and output buffer.

Definition

```
typedef struct _PSP_RSZresize
{
    Ptr  *inBuf;
    Ptr  *outBuf;
```

```
    Int inBufSize;
    Int outBufSize;
}PSP_RSZresize;
```

Fields

inBuf	Represents input frame starting address
outBuf	Represents output frame starting address
inBufSize	Represents input buffer size
outBufSize	Represents output buffer size

Comments

None

Constraints

Input and output address should be 32-bit aligned.

4.2.4 Resize status

This structure is used to get the status of resizing.

Definition

```
typedef struct _PSP_RSZstatus
{
    Int chanBusy;
    Int hwBusy;
    Int src;
}PSP_RSZstatus;
```

Fields

chanBusy	1: channel is busy, 0: channel is not busy
hwBusy	1: hardware is busy, 0: hardware is not busy
Src	Defined, can be either DDR or CCDC/PREVIEWER

Comments

None

Constraints

None

4.2.5 Resize priority

This structure is used to get and set the priority of configuration channel.

Definition

```
typedef struct _PSP_RSZpriority
{
    Int priority;
```

```
}PSP_RSZpriority;
```

Fields

priority	To set the priority of config channel.0=>5, with 5 the highest priority

Comments

None

Constraints

None

4.2.6 Resize Cropping information

This structure is used to get the cropping information.

Definition

```
typedef struct _PSP_RSZcropsize
{
    Int32 hcrop ;
    Int32 vcrop;
 }PSP_RSZcropsize;
```

Fields

hcrop	Represents number of pixels per line cropped in output image
vcrop	Represents number of lines cropped in output image

Comments

None

Constraints

None

Global Structures at IOM Layer

4.2.7 IOM Level Resizer Port structure

This structure is global structure which is used hold the port specific information.

Definition

```
typedef struct RSZPORTOBJ
{
      Uns                        rszPortNumber;
      RSZChannelObject           chan[NUMCHANS];
      DDC_RSZDeviceObject       *dda_PortObj;
      resizerState               state;
      Uint                       numOfOpens;
      SEM_Handle                 PortSemaphore;
} RSZPortObject, *RSZPortHandle;
```

Fields

rszPortNumber	Represents Instance number
Chan	Represents channels for the Resizer
dda_PortObj	Represents underlying DDA Structure
state	State of VPSS driver
numOfOpens	Represents the number of channel opened
PortSemaphore	To protect the variables of port structure

Comments

None

Constraints

None

4.2.8 IOM Layer Resizer Channel Handle

This structure is a channel handle which Contains channel specific information.

Definition

```
typedef struct RSZCHANNELOBJ
{

    struct RSZPORTOBJ           *port;
    PSP_Handle                  ddc_Handle;
} RSZChannelObject;
```

Fields

Port	Reverse pointer to Port
ddc_Handle	Handle to Resizer

Comments

None

Constraints

None

DDC Layer Channel specific Structure

4.2.9 DDC Level Resizer Device structure

This structure is global structure which is used to hold the device specific structure.

Definition

```
typedef struct _DDC_RSZDeviceObject
{
      Uint                              rszInstanceId;
```

```
        resizerState                    State;
        Uint                            IntNum;
        Ptr                             regs;
        DDC_RSZChanObject *DDC_RSZChanConfig[MAX_CHANNELS];
        Uint ArrayCount;
        PAL_OsSemHandle                 SemIsr;
        PAL_OsSemHandle                 ArraySem;
        Int                             MemSegId;

}DDC_RSZDeviceObject;
```

Fields

rszInstanceId	Represents resizer device id
state	Represents resizer device state
IntNum	Represents resizer Interrupt number
regs	Handle for resizer register overlaying structure
DDC_RSZChanConfig	Pointer to resizer parameter structure
ArrayCount	For counting number of elements in array
SemIsr	Semaphore for hardware protection
ArraySem	Semaphore to protect the array of channels
MemSegId	stores Memory segment id

Comments

None

Constraints

None

4.2.10 DDC layer Resizer Channel Specific structure

This structure is channel specific structure which holds the data regarding specific channel.

Definition

```
typedef struct _DDC_RSZChanObject
{

      DDC_RSZChanState              state;
      LLC_ResizerConfigParams       params;
      int                           priority;
      DDC_ParamsConfigDone          DDC_ParamsConfigState;
      PAL_OsSemHandle               channelSem;

} DDC_RSZChanObject;
```

Fields

params	Handle for access the resizer config params
state	Represents resizer channel state
priority	Stores priority of the channel
DDC_Params_config_state	Stores state of configuration
channelSem	Channel specific semaphore

Comments

None

4.2.11 Resizer channel config parameters

This structure is used to store various channel specific parameters.

Definition

```
typedef struct _ LLC_ResizerConfigParams
{
      Int inHsize;
      Int inVsize;
      Int inPitch;
      Int inptyp;
      Int vertStartingPixel;
      Int horzStartingPixel;
      Int cbilin;
      Int pixfmt;
      Int outHsize;
      Int outVsize;
      Int outPitch;
      Int hstph;
      Int vstph;
      Int16 hfiltCoeffs[32];
      Int16 vfiltCoeffs[32];
      PSP_RSZyenhParams yenhParams;
      Int hrsz;
      Int vrsz;
      Int inAddress;
      Int outAddress;
}LLC_ResizerConfigParams;
```

Fields

inHsize	Represents input frame horizontal size
inVsize	Represents input frame vertical size
inPitch	Represents offset between two rows of input frame
inptyp	For determining 16 bit or 8 bit data

vertStartingPixel	For specifying vertical starting pixel in input
horzStartingPixel	For specifying horizontal starting pixel in input
cbilin	Defined, filter with luma or bi-linear Interpolation
pixfmt	Defined, UYVY or YUYV image format
outHsize	Represents output frame horizontal size
outVsize	Represents output frame vertical size
outPitch	Represents offset between two rows of output frame
hstph	For specifying horizontal starting phase
vstph	For specifying vertical starting phase
hfiltCoeffs	Horizontal filter coefficients
vfiltCoeffs	Vertical filter coefficients
yenhParams	Represents luma enhancement parameters
Hrsz	Contains the horizontal resize ratio
Vrsz	Contains vertical resize ratio
inAddress	Input address of the image
outAddress	Output address of the image

Comments

All parameters should be configured before resizing.

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 "/resizer".

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.

	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. Segment id for memory is passed via these structure

	GIO_Attrs	attrs

The attrs parameter is a pointer to a structure of type GIO_Attrs. If attrs is NULL, a default set of attributes is used.

OUT	Int*	status

If the status parameter is non-NULL, a status value is placed at the address specified by the status parameter.

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 open will not be supported by the resizer driver.

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 RSZ_mdBind

Function	RSZ_mdBindDev()
Function Prototype	Int RSZ_mdBindDev (Ptr *devp, Int devid, Ptr devParams)
Input Parameters	devid – number of device instances devParams – would be the H/W configuration information pointer variable. devp – void pointer to be updated once instance is created
Output Parameters	Int returning the IOM Status
Description	This function would be implemented at the IOM layer. This function would create a DDC instance of the driver and initialize the driver as well.
Preconditions	The Driver supports only one instance so devid should not be more than 1. The driver should be opening for the first time at the OS initialization time.
Design	Logic in steps. 1) Check the number of instances created 2) Set initial global values to zero 3) Call PSP_RSZCreate function 4) Check if the handle return is NULL or Not If handle is Null return error. 5) Register the interrupt Handler 6) Update port status

4.4.2 RSZ_mdCreateChan

Function	RSZ_mdCreateChan
Function Prototype	Int RSZ_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 initialized devp – pointer to the port structure name – name of the driver.Name will differentiate between the driver. mode – mode of the driver i.e. INPUT or OUTPUT chanParams –Segment if for memory allocation cbFxn – callback function to the GIO cbArg – callback arguments
Output Parameters	Int according to the IOM errors
Description	This function declares the driver is ready for any IO transactions.
Preconditions	The DDC channel need to be initialized and it should be closed before opening
Design	Logic in steps 1) Check for the valid mode of operation of the channel 2) Check that the channel is not in use. 3) Assign the channel. 4) Set the Channel Type 5) Call PSP_RSZOpen function. 6) Update the port state and put channel as 'in use'.

4.4.3 RSZ_mdControlChan

Function	RSZ_mcControlChan
Function Prototype	Int RSZ_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 perform various control actions runtime commands on the device driver.
Preconditions	The device driver state should be opened.
Design	Logic in steps 1) Check for valid IOCTL command. 2) Call a function PSP_RSZIoctl to Execute the IOCTL command.

4.4.4 RSZ_mdUnbindDev

Function	RSZ_mdUnbindDev
Function Prototype	Int RSZ_mdUnBindDev(Ptr devp)
Input Parameters	Devp – Handle to RSZ device
Output Parameters	Int according to the IOM error code
Description	This function would remove or unload the driver instance.
Preconditions	The DDC has to be created and initialized
Design	Logic in steps. 1) Call the PSP_RSZDelete function. 2) Check the state of driver. 3) Update the port status.

4.4.5 RSZ_mdDeleteChan

Function	RSZ_mdDeleteChan
Function Prototype	Int RSZ_mdDeleteChan (Ptr chanp)
Input Parameters	Chanp – pointer to chan object
Output Parameters	Int according to the IOM error codes
Description	This function would close current session of IO by calling the DDC Layer _DDC_RSZClosehandle ( ) function.
Preconditions	The driver should be opened

Design	Logic in steps 1) Check the State of the Channel. 2) Call PSP_RSZClose Function

4.4.6 PSP_RSZOpen

Function	PSP_RSZOpen
Function Prototype	PSP_Handle PSP_RSZOpen (int)
Input Parameters	Variable holding segment id for memory allocation.
Output Parameters	NULL
Description	This function creates a channel to carry out the transaction.
Preconditions	Driver must be in Created state
Design	Logic in steps 1) Check the State of the Driver 2) Call DDC_RSZOpenHandle Function.

4.4.7 PSP_RSZClose

Function	PSP_RSZClose
Function Prototype	PSP_Result PSP_RSZClose(PSP_Handle handle)
Input Parameters	Handle – Handle of the Channel Cmd – Submit command to be passed CmdArg Argument to the command Ptr Params – Pointer to parameter structure
Output Parameters	Int according to the IOM error code
Description	It acts as wrapper, which provides the information as per command.
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps 1) Call DDC_RSZCloseHandle Function of the corresponding channel.

4.4.8 PSP_RSZIoctl

Function	PSP_RSZIoctl
Function Prototype	PSP_Result PSP_RSZIoctl(PSP_Handle handle,PSP_AFIoctlCommand cmd,Ptr cmdArg,Ptr params);
Input Parameters	Handle – Handle of the Channel Cmd – Submit command to be passed CmdArg Argument to the command Ptr Params – Pointer to parameter structure
Output Parameters	Int according to the PSP errors
Description	It acts as wrapper, which provides the information as per command.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Check the Input Parameters 2) Check Channel is Closed or not. 3) Call DDC_RSZIoctl

4.4.9 PSP_RSZCreate

Function	PSP_RSZCreate
Function Prototype	Void * PSP_RSZCreate(Int32 devid);
Input Parameters	Int32 –Device id
Output Parameters	Void * to device structure of DDC layer
Description	This function initializes the device.

Preconditions	Driver must not be in created state
Design	Logic in steps 1) Check the Input Parameters 2) Check the state of the driver 3) Initialize port to zero 4) Call Resizer DDC_RSZRegisterOverlaying depending on the device id 5) Update the state of the driver 6) call DDC_RSZSetInterruptNumber to set the interrupt number. 7) Call InitializeModule

4.4.10 PSP_RSZDelete

Function	PSP_RSZDelete
Function Prototype	PSP_Result PSP_RSZDelete(Int32 devid);
Input Parameters	Int32 –Device id
Output Parameters	Int according to the PSP errors
Description	This function initializes the device.
Preconditions	Driver must be in Created state
Design	Logic in steps 1) Check the Device id 2) Check the state of the driver 3) Call DDC_RSZRemoveModule

4.4.11 DDA_RSZRegisterIntrHandler

Function	DDA_rszRegisterIntrHandler
Function Prototype	void DDA_rszRegisterIntrHandler(Uint32 intrNum);
Input Parameters	Int32 –intrNum
Output Parameters	None

Description	This function register the interrupt handler
Preconditions	Driver must be in Created state
Design	Logic in steps 1) Call system function ECM_dispatchPlug 2) Call system function C64_enableIER 3) Call system function ECM_enableEvent

4.4.12 DDA_RSZUnRegisterIntrHandler

Function	DDA_rszUnRegisterIntrHandler
Function Prototype	void DDA_rszRegisterIntrHandler(Uint32 intrNum);
Input Parameters	Int32 –intrNum
Output Parameters	None
Description	This function deregister the interrupt handler
Preconditions	Driver must be in Created state and interrupt handler must be registered
Design	Logic in steps 1) Call system function ECM_disableEvent

4.5 DDC Layer Functions

4.5.1 DDC_RSZPerformRegisterOverlaying

Function	DDC_RSZPerformRegisterOverlaying
Function Prototype	void DDC_RSZPerformRegisterOverlaying(void);
Input Parameters	NULL

Output Parameters	NULL
Description	It will Perform register overlaying.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get the resizer device handle. 2) Assign the resizer base address to the regs member of the device handle.

4.5.2 DDC_RSZInitailizeModule

Function	DDC_RSZInitializeModule
Function Prototype	Void * DDC_RSZInitializeModule (void);
Input Parameters	NULL
Output Parameters	Pointer to DDC layer device structure
Description	It will Initialize the semaphore for device structure and update the state of device.
Preconditions	None
Design	Logic in steps 1) Get the resizer device handle. 2) Initialize the semaphores and variables. 3) Call LLC_RSZSetExp 4) Return the resizer device handle

4.5.3 DDC_RSZRemoveModule

Function	DDC_RSZRemoveModule
Function Prototype	Void * DDC_RSZRemoveModule (void);
Input Parameters	NULL
Output Parameters	Pointer to DDC layer device structure
Description	It will update the update the state of device to RESIZER_DELETED.
Preconditions	None
Design	Logic in steps 1) Get the resizer device handle. 2) Change the status. 3) Return the resizer device handle

4.5.4 DDC_RSZSetDeviceInterruptNumber

Function	DDC_RSZSetDeviceInterruptNumber
Function Prototype	void DDC_RSZSetDeviceInterruptNumber();
Input Parameters	NULL
Output Parameters	NULL
Description	It will assign the interrupt number to the resizer driver.
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps 1) Get the resizer device handle. Assign the interrupt number to the IntNum member of the device handle.

4.5.5 DDC_RSZGetDeviceHandle

Function	DDC_RSZGetDeviceHandle
Function Prototype	Void * DDC_RSZgetDeviceHandle(void);
Input Parameters	NULL
Output Parameters	NULL
Description	It will return the handle for resizer.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Return the device object.

4.5.6 DDC_RSZGetChannelHandle

Function	DDC_RSZGetchannelHandle
Function Prototype	Void * DDC_RSZgetChannelHandle(int);
Input Parameters	Variable holding segment id for memory region.
Output Parameters	NULL

Description	It will return the Channel handle for resizer.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Return the channel specific object.

4.5.7 DDC_RSZOpenHandle

Function	DDC_RSZOpenHandle
Function Prototype	Void * DDC_RSZOpenHandle(int);
Input Parameters	Variable holding segment id for memory allocation.
Output Parameters	NULL
Description	It will return the Channel handle and update the state of resizer channel.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Call the function DDC_RSZGetDeviceHandle. 2) Increment the no of channels created 3) Initialize the channel configuration structure 4) Create the Channel semaphore 5) Update the Channel state to RESIZER_OPENED 6) Return the channel handle

4.5.8 DDC_RSZIoctl

Function	DDC_RSZIoctl

Function Prototype	PSP_Result DDC_RSZIoctl(PSP_Handle handle,PSP_AFIoctlCommand cmd,Ptr cmdArg,Ptr params);
Input Parameters	Handle – Handle of the Channel Cmd – Submit command to be passed CmdArg Argument to the command Ptr Params – Pointer to parameter structure
Output Parameters	Int according to the PSP errors
Description	It acts as wrapper, which provides the information as per command.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Check the Input Parameters 2) Check Channel is Closed or not. 3) Read the ioctl command 4) For setting parameters –call DDC_RSZValidateParameters and set the register using LLC Functions

4.5.9 DDC_RSZIsChannelHandleClosed

Function	DDC_RSZIsChannelHandleClosed
Function Prototype	Int DDC_RSZIsChannelHandleClosed (void * handle);
Input Parameters	Handle
Output Parameters	NULL
Description	It will return the status of the handle. If the handle is opened it will return DDC_RSZ_SOK else it will return DDC_VPSS_E_FAIL.
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps 1) Check the channel state 2) Return the status accordingly

4.5.10 DDC_RSZValidateParameters

Function	DDC_RSZValidateParameters
Function Prototype	Int DDC_RSZValidateParameters(DDC_RSZChanObject *, PSP_RSZparams *)
Input Parameters	Pointer to DDC_RSZChanObject Pointer to PSP_RSZparams
Output Parameters	NULL
Description	It will Validate the resizer channel parameters passed by the user.
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps 1) Calculate the phase and resize ratio and set the value in channel specific structure 2) Recalculate the horizontal and vertical resize ratio with the help of input size output size and phase passed by application and update the channel specific structure 3) If input line offset and output line offset is not 32 bit aligned then return –EINVAL 4) if output horz size is not less than 1280 for resize ratio 64 to 512 then return –EINVAL 5) if output_horz_size is not less than 640 for resize ratio 512 to 1024 then return –EINVAL 6) if output width is not 16 byte aligned for vertical upsizing then return _EINVAL 7) Set the resize ratio and phase in channel_config structure. 8) Recalculate the input horizontal and vertical size according to starting phase and output size passed by application and update the channel_config structure 9) Set the parameters input_vert_size,input_horz_size,input_line_offset,output_vert_size 10) Output_horz_size luma coefficients and horizontal and vertical coefficients from the values passed in rsz_params 11) Set the horizontal and vertical phase from the values passed in rsz_params

4.5.11 DDC_RSZGetParameters

Function	DDC_RSZGetParameters
Function Prototype	Int DDC_RSZGetParameters(DDC_RSZChanObject *, PSP_RSZparams *)
Input Parameters	Pointer to DDC_RSZChanObject Pointer to PSP_RSZparams

Output Parameters	NULL
Description	It will get the channel specific parameters.
Preconditions	Driver must be in opened state and channel handle should not be NULL and parameters must be configured
Design	Logic in steps 1) Get the resizer parameters from the channel specific structure.

4.5.12 DDC_RSZGetCropSize

Function	DDC_RSZGetCropSize
Function Prototype	Int DDC_RSZGetCropSize(DDC_RSZChanObject *, PSP_RSZcropsize *)
Input Parameters	Pointer to DDC_RSZChanObject Pointer to PSP_RSZcropsize
Output Parameters	NULL
Description	It will get the information regarding number of horizontal and vertical cropping pixels.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get the number of horizontal and vertical pixels to be cropped.

4.5.13 DDC_RSZGetPriority

Function	DDC_RSZGetPriority
Function Prototype	Int DDC_RSZGetPriority(DDC_RSZChanObject *, PSP_RSZPriority *)
Input Parameters	Pointer to DDC_RSZChanObject Pointer to PSP_RSZPriority
Output Parameters	NULL
Description	It will get the information regarding the priority of the particular channel.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get the priority of the particular channel from channel specific structure. 2) Return

4.5.14 DDC_RSZGetStatus

Function	DDC_RSZGetStatus
Function Prototype	Int DDC_RSZGetStatus(DDC_RSZChanObject *, PSP_RSZStatus *)
Input Parameters	Pointer to DDC_RSZChanObject Pointer to PSP_RSZStatus
Output Parameters	NULL
Description	It will get the information regarding the Status of the hardware and channel.

Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get the status of the channel from channel specific structure 2) Get the status of hardware by calling function LLC_RSZGetHWStatus 3) Get the input source

4.5.15 DDC_RSZSetPriority

Function	DDC_RSZSetPriority
Function Prototype	Int DDC_RSZSetPriority(DDC_RSZChanObject *, PSP_RSZPriority *)
Input Parameters	Pointer to DDC_RSZChanObject Pointer to PSP_RSZPriority
Output Parameters	NULL
Description	It will set the channel priority.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Check the that priority send my the user lie in the range 0 to 5 2) Set the priority in the channel specific structure

4.5.16 DDC_RSZStartResize

Function	DDC_RSZStartResize
Function Prototype	Int DDC_RSZStartResize(DDC_RSZChanObject *, PSP_RSZresize *)

Input Parameters	Pointer to DDC_RSZChanObject Pointer to PSP_RSZresize
Output Parameters	NULL
Description	This function enables resize bit after doing the hardware register configuration after which resizing will be carried on.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps : 1) Check the input and output size send by user 2) Assign input and output address to the respective fields of the channel specific structure. 3) Call DDC_RSZAddToArray 4) Call LLC_RSZHardwareSetup 5) Enable ISR 6) Call LLC_RSZResizerEnable 7) Wait on SemIsr 8) Call LLC_GetWriteBufferStatus 9) Call DDC_ DeleteFromArray 10) Return 0

4.5.17 DDC_RSZAddToArray

Function	DDC_RSZAddToArray
Function Prototype	Int DDC_RSZAddToArray(DDC_RSZChanObject *)
Input Parameters	Pointer to DDC_RSZChanObject
Output Parameters	NULL

Description	Function to add the current channel configuration into array according to priority.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps : 1) Lock on ArraySem 2) Add channel handle at appropriate position in global array of channel specific structure array ,depending on its priority 3) Increment ArrayCount of global device structure 4) If ArrayCount is not equal to 1than Unlock ArraySem andwait on channel specific mutex 5) Else Unlock ArraySem

4.5.18 DDC_RSZDeleteFromArray

Function	DDC_RSZDeleteFromArray
Function Prototype	Int DDC_RSZDeleteFromArray(DDC_RSZChanObject *)
Input Parameters	Pointer to DDC_RSZChanObject
Output Parameters	NULL
Description	Function to delete the processed array entry form the array
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps : 1) Lock on ArraySem 2) Remove the top most entry from global structure channel specific array. 3) Decrement ArrayCount 4) If count is not equal to 0 then Unlock ArraySem 5) post semaphore channel_config.channel_mutex for next array entry 6) Else Unlock Arraysem

4.5.19 DDC_RSZCloseHandle

Function	DDC_RSZCloseHandle
Function Prototype	Void * DDC_RSZCloseHandle(Ptr handle);
Input Parameters	Void * handle
Output Parameters	NULL
Description	It will return the Close the logical channel associated with the handle.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Call the function DDC_RSZGetDeviceHandle. 2) Decrement the no of channels created 3) Free the channel configuration structure 4) Delete the Channel semaphore 5) Update the Channel state to RESIZER_CLOSED

4.5.20 DDC_RSZisr

Function	DDC_RSZisr
Function Prototype	Int DDC_RSZisr();
Input Parameters	NULL
Output Parameters	Returns the 0 on sucess
Description	It will be called when Resizer completes processing of one frame.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Post the SemIsr of the global structure.

4.6 LLC Layer Functions

4.6.1 LLC_RSZHardwareSetup

Function	LLC_RSZHardwareSetup
Function Prototype	Int LLC_RSZHardwareSetup(LLC_ResizerConfigParams *,Ptr regs);
Input Parameters	Pointer to channel parameters structure Pointer to register overlaying structure
Output Parameters	Returns the 1 on sucess
Description	This Function will write all the parameters into the hardware registers except enable bit

Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Configure Resizer hardware registers from channel config parameter structure after doing appropriate masking.

4.6.2 LLC_RSZresizerEnable

Function	LLC_RSZresizerEnable
Function Prototype	Int LLC_RSZresizerEnable(Ptr Regs);
Input Parameters	Pointer to register overlaying structure
Output Parameters	Returns the 1 on sucess
Description	This Function will enable the resizer 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 1) Set enable bit into the resizer PCR register.

4.6.3 LLC_RSZGetHWStatus

Function	LLC_RSZGetHWStatus
Function Prototype	Int LLC_RSZGetHWStatus(Ptr regs);
Input Parameters	Pointer to register overlaying structure

Output Parameters	Returns the 1 on sucess
Description	This Function will return the hardware status .i e: busy or not busy.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get busy bit from the resizer PCR register.

4.6.4 LLC_RSZSetExp

Function	LLC_RSZSetExp
Function Prototype	Int LLC_RSZSetExp(int exp);
Input Parameters	User defined expression value to configure read request delay
Output Parameters	Returns the 1 on sucess
Description	This Function will set the delay time for read request.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Set RESZ_EXP bit from the SDR_REQ_EXP register.

4.6.5 LLC_GetRSZWriteBufferStatus

Function	LLC_RSZGetWriteBuferStatus

Function Prototype	Int LLC_GetWriteBufferStatus();
Input Parameters	None
Output Parameters	Returns the write buffer status
Description	This Function is used to check the write buffer overflow bit.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1)Get RESZ_WBL bits from the VPSS_PCR register.

5 Decision Analysis & Resolution

5.1 DAR Criteria

The list of DAR Criteria is as follows:

- Complexity of design
- Interface Simplicity

5.2 Available Alternatives

5.2.1 Alternative 1

Resizer Driver will recalculate Resize ratio as per input-output size and starting phase.

5.2.2 Alternative 2

Resizer Driver will recalculate input size as per resize ratio, output size and starting phase

5.3 Decision

Since input size and output size are passed from the application it is more feasible to recalculate the resize ratio accordingly.

6 Revision History

Version #	Date	Author Name	Revision History
Draft 1.01	10 OCT 2006	EI3	Initial Draft Created
Draft 1.02	13 OCT 2006	EI3	Updated for technical & QA review Comments
Issue 1.00	13 OCT 2006	EI3	Issued to TII
Issue 1.01	14 NOV 2006	EI3	Added 1) Changed arguments of open function in all layers 2) Added states in resizer channel status enumeration 3) Added LLC_RSZSetExp and LLC_GetRszWriteBufer functions 4) Changed prototypes of all LLC level functions. 5) Added semaphore in port structure and segment id in device structure 6) Moved resizer channel config structure from DDC to LLC layer
Pre-silicon Release 0.3.0	20 NOV 2006	EI3	Release to TI
Post-silicon Release 0.3.0	29 NOV 2006	EI3	Release to TI
1.00.01	June 22, 2007	Anuj Aggarwal	GA Patch Release
1.00.02	June 29, 2007	Amit Chatterjee	Modified Release Version
1.003	July 18, 2007	EI3	Modified Release Version

1.00.04	November 29, 2007	Sivaraj R	PSP merge package changes - directory structure changes