TII DM6437 VPSS Drivers

Histogram Design Specifications

TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

This document provides details of the Histogram module driver design on DM6437.

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.

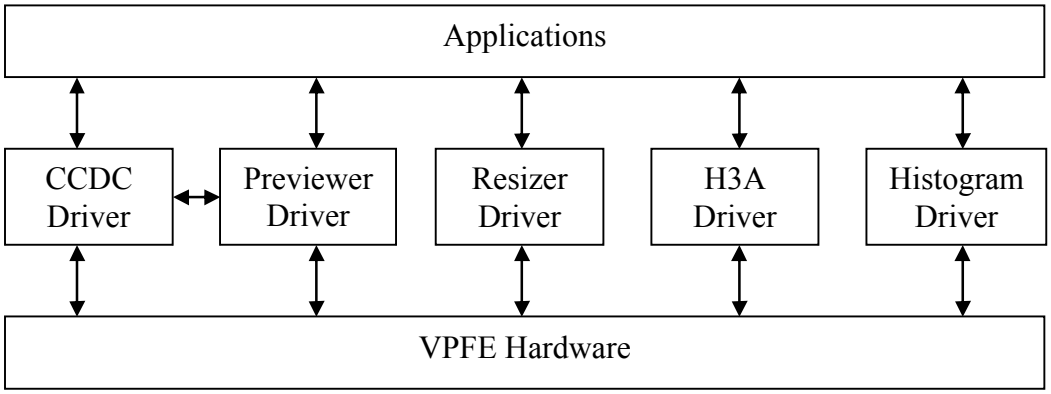

Figure 1. VPFE Driver Stack

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.

1.2 Terms & Abbreviations

AF	Auto Focus
AEW	Auto Exposure & Auto White Balance
CCDC	Charged Couple Device Controller
DDR	DD Random Access Memory
FIFO	First in first out
H3A	Hardware Bases AF & AEW Modules
HW	Hardware

MADUs	Memory data units
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 Version 1.01 dated OCT 07, 2006

1.4 Overview

The histogram accepts RAW image/video data from the video port interface of the CCD Controller, performs a color-separate gain on each pixel (white/channel balance), and bins them according to the amplitude, color, and region which are all specified via its register settings. It supports 4 color (Bayer pattern), and up to 4 regions simultaneously. The figure below shows the processing flow of the Histogram module.

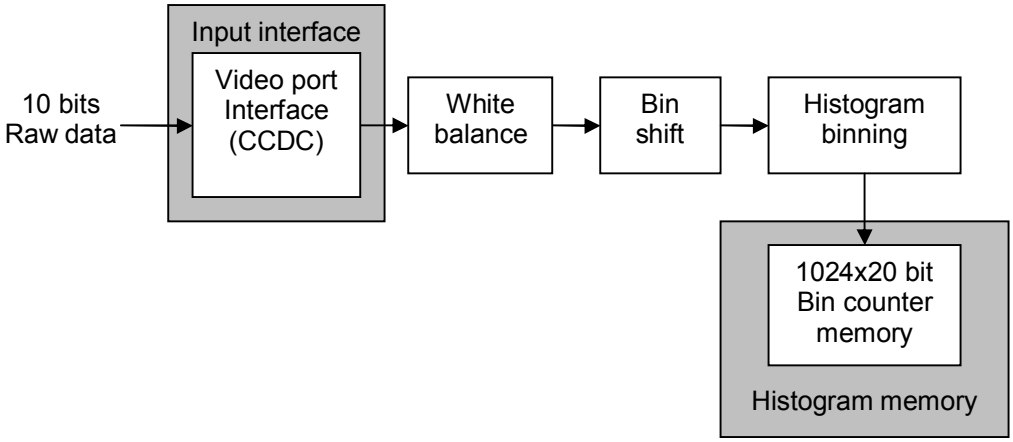

Figure 2. Block diagram of Histogram module

- **Input interface** – Histogram receives the raw image/video data from the video port (CCDC). Input data is 10bits wide.
- **White balance** – Histogram provides separate white balance gain settings for each color.
- **Histogram binning** – The Histogram bins the input data by amplitude, color, and region. Each bin is a counter, counting the number of pixels of a color in the range associated with the bin. The number of bins can be 32, 64, 128, or 256 bins. However, due to the limited Histogram memory size (1024 words), the number of bins (times 4 colors) limits the number of regions that can be active. Up to four regions can be active at any time but a pixel is only binned into one region. The priority is Region 0 > Region 1 > Region 2 > Region 3. In case of overlapping regions the statistics would be accumulated in only 1 region which is of higher priority.
- **Bin shift** - The number of least significant input bits which are used for binning is equal to $\log_2$ (Number of Bins). If the input bit width is larger than this, then the higher bits will be clipped to the highest bin location. This allows data from above the bin range to be included in the upper-most bin. Right shifting the input data basically increases the range of input values that will be sent to a

single bin. The number of input pixel values stored in a single bin is equal to 2^{SHIFT}.

- **Histogram memory -** The Histogram output memory contains 1024 20-bit words. The data can be written or read using the histogram memory specific registers.

2 Requirements

The histogram module driver Functional requirements are as follows:

- Driver shall support input from CCDC.
- Driver shall support output to DDR.
- Driver shall support configuration of White Balance Gain.
- Driver shall support Histogram Binning for multiple regions.
- Histogram Driver shall add time stamp to the captured statistics.
- Histogram Driver shall manage the buffers using Queue/Dequeue mechanism. It should automatically enable HW engine when the first buffer is enqueued.

2.1 Assumptions

- Histogram module always returns statistics from the base address of its output memory (i.e. it will return statistics for all active regions).

2.2 Constraints

- A region dimension of 1 (horizontal or vertical or both) is not allowed.
- Bin counter will be saturated to maximum value if its value exceeds the maximum value, which is $2^{20} - 1$.
- If N buffers are enqueued then only, N-1 buffer can be dequed.
- In case of overlapping regions the statistics would be accumulated in only 1 region which is of higher priority.

3 Design Description

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

The sub-section describes the interaction between different layers and functionalities of the Histogram driver.

The various functionalities of Histogram driver are as follows:

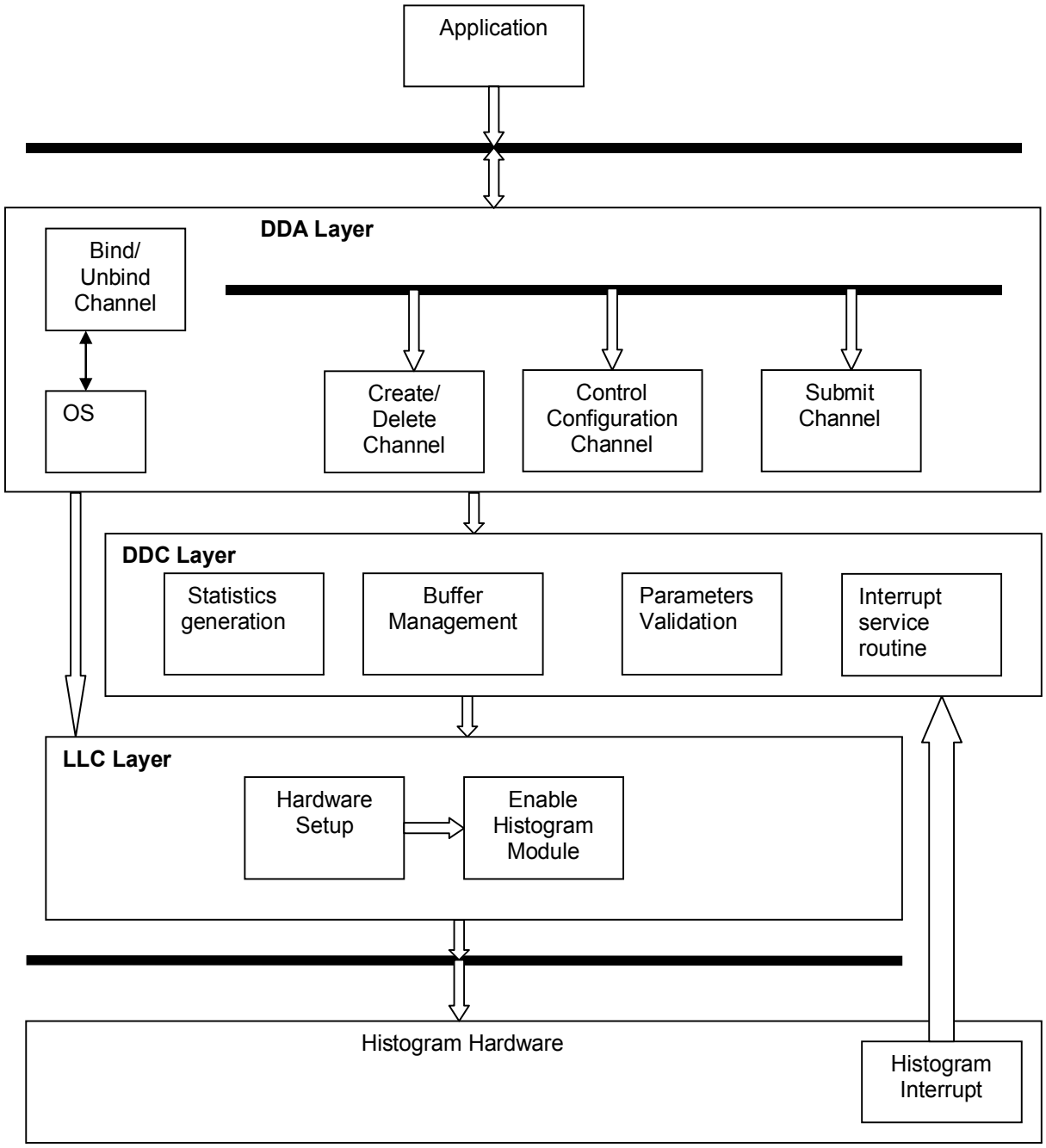

Figure 3. Detailed Design Diagram for Histogram

DDA (IOM) layer

- Bind/Unbind
- Create/Delete
- Various controls to configure HW
- Reading of statistics

DDC Layer

- Buffer management
- Parameters Validation
- Interrupt Service Routine

LLC Layer

- Hardware setup
- Enabling histogram module

3.1.2.1 Driver Creation/deletion

Driver Creation

The sequence diagram below depicts the creation phase of the BIOS Histogram driver. While at the DDC level, create phases of driver instance are clearly demarcated, the same is not the case in IOM and above. Regardless, once this phase is complete, the basic driver data structures and setups are complete and ready for formally opening device to perform IO.

User is expected to invoke HIST_mdBindDev(), way up in the application startup phase, perhaps in a central driver initialization function.

The HIST_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.

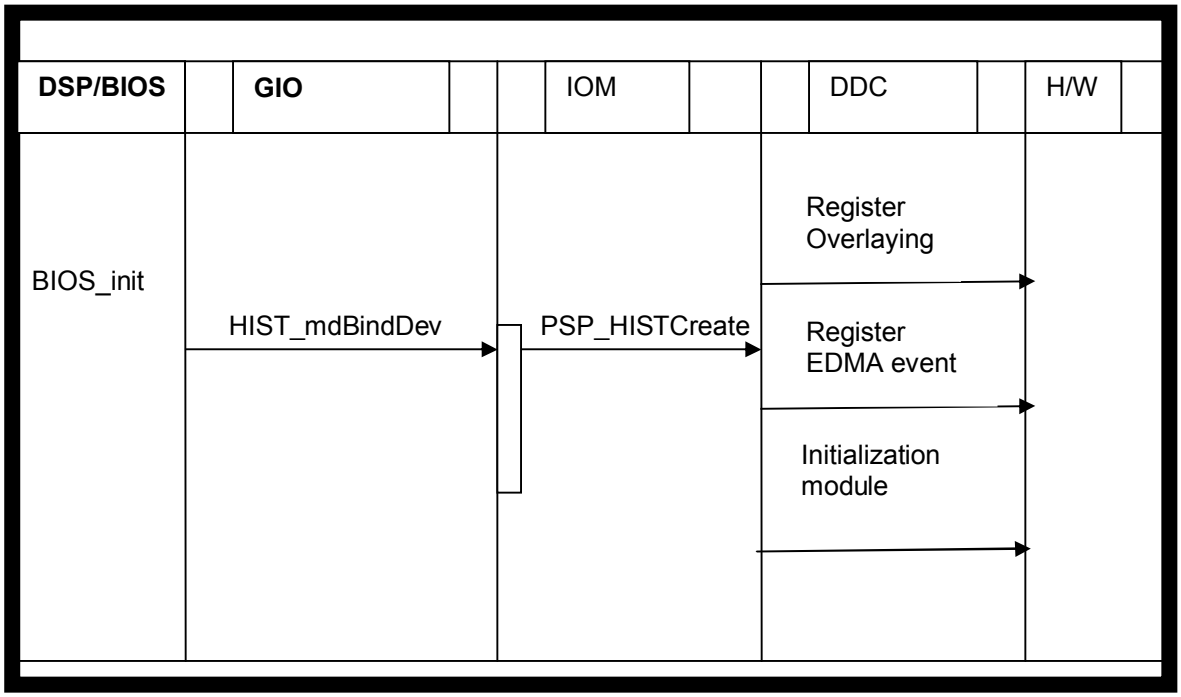

Figure 4. Driver Creation overview

HIST_mdBindDev

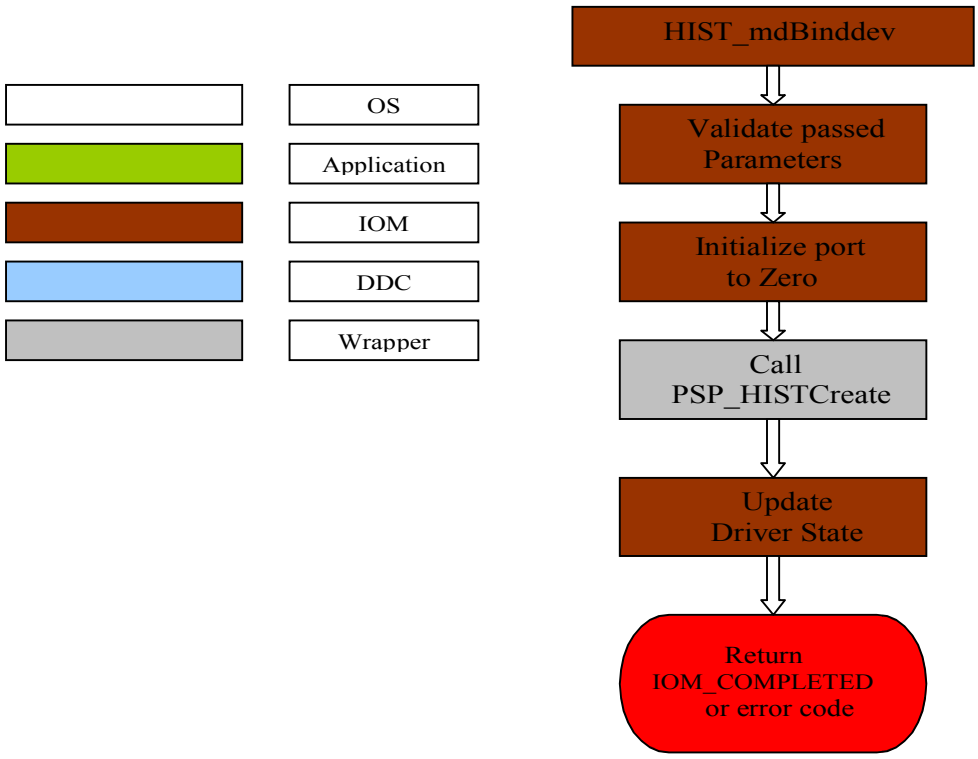

Figure 5. Driver Creation detail flow diagram-1

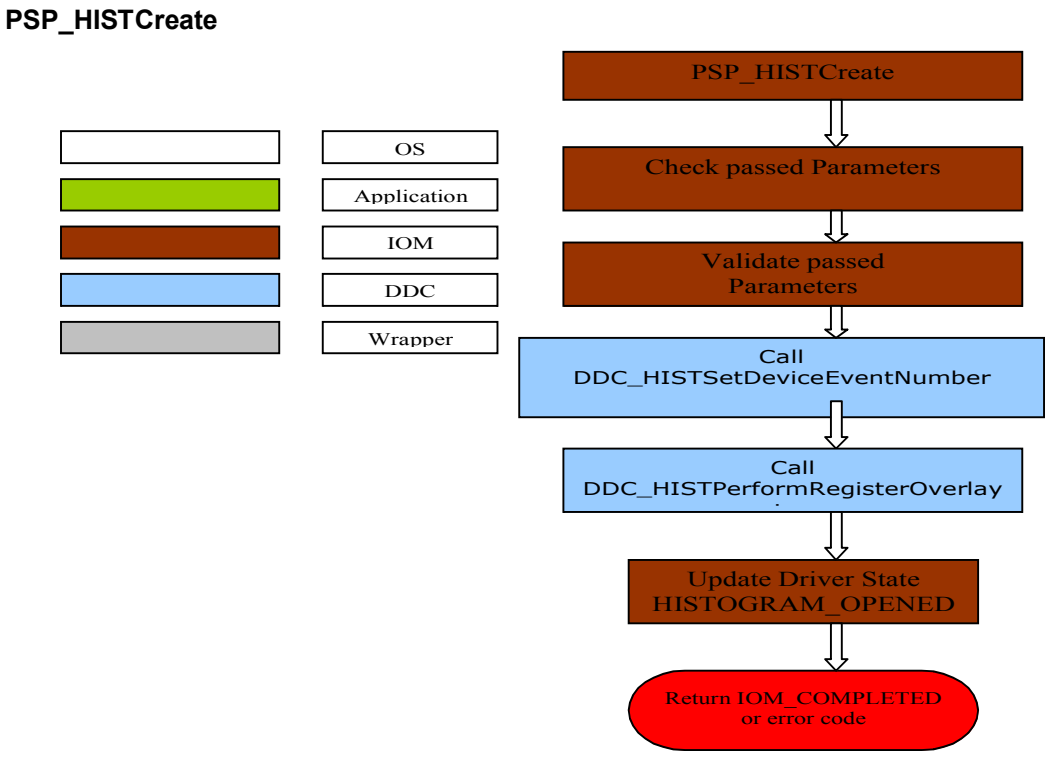

Figure 6. Driver Creation Detail flow diagram -2

Driver Deletion

Following the call HIST_mdUnBindDev() one is required to restart from beginning over a HIST_mdBindDev () call to bring driver back to life. The driver de-initialize and delete function de-initialize the Histogram DDC and delete if any OS resources originally allocated through HIST_mdBindDev ().

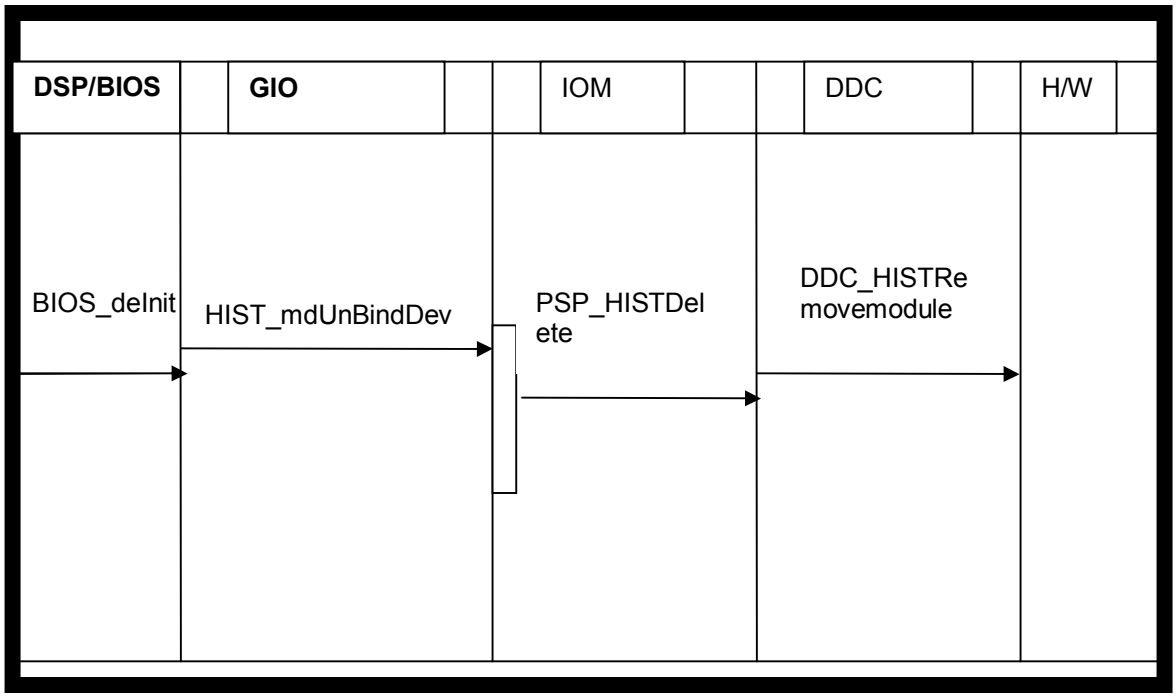

Figure 7. Driver deletion overview

HIST_mdUnBindDev

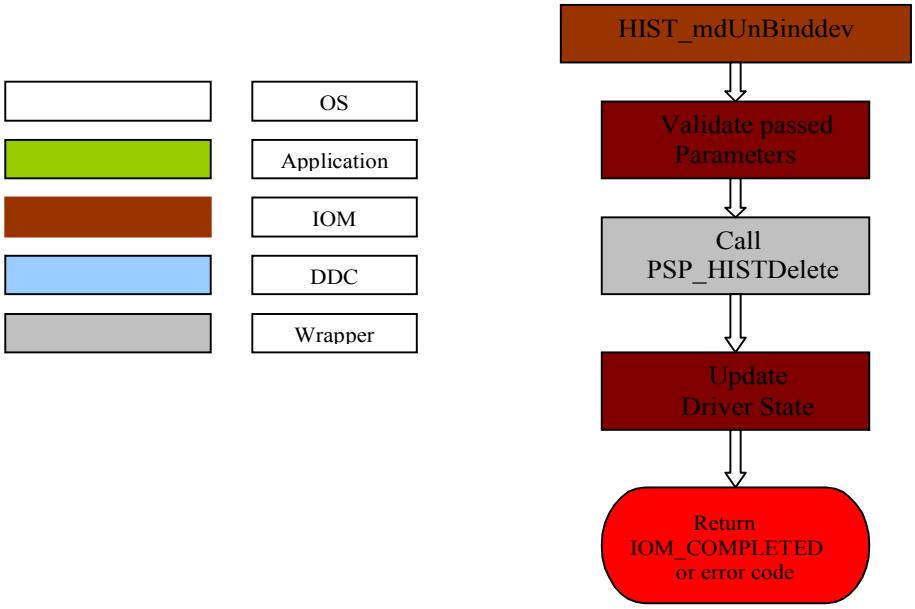

Figure 8. Driver Deletion detail Flow -1

PSP_HISTDelete

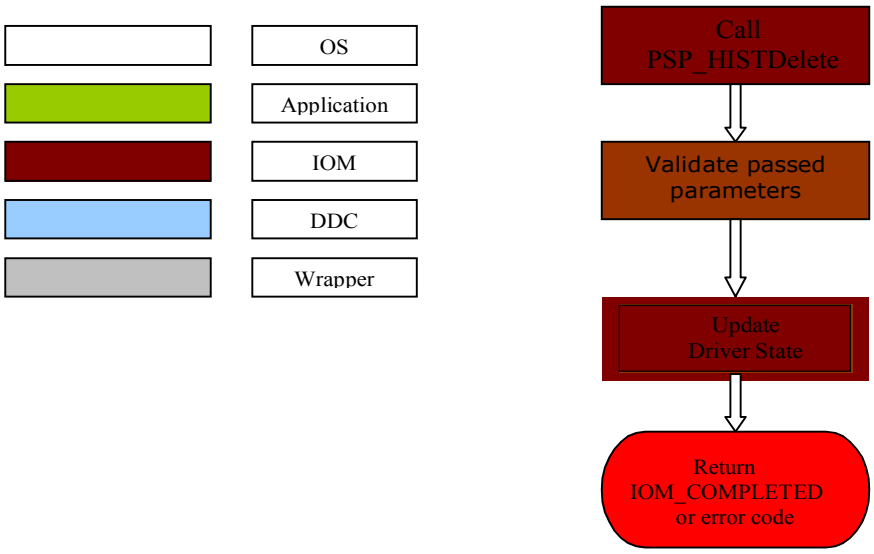

Figure 9. Driver deletion detail flow-2

3.1.2.2 Driver Open/Close

Driver Open

When the application calls the HIST_mdCreateChan ( ), driver entry point is created. The callback is registered. The device interrupts are enabled and driver is ready to accept Read/Write jobs.

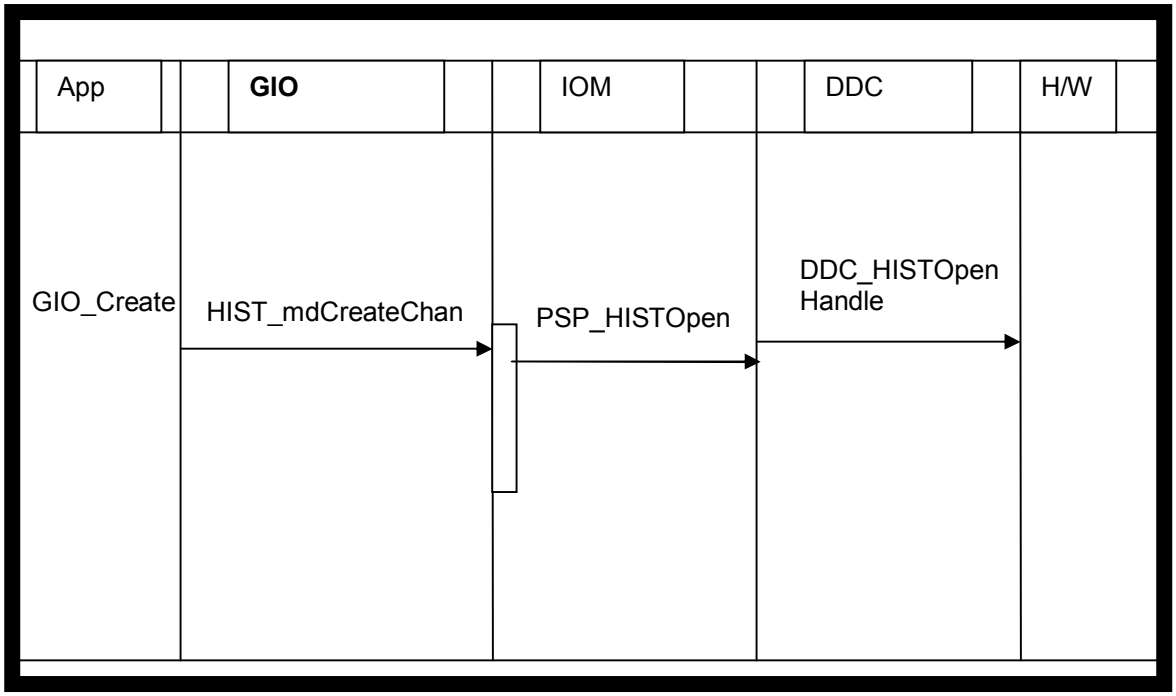

Figure 10. Driver create channel overview

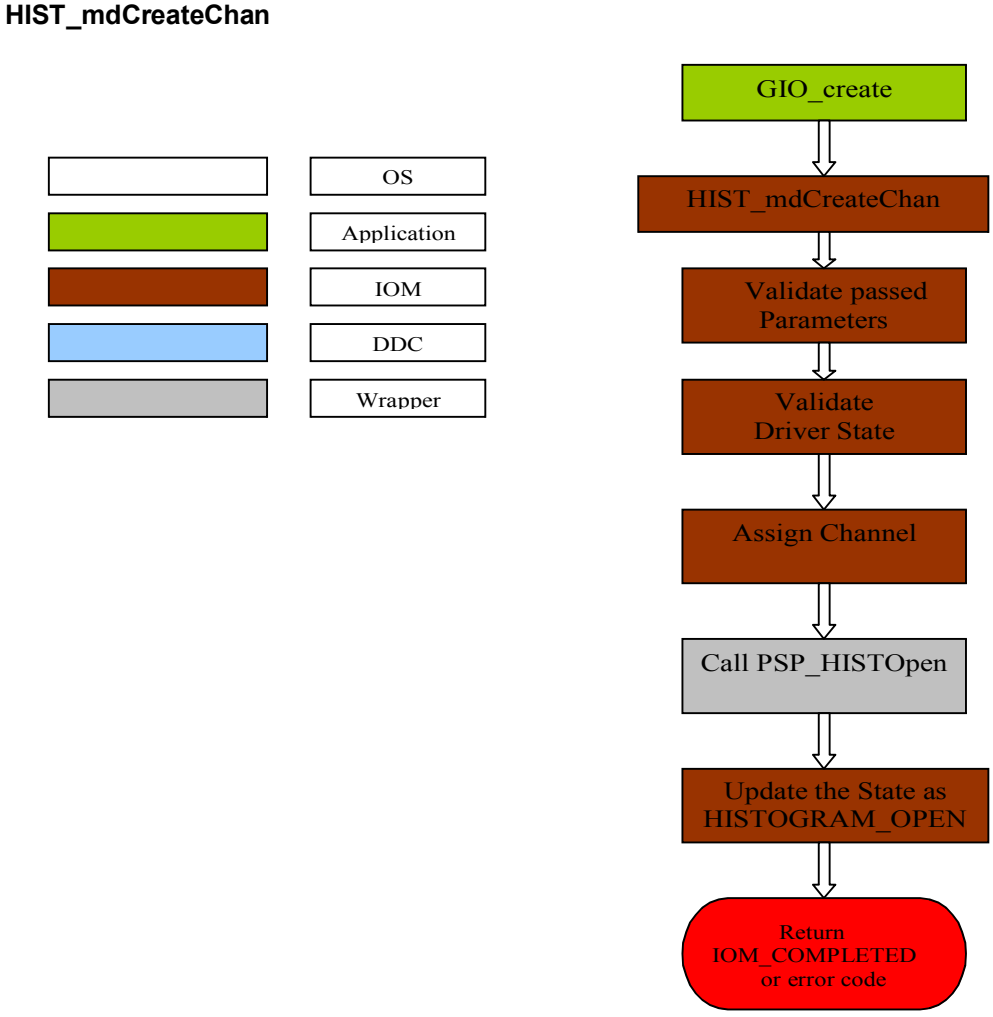

Figure 11. Driver Open Detail flow-1

PSP_HISTOpen

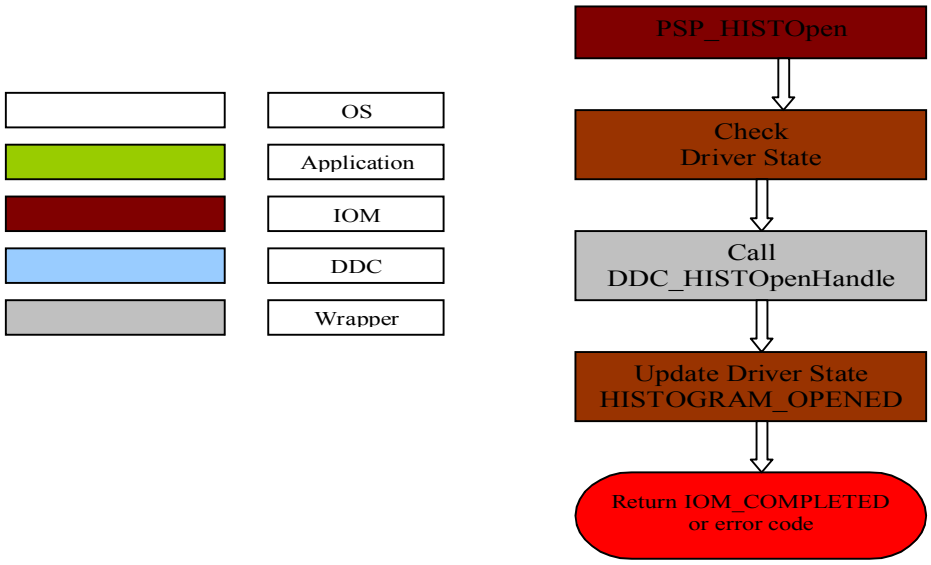

Figure 12. Driver Open detail flow -2

DDC_HISTOpenHandle

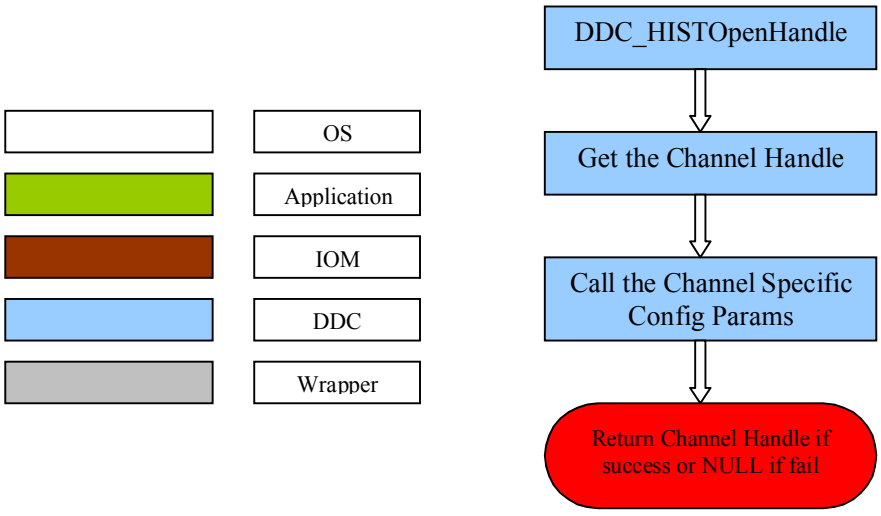

Figure 13. Driver Open detail flow -3

Driver Close

The application invokes the HIST_mdDeleteDev () function to close the channel of the Histogram device. Once the channel is closed it has no life. The user will have to bring the driver back to life by creating the driver through HIST_mdCreateChan ().

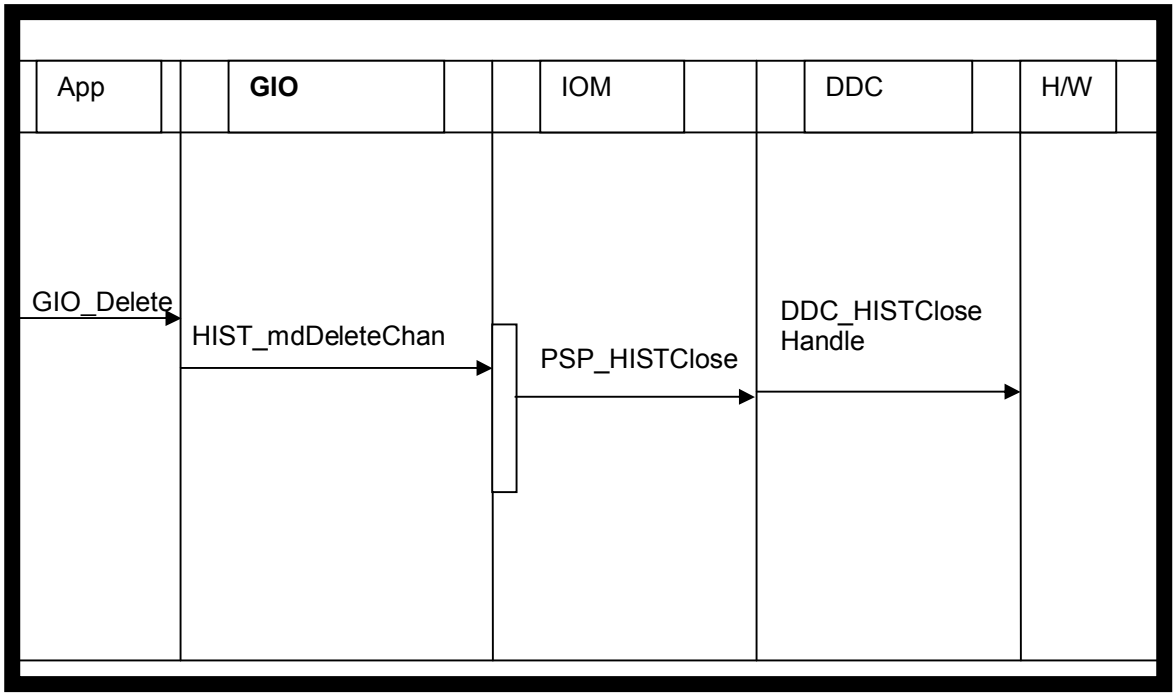

Figure 14. Driver channel close overview

HIST_mdDeleteChan

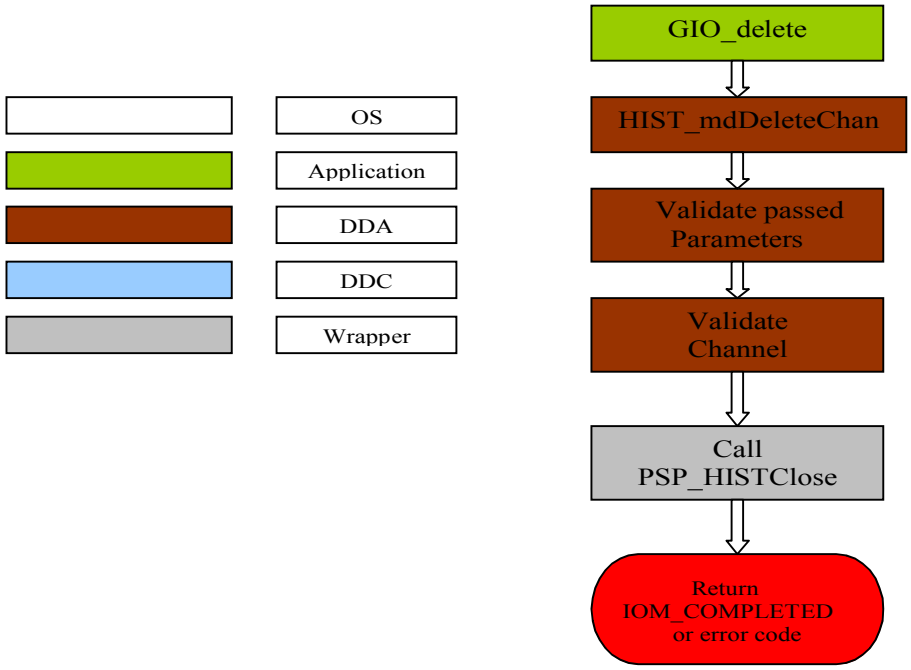

Figure 15. Driver close channel detail flow -1

PSP_HISTClose

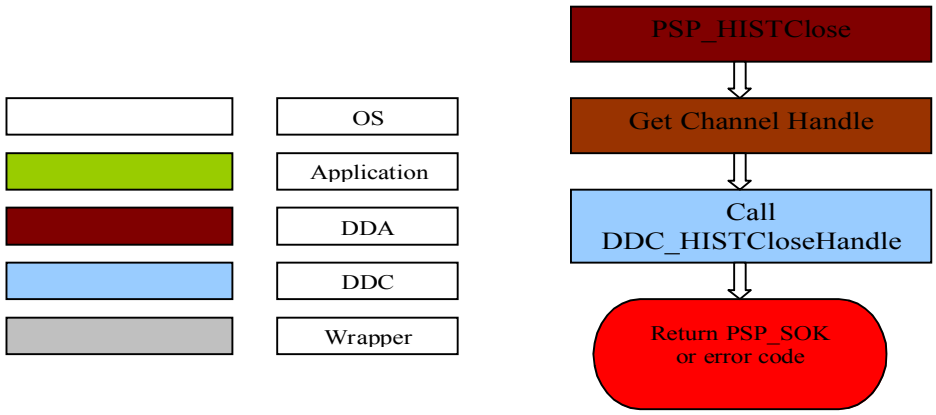

Figure 16. Driver close channel detail flow-2

DDC_HISTCloseHandle

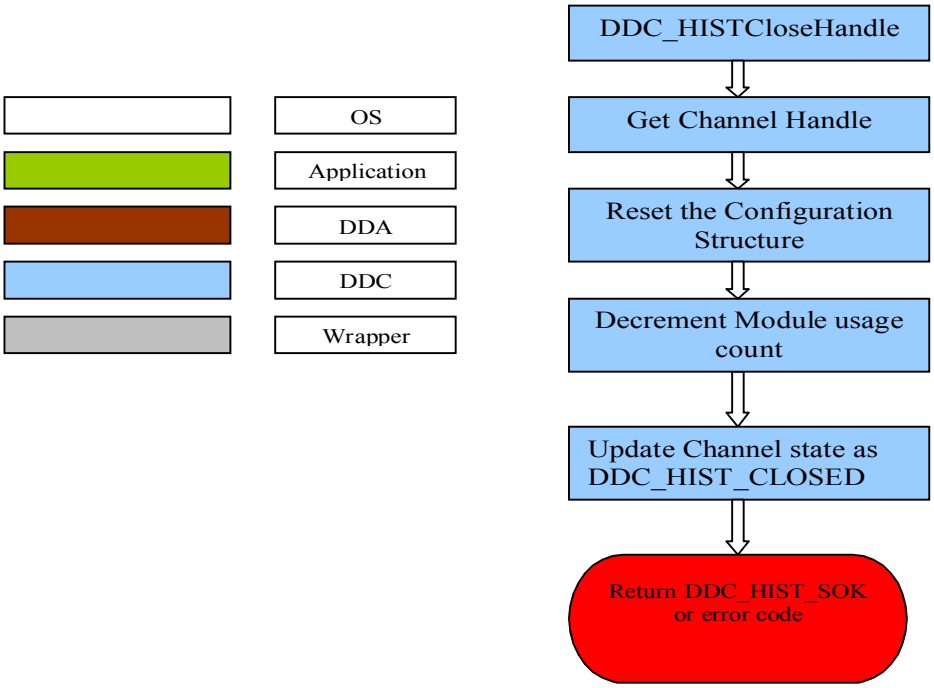

Figure 17. Driver close Channel detail flow -3

3.1.2.3 Various Controls

The Histogram river provides HIST_mdControlChan ( ) to set/get common configuration parameters on the driver at run time through the corresponding DDC IOCTL function, ddc_HISTIoctl ( ). Moreover IOCTL commands that are device specific or that require action on the part of the device driver call the driver's IOCTL.

Following is the flow diagram for the above functionality.

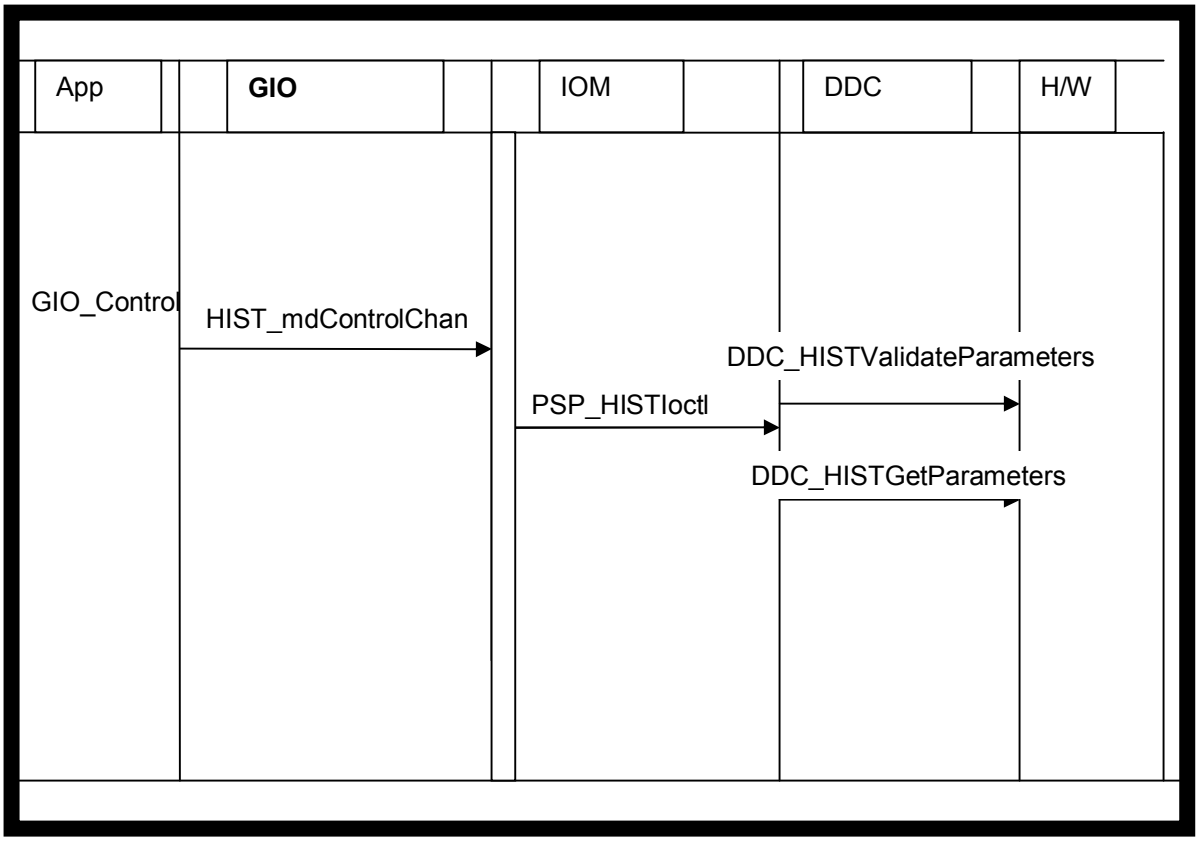

Figure 18. Control Command overview

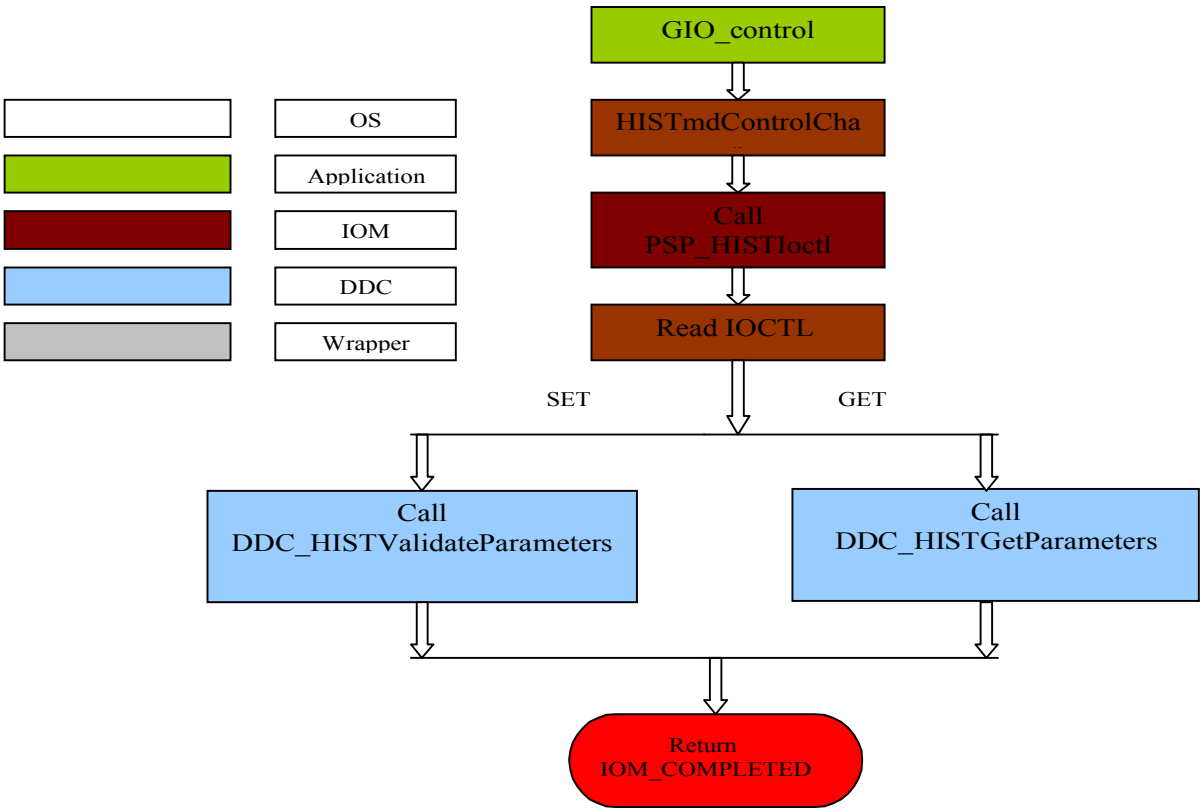

Figure 19. Control Command detail flow-1

3.1.2.4 IO access

The Application will access Histogram driver HIST_mdSubmitChan through interface function from DSP/BIOS. These functions are registered on the GIO Layer during the driver initialization. The DDC will maintain two frame buffers queues with a defined size of the buffers. Once the buffer is prepared, application will issue a "QUEUE" call to the driver and that buffer is queued in the Active Queue. At some point in time, when the H/W interrupt is asserted, the ISR checks for any new frame buffer in the ACTIVE buffer and if there is any, driver puts the last captured buffer in the FREE queue and updates the address of the new queued frame buffer on the DMA Address.

Application specific callback functions shall be invoked from the Interrupt context. The callback function will be registered with the driver object.

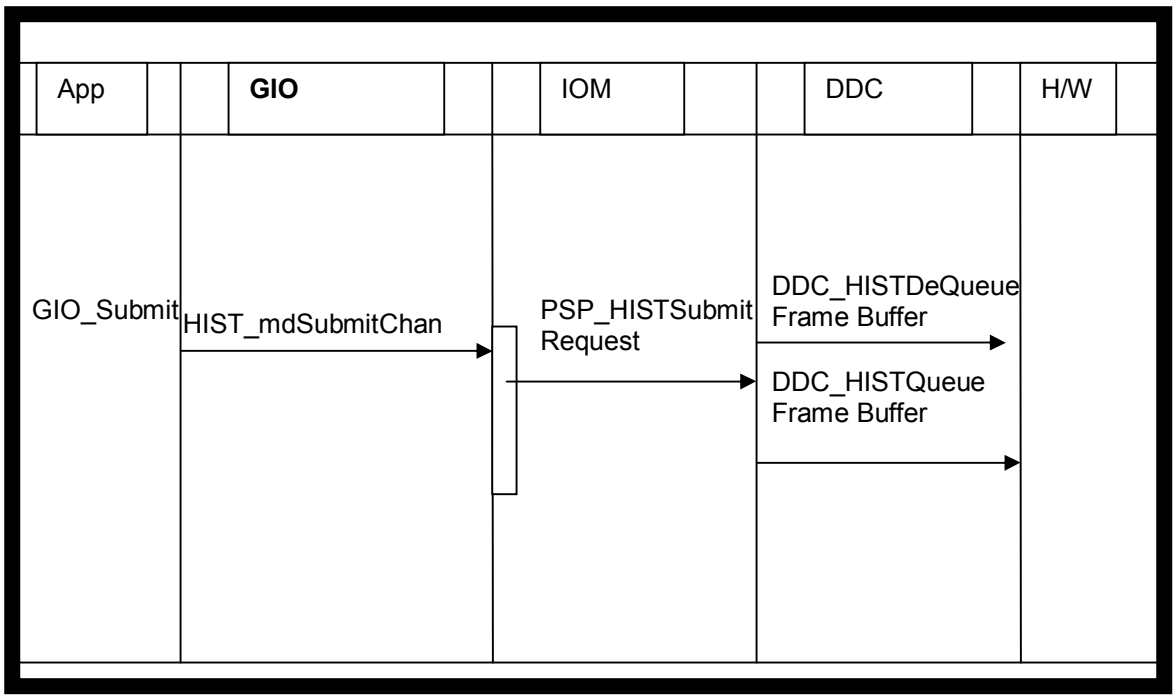

Figure 20. Driver Submit Overview

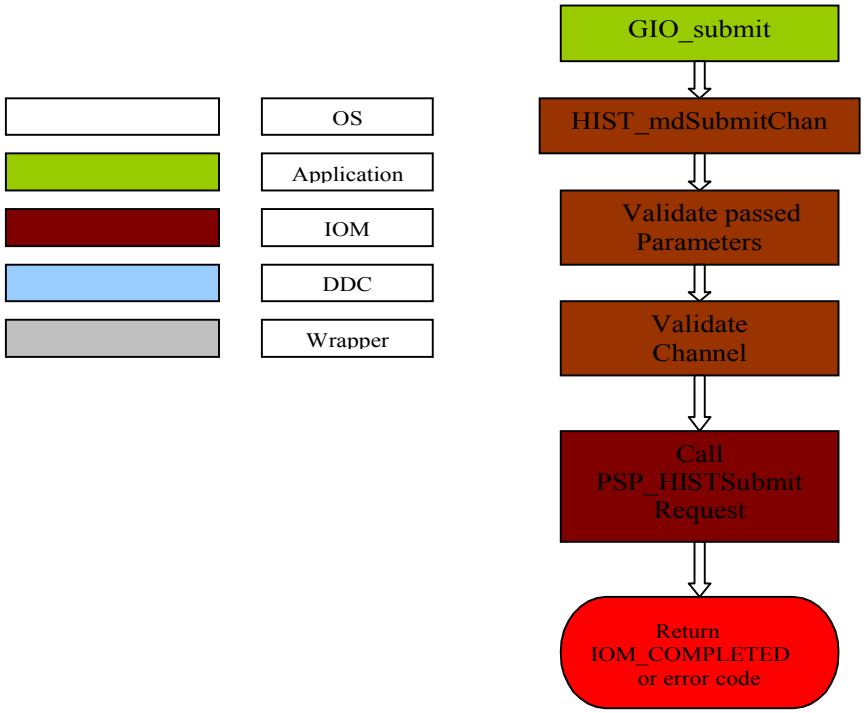

Figure 21. Driver Submit Detailed Flow Diagram – 1

PSP_HISTSubmitRequest

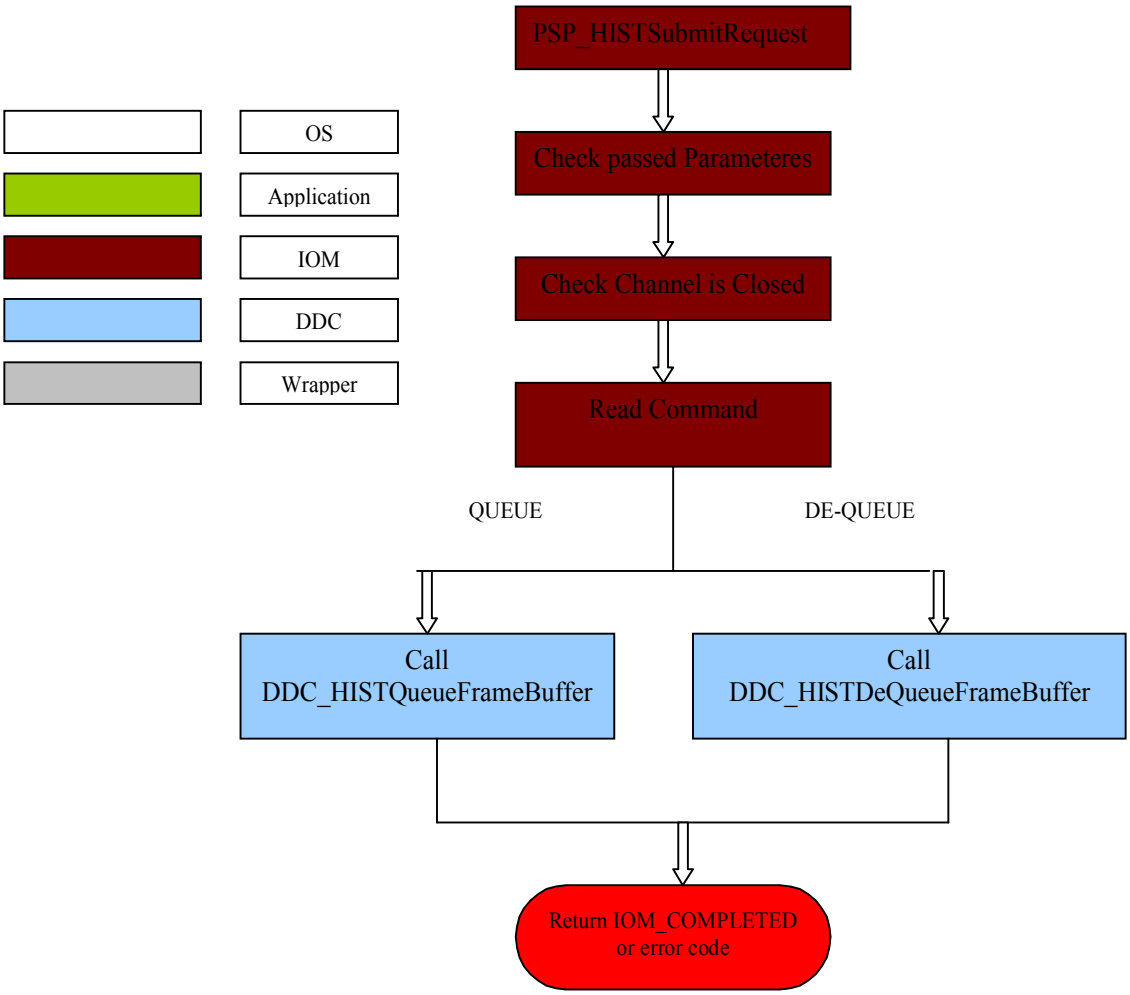

Figure 22. Driver Submit Detailed Flow Diagram – 2

Interrupts

The interrupt is generated when frame is completed.

DDC Layer Functions

3.1.2.5 Buffer management

When application puts a first buffer in a queue, the driver will automatically enabled & start capturing frame. After completion of a frame the driver will copy the data into queued buffer. It is assumed that application will always keep one buffer in queue. When last buffer will be there in queue, it will be overwritten for each frame. Enqueue/Dequeue will follow the FIFO mechanism.

3.1.2.6 Edma Service Routine

Histogram driver will implement Edma transfer in sync with events generated after the completion of a frame. After the completion of the transfer the ISR will be called. ISR extract buffer from the enqueue & insert in the dequeue. It will also reconfigure the Edma for the next available enqueued buffer. If only one buffer is there in queue then Edma will be reconfigured for that buffer.

3.1.2.7 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

4.1 Constants & Enumerations

Constants

4.1.1 Region's maximum position

Defines the maximum allowed position for region boundaries.

Definition

```
#define PSP_HIST_REGION_MAX_HPOS 16384u
#define PSP_HIST_REGION_MAX_VPOS 16384u
```

Comments

None

Constraints

None

4.1.2 Maximum device instance

Maximum device instance that can be created

Definition

```
#define PSP_HIST_NUM_INSTANCES      1u
```

Comments

None

Constraints

None

Interface layer Enumerations

4.1.3 Clear output data

Enumeration to indicate the status for data clearing after read.

Definition

```
typedef enum _PSP_HISTclearopdata
{
      PSP_HIST_NO_CLEAR_AFTER_READ = 0,
      PSP_HIST_CLEAR_AFTER_READ
} PSP_HISTclearopdata;
```

Comments

None

Constraints

None

4.1.4 Histogram binning

It describes the possible values for binning.

Definition

```
typedef enum _PSP_HISTbins
{
      PSP_HIST_32BINS = 0,
      PSP_HIST_64BINS,
      PSP_HIST_128BINS,
      PSP_HIST_256BINS
} PSP_HISTbins;
```

Comments

None

Constraints

None

4.1.5 Right shift

Describe the possible right shift values before binning.

Definition

```
typedef enum _PSP_HISTdatarightshift
{
      PSP_HIST_DATA_RSHIFT_ZERO = 0,
      PSP_HIST_DATA_RSHIFT_ONE,
      PSP_HIST_DATA_RSHIFT_TWO,
      PSP_HIST_DATA_RSHIFT_THREE,
      PSP_HIST_DATA_RSHIFT_FOUR,
      PSP_HIST_DATA_RSHIFT_FIVE,
      PSP_HIST_DATA_RSHIFT_SIX,
      PSP_HIST_DATA_RSHIFT_SEVEN
} PSP_HISTdatarightshift;
```

Comments

None

Constraints

None

4.1.6 Control commands

This enum describes various control commands.

Definition

```
typedef enum PSP_HISTIoctlCmd
{
    PSP_HIST_IOCTL_SET_PARAMS = IOM_USER,
    PSP_HIST_IOCTL_GET_PARAMS
} PSP_HISTIoctlCmd;
```

Comments

None

Constraints

None

DDC layer Enumerations

4.1.7 Histogram device state

It describes the status of histogram device.

Definition

```
typedef enum _DDC_HISTOGRAM_STATE
{
      DDC_HIST_DELETED = 0,
      DDC_HIST_CREATED,
      DDC_HIST_OPENED,
      DDC_HIST_CLOSED
} DDC_HISTOGRAM_STATE;
```

Comments

None

Constraints

None

4.1.8 Parameter configuration state

It describes the status of parameter configuration.

Definition

```
typedef enum _DDC_HISTParams_config_status
{
      DDC_STATE_CONFIGURED = 0,
      DDC_STATE_NOT_CONFIGURED
} DDC_HISTParams_config_status;
```

Comments

None

Constraints

None

4.1.9 Histogram device channel state

It describes the channel state.

Definition

```
typedef enum _DDC_HISTchannelstate
{
      DDC_HIST_CHAN_CLOSED = 0,
      DDC_HIST_CHAN_OPENED
} DDC_HISTchannelstate;
```

Comments

None

Constraints

None

4.1.10 Histogram error codes

It describes the error codes related to histogram

Definition

```
typedef enum _DDC_HISTstatus
{
      DDC_HIST_E_FAIL = 0,
      DDC_HIST_SOK,
      DDC_HIST_INVALID_STATE,
      DDC_HIST_INVALID_PARAM,
      DDC_HIST_NOT_SUPPORTED,
      DDC_HIST_NO_DATA
} DDC_HISTstatus;
```

Comments

None

Constraints

None

4.2 Data Structures

Interface layer structures

4.2.1 Histogram parameters

This structure is used to configure the various histogram parameters.

Definition

```
typedef struct _PSP_HISTparam
{
      PSP_HISTclearopdata     clearOpDataAfterRead;
      PSP_HISTbins            bins;
      PSP_HISTdatarightshift  shift;

      Uint8                   wbGain[4];

      PSP_HISTregionboundries region[4];

} PSP_HISTparam;
```

Fields

clearOpDataAfterRead	clearing of Histogram output data after read
Bins	bins per Histogram
Shift	right shift of data for Histogram binning
wbGain[4]	white balance gain for all colors
region[4]	Region boundaries for all regions

Comments

All parameters should be configured before statistical analysis begins.

Constraints

None

4.2.2 Region Boundaries

It describes the horizontal & vertical boundaries for the regions

Definition

```
typedef struct _PSP_HISTregionboundries
{
    Uint16     hStart;
    Uint16     hEnd;
    Uint16     vStart;
    Uint16     vEnd;
} PSP_HISTregionboundries;
```

Fields

hStart	Horizontal start information
hEnd	Horizontal end information

vStart	Vertical start information
vEnd	Vertical end information

4.2.3 Histogram buffer

This structure contains buffer pointer & timestamp information.

Definition

```
typedef struct _PSP_HISTbuffer
{
      PAL_OsListNodeHeader  hNode;
      Ptr                   ramIpAddr;
      Uint32                timeStmp;
      Uint32                size;
} PSP_HISTbuffer;
```

Fields

hNode	Node for queue entry
ramIpAddr	Buffer address to store data
timeStmp	Tells the time(Milliseconds) when data was read from histogram
size	Size of the enqueued buffer

Comments

This structure will be passed as a parameter at the time of enqueue/dequeue operation.

Constraints

None

DDC level structures

4.2.4 DDC Layer Histogram memory management buffer

It contains parameters related to enqueue/dequeue buffer management.

Definition

```
typedef struct _DDC_HISTbufferobject
{
      Uint       IndexEnQueue;
      Uint       IndexDeQueue;
      PAL_OsListNodeHeader    enQueue;
      PAL_OsListNodeHeader    deQueue;
} DDC_HISTbufferobject;
```

Fields

IndexEnQueue	Index indicating the number of buffer pushed on the enqueue
IndexDeQueue	Index indicating the number of buffers containing statistics on dequeue
enQueue	Queue head which keeps track of the buffer sent by the application
deQueue	Queue head which keeps track of the buffer containing statistics

Comments

None

Constraints

None

4.2.5 DDC Layer Histogram device object structure

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

Definition

```
typedef struct _DDC_HISTdeviceobject
{
      Uint                    histInstanceId;
      HISTOGRAM_STATE         stateDVC;
      Uint                    edmaChannelNo;
      Ptr                     regs
      DDC_HISTchannelobject   histChannelObj;
} DDC_HISTdeviceobject;
```

Fields

histInstanceId	Describe histogram device ID.
stateDVC	Describes histogram device state.
edmaChannelNo	Describes edma channel number.
regs	Handle for histogram register overlaying structure
histChannelObj	Handle for Histogram channel object

Comments

None

Constraints

None

4.2.6 DDC Layer Histogram channel object structure

This structure is contains parameter related to histogram channel.

Definition

```
typedef struct _DDC_HISTchannelobject
{
      DDC_HISTchannelstate              stateCH;
      PSP_HISTparam                     reg_param;
      DDC_HISTParams_config_status      paramConfigState;
      DDC_HISTbufferobject              bufferObj;
      PAL_OsSemHandle                   channelSem;
      Ptr                               edmaHandle;
      Uint32                            tcc;
      PAL_OsSemHandle                   semIsr;
} DDC_HISTchannelobject;
```

Fields

stateCH	Indicates the status of channel
reg_param	Instance to configuration parameters
paramConfigState	State of parameter configuration
bufferObj	Pointer to buffer management structure at DDC level
channelSem	Handle to blocking semaphore for channel
edmaHandle	Handle to EDMA device
tcc	EDMA transmission interrupt number
semIsr	Handle to blocking semaphore for dequeue call

Comments

None

Constraints

None

IOM level structures

4.2.7 IOM Layer Histogram port information

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

Definition

```
typedef struct HISTPORTOBJ
{
```

```
    Uns                         histPortNumber;
    HIST_iomchannelobj          iomChannelObj;
    DDC_HISTdeviceobject        *deviceObj;
    PSP_HISTOGRAM_STATE         state;
} HISTPortObject;
```

Fields

histPortNumber	Histogram port number
iomChannelObj	histogram channel object
deviceObj	Pointer to histogram device object
state	State of histogram driver.

Comments

None

Constraints

None

4.2.8 IOM Layer Histogram Channel Handle

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

Definition

```
typedef struct HISTIOMCHANNELOBJ
{
    HISTportinfo            *port;
    PSP_Handle              ddc_Handle;
} HIST_iomchannelobj;
```

Fields

Port	Reverse pointer to Port
ddc_Handle	Handle to Histogram channel

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	Stirng	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 "/histogram".

Int	mode

The mode argument specifies the mode in which the device is to be opened. Histogram driver will support only IOM_INPUT mode.

Ptr	chanParams

The chanParams parameter is a pointer that may be used to pass device specific arguments to the mini-driver. For histogram module it will be a handle to EDMA device.

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 channel open will not be supported by the Histogram 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. Application must pass EDMA device handle.

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.

4.3.4 GIO_SUBMIT

Syntax

```
int GIO_submit(GIO_Handle gioChan, Uns cmd, Ptr bufp, Uns *pSize,
GIO_AppCallback      *appCallback);
```

Arguments

IN GIO_Handle gioChan

The gioChan parameter is the handle returned by GIO_create.

Uns cmd

Specified mini-driver command to perform functionality.

Ptr Bufp

Pointer to data structure which contains data buffer pointer & time stamp parameter.

OUT GIO_AppCallback* appCallback

The appCallback parameter point to either a callback structure that contains the callback function to be called when the request completes is passed, or NULL which causes the call to be synchronous.

IN OUT Uns* pSize

The pSize parameter points to the size of the buffer. When it returns it will point to the number of MADUs transferred to or from the device.

Return Value

int IOM_COMPLETED on success and negative value if error.

Comments

An application calls GIO_submit to enqueue/dequeue data buffer.

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.

4.4 DDA Layer Functions

4.4.1 HIST_mdBind

Function	HIST_mdBindDev()
Function Prototype	Int HIST_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	1. 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_HISTCreate function 4) Check if the handle return is NULL or Not If handle is Null return error. 5) Update port status

4.4.2 HIST_mdCreateChan

Function	HIST_mdCreateChan
Function Prototype	Int HIST_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 drivers. mode – mode of the driver (i.e. INPUT) chanParams – EDMA device handle 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) Call PSP_HISTOpen function. 5) Update the port state and put channel as 'in use'.

4.4.3 HIST_mdControlChan

Function	HIST_mcControlChan
Function Prototype	Int HIST_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 port state. 2) Call a function PSP_HISTIoctl to Execute the IOCTL command.

4.4.4 HIST_mdSubmitChan

Function	HIST_mdSubmitChan
Function Prototype	static Int HIST_mdSubmitChan(Ptr chanp, IOM_Packet *packet);
Input Parameters	chanp – pointer to the channel object. IOM_packet * : Pointer to IOM Packet
Output Parameters	Int according to the IOM error codes
Description	This function will call DDA layer functions to perform the various submit operations.
Preconditions	The device driver state should be opened.
Design	Logic in steps 1) Check for port state 2) Call Function PSP_HISTSubmit

4.4.5 HIST_mdUnbindDev

Function	HIST_mdUnbindDev
Function Prototype	Int HIST_mdUnBindDev(Ptr devp)
Input Parameters	Devp – Handle to Histogram device
Output Parameters	Int according to the IOM error code
Description	This function would remove or unload the driver instance.
Preconditions	The driver has to be created

Design	Logic in steps. 1) Check the port state 2) Call the PSP_HISTDelete function. 3) Update the state as DDC_HIST_DELETED.

4.4.6 HIST_mdDeleteChan

Function	HIST_mdDeleteChan
Function Prototype	Int HIST_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 DDA Layer PSP_HISTClose ( ) function.
Preconditions	The driver should be opened
Design	Logic in steps 1) Check the State of the Channel. 2) Call PSP_HISTClose Function 3) Update the state of the channel.

4.4.7 PSP_HISTOpen

Function	PSP_HISTOpen
Function Prototype	PSP_Handle PSP_HISTOpen (PSP_Handle edmaHandleDDA)
Input Parameters	edmaHandleDDA – EDMA device handle.
Output Parameters	Handle to opened channel
Description	This function creates a channel to carry out the transaction.
Preconditions	Driver must be in Created state
Design	Logic in steps 1) Call DDC_HISTOpenHandle Function. 2) Return channel handle to IOM layer.

4.4.8 PSP_HISTClose

Function	PSP_HISTClose
Function Prototype	PSP_Result PSP_HISTClose(PSP_Handle handle)
Input Parameters	Handle – Handle to the Channel
Output Parameters	Int according to the IOM error code
Description	This function calls the dda layer function DDC_HISTCloseHandle() to close the channel.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Call DDC_HISTCloseHandle Function to close the channel.

4.4.9 PSP_HISTIoctl

Function	PSP_HISTIoctl
Function Prototype	PSP_Result PSP_HISTIoctl(PSP_Handle handle, PSP_HISTIoctlCmd cmd, Ptr cmdArg);
Input Parameters	Handle – Handle of the Channel Cmd – Submit command to be passed CmdArg - Argument to the command
Output Parameters	Int according to the PSP errors
Description	This function calls the DDC_HISTIoctl to perform the Ioctl operation.
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_HISTIoctl and set the register using LLC Functions

4.4.10 PSP_HISTCreate

Function	PSP_HISTCreate
Function Prototype	PSP_Handle PSP_HISTCreate(Int32 devid);
Input Parameters	devid – Device id
Output Parameters	Handle 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) Call DDC_HISTInitializeModule 3) Call DDC_HISTSetEventNumber to set the event number. 4) Call DDC_HISTPerformRegOverlaying

4.4.11 PSP_HISTDelete

Function	PSP_HISTDelete
Function Prototype	PSP_Result PSP_HISTDelete(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_HISTRemoveModule

4.4.12 PSP_HISTSubmit

Function	PSP_HISTSubmit
Function Prototype	PSP_Result PSP_HISTSubmit(PSP_Handle chnlHandle, PSP_VPSSSubmitCommand cmd, Ptr cmdArg)
Input Parameters	chnl_Handle - Handle of the Channel cmd - command for enqueue/dequeue cmdArg – command argument
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 & channel must be configured and channel handle should not be NULL.
Design	Logic in steps 1) Check input parameters 2) Check Channel is Closed or not. 3) Call DDC_HISTSubmit

4.5 DDC Layer Functions

4.5.1 DDC_HISTPerformRegOverlaying

Function	DDC_HISTPerformRegOverlaying
Function Prototype	DDC_HISTstatus DDC_HISTPerformRegOverlaying (void);
Input Parameters	NULL

Output Parameters	It returns success of error code.
Description	It will Perform register overlaying.
Preconditions	Device handle should not be NULL.
Design	Logic in steps 1) Get the Histogram device handle. 2) Assign the Histogram register memory base address to the reg_param member of the device handle.

4.5.2 DDC_HISTSetDeviceEventNumber

Function	DDC_HISTSetDeviceEventNumber
Function Prototype	DDC_HISTstatus DDC_HISTSetDeviceEventNumber();
Input Parameters	NULL
Output Parameters	It returns success of error code.
Description	It will assign the event number to the device object.
Preconditions	Device handle should not be NULL.
Design	Logic in steps 1) Get the histogram device handle. Assign the Event number to the element of device handle.

4.5.3 DDC_HISTGetDeviceHandle

Function	DDC_HISTGetDeviceHandle

Function Prototype	PSP_Handle DDC_HISTGetDeviceHandle (void);
Input Parameters	NULL
Output Parameters	Device handle to histogram driver.
Description	It will return the handle for histogram.
Preconditions	None
Design	Logic in steps 1) Return the device object.

4.5.4 DDC_HISTGetChannelHandle

Function	DDC_HISTGetchannelHandle
Function Prototype	PSP_Handle DDC_HISTgetChannelHandle(void);
Input Parameters	NULL
Output Parameters	Channel handle to histogram driver.
Description	It will return the Channel handle for histogram.
Preconditions	None
Design	Logic in steps 1) Return the channel specific object.

4.5.5 DDC_HISTOpenHandle

Function	DDC_HISTOpenHandle
Function Prototype	PSP_Handle DDC_HISTOpenHandle(PSP_Handle edmaHandleDDC);
Input Parameters	edmaHandleDDC – EDMA device handle
Output Parameters	Returns the channel handle.
Description	It will return the Channel handle and update the state of histogram channel & also registers EDMA isr.
Preconditions	Driver must be created.
Design	Logic in steps 1) Call the function DDC_HISTGetDeviceHandle. 2) Initialize the channel configuration structure 3) Update the Channel state to DDC_HIST_CHAN_OPENED. 4) Update the channel state to DDC_HIST_OPENED. 5) Initialize the queue. 6) Create the blocking semaphores. 7) Return the channel handle

4.5.6 DDC_HISTCloseHandle

Function	DDC_HISTCloseHandle
Function Prototype	DDC_HISTstatus DDC_HISTCloseHandle(Ptr handle);
Input Parameters	pointer to handle

Output Parameters	Returns the success or error code
Description	It will Close the logical channel associated with the handle & unregisters the EDMA Isr.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Call the function DDC_HISTGetDeviceHandle. 2) Disable EDMA transfer. 3) Free EDMA channel. 4) Delete channel and isr semaphores. 5) Update the device state to DDC_HIST_CLOSED

4.5.7 DDC_HISTIsChannelHandleClosed

Function	DDC_HISTIsChannelHandleClosed
Function Prototype	DDC_HISTstatus DDC_HISTIsChannelHandleClosed (PSP_Handle handle);
Input Parameters	Handle to channel
Output Parameters	returns the success or error code
Description	It will return the status of the handle. If the handle is opened it will return DDC_HIST_SOK else it will return DDC_HIST_E_FAIL.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps Check the channel state Return the status accordingly

4.5.8 DDC_HISTValidateParameters

Function	DDC_HISTValidateParameters
Function Prototype	DDC_HISTstatus DDC_HISTValidateParameters(DDC_HISTChanObject *, PSP_HISTparams *)
Input Parameters	Pointer to DDC_HISTChanObject Pointer to PSP_HISTparams
Output Parameters	Returns the success of error code
Description	It will Validate the Histogram channel parameters passed by the user & calls LLC layer function to configure the Histogram module.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Validate all parameters. 2) Check the status of histogram. 3) Call LLC_HISTWritereg to set the values of registers.

4.5.9 DDC_HISTGetParameters

Function	DDC_HISTGetParameters
Function Prototype	DDC_HISTstatus DDC_HISTGetParameters(DDC_HISTChanObject *, PSP_HISTparams *)
Input Parameters	Pointer to DDC_HISTChanObject Pointer to PSP_HISTparams

Output Parameters	Returns the success or error code
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 Histogram parameters from the channel specific structure.

4.5.10 DDC_HISTEnquebuffer

Function	DDC_HISTEnquebuffer
Function Prototype	DDC_HISTstatus DDC_HISTEnquebuffer (PSP_Handle chnlHandle, Ptr cmdArg);
Input Parameters	Pointer to channel handle Pointer to command argument
Output Parameters	Return the success or error code
Description	It will perform enqueue operation on the queue.
Preconditions	Driver must be in opened state and channel handle should not be NULL and parameters must be configured
Design	Logic in steps 1) Check the input parameters. 2) Enqueue the buffer and increment enqueue index variable. 3) If it a first buffer then configure & enable the EDMA module.

4.5.11 DDC_HISTDequebuffer

Function	DDC_HISTDequebuffer
Function Prototype	DDC_HISTstatus DDC_HISTDequebuffer (PSP_Handle chnlHandle, Ptr cmdArg);
Input Parameters	Pointer to channel handle Pointer to command argument
Output Parameters	NULL
Description	It will perform dequeue operation on the queue. If dequeue operation is successful then it will return buffer containing the statistics.
Preconditions	Driver must be in opened state and channel handle should not be NULL and parameters must be configured
Design	1)If no buffer available in dequeue but if enqueue is having a buffer then perform the semtake operation on dequeue blocking semaphore. 2)Deque the buffer from dequeue list and decrement Dequeue index variable.

4.5.12 DDC_HISTEdmaisr

Function	DDC_HISTEdmaisr
Function Prototype	void DDC_HISTEdmaisr (Uint tcc, EDMA3_RM_TccStatus status, Ptr data);
Input Parameters	tcc - transmission interrupt number status – transmission status data – argument to be passed to EDMA isr

Output Parameters	None
Description	It will put buffer containing the statistics on the dequeue from enqueue.
Preconditions	None
Design	1) Disable edma transfer. 2) Check that there are at least two buffers in enqueue queue. 3) Extract the buffer from enqueue and decrement enqueue index. 4) Insert buffer containing the statistics on the dequeue stack & increment dequeue index. 5) Perform semgive operation on dequeue blocking semaphore. 6) Set EDMA parameters for next transfer, with address as next buffer in Enqueue queue. 7) If only one buffer is there in Enqueue queue, buffer will not be dequeue, but it will be put again for EDMA transfer.

4.5.13 DDC_HISTRemoveModule

Function	DDC_HISTRemoveModule
Function Prototype	PSP_Handle DDC_HISTRemoveModule (void);
Input Parameters	NULL
Output Parameters	Pointer to DDC layer device structure
Description	It will make the device handle NULL.
Preconditions	None

Design	Logic in steps 1) Get the Histogram device handle. 2) Check the status of the device. 3) Make the Histogram device handle NULL & return it.

4.5.14 DDC_HISTIoctl

Function	DDC_HISTIoctl
Function Prototype	DDC_HISTstatus DDC_HISTIoctl(PSP_Handle handle, PSP_AFIoctlCommand cmd, Ptr cmdArg);
Input Parameters	Handle – Handle of the Channel Cmd – Submit command to be passed CmdArg Argument to the command
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) Execute ioctl related Functions

4.5.15 DDC_HISTSubmit

Function	DDC_HISTSubmit
Function Prototype	DDC_HISTStatus DDC_HISTSubmit(PSP_Handle chnlHandle, PSP_VPSSSubmitCommand cmd, Ptr cmdArg)
Input Parameters	chnlHandle - Pointer to Channel Object cmd – command for enqueue/dequeue cmdArg – command argument

Output Parameters	NULL
Description	Function to execute the Submit Command.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps : 1) Read the command 2) Call DDC_HISTEnquebuffer if command is enqueue. 3) Call DDC_HISTDequebuffer if command is dequeue.

4.5.16 DDC_HISTInitailizeModule

Function	DDC_HISTInitializeModule
Function Prototype	Ptr DDC_HISTInitializeModule (void);
Input Parameters	NULL
Output Parameters	Pointer to DDC layer device structure
Description	It will update the state of device.
Preconditions	None
Design	Logic in steps 1) Get the histogram device handle. 2) Update the device handle parameters. 3) Return the histogram device handle.

4.6 LLC Layer Functions

4.6.1 LLC_HISTWritereg

Function	LLC_HISTWritereg
Function Prototype	Int LLC_HISTWritereg (PSP_HISTparam * param, Ptr regs);
Input Parameters	Pointer to parameter structure Pointer to register overlaying structure.
Output Parameters	Success of error code.
Description	This Function will write all the parameters into the hardware registers except enable bit.
Preconditions	Driver must be in opened state & overlaying structure pointer should not be NULL.
Design	Logic in steps 1) Configure Histogram hardware registers from channel config parameter structure after doing appropriate masking.

4.6.2 LLC_HISTEnableHistogram

Function	LLC_HISTEnableHistogram
Function Prototype	Int LLC_HISTEnableHistogram (Ptr regs);
Input Parameters	Pointer to register overlaying structure.
Output Parameters	Success of error code.

Description	This Function will enable the histogram module.
Preconditions	Driver must be in opened state & overlaying structure pointer should not be NULL.
Design	Logic in steps 1) Set enable bit into the Histogram PCR register.

4.6.3 LLC_HISTDisableHistogram

Function	LLC_HISTDisableHistogram
Function Prototype	Int LLC_HISTDisableHistogram (Ptr regs);
Input Parameters	Pointer to register overlaying structure.
Output Parameters	Success of error code.
Description	This Function will disable the histogram.
Preconditions	Driver must be in opened state & overlaying structure pointer should not be NULL.
Design	Logic in steps 1) Clear enable bit into the Histogram PCR register.

4.6.4 LLC_HISTGetHWStatus

Function	LLC_HISTGetHWStatus
Function Prototype	Int LLC_HISTGetHWStatus (Ptr regs);

Input Parameters	Pointer to register overlaying structure.
Output Parameters	None
Description	This Function will return the hardware status .i e: busy(1)/not busy(0) or error code(-1).
Preconditions	Driver must be in opened state & overlaying structure pointer should not be NULL.
Design	Logic in steps 1) Read status bit from the Histogram PCR register.

4.6.5 LLC_HISTGetHistDataAdrs

Function	LLC_HISTGetHistDataAdrs
Function Prototype	Uint32 LLC_HISTGetHistDataAdrs (PSP_Handle regs);
Input Parameters	Pointer to register overlaying structure.
Output Parameters	None
Description	This Function will return the address of the HIST_DATA register.
Preconditions	Driver must be in opened state & overlaying structure pointer should not be NULL.
Design	Logic in steps 1) Return the address of the HIST_DATA register.

5 Decision Analysis & Resolution

5.1 Buffer Management Mechanism

5.1.1 DAR Criteria

The list of DAR Criteria is as follows:

- Interface Simplicity
- Better control to application

5.1.2 Available Alternatives

5.1.2.1 Alternative 1

Histogram driver will use same buffer management as CCDC.

5.1.2.2 Alternative 2

Histogram driver will use buffer management similar to CCDC, but it will not keep one buffer reserved in input queue. If input queue runs out of buffers, driver will disable the Histogram engine. Engine will be enabled again when ever new buffer is added to the input queue.

5.1.3 Decision

As continuous capturing is possible in alternative 1, we will go with alternative 2.

5.2 Copy Mechanism For Statistics

5.2.1 DAR Criteria

The list of DAR Criteria is as follows:

- Lower Resource Utilization

5.2.2 Available Alternatives

5.2.2.1 Alternative 1

Histogram driver will use interrupt service to transfer data from histogram output memory to enqueued buffer.

5.2.2.2 Alternative 2

Histogram driver will use EDMA service to transfer data from histogram output memory to enqueued buffer. This EDMA will be linked with the EDMA event of the Histogram.

5.2.3 Decision

Since EDMA transfer will be done in parallel and will also reduce the interrupt execution time, histogram will use EDMA service to transfer data from histogram output memory to enqueued buffer.

6 Revision History

Version #	Date	Author Name	Revision History
Draft 1.01	12 OCT 2006	EI5	Initial Draft Created
Draft 1.02	13 OCT 2006	EI5	Updated for technical & QA review comments.
Issue 1.00	13 OCT 2006	EI5	Issued to TII
Issue 1.01	19 NOV 2006	EI5	1) Name of the control commands are corrected 2) DDC_HISTOpenHandle & DDC_HISTCloseHandle are modified. 3) Added HIST_mdSubmitChan, PSP_HISTSubmit, DDC_HISTSubmit 4) Maximum device instance macro is added 5) Handle for Histogram channel object is added in DDC_HISTdeviceobject structure 6) Six parameters are added to DDC_HISTchannelobject : semIsr,edmaCreatedmaInstanceId,stateCH,edmaHandle,tcc 7) DDC_HISTchannelobject is replaced with HIST_iomchannelobj in HISTPORTOBJ structure 8) 4th argument is removed from PSP_HISTIoctl() 9) Parameters are added in DDC_HISTEdmaisr routine 10) Function input Parameters are changed in LLC layer functions 11) DDA_HISTRegisterEventHandler & DDA_HISTunRegisterEventHandler are deleted. 12) PSP_HISTEdmaobj structure info is added in GIO_CREATE 13) DDC_HISTDequebuffer & DDC_HISTEdmaisr are modified. 14) Values of CFA pattern macro & input source macro are changed. 15) DDC_HIST_NO_DATA error code is added

			in DDC_HISTstatus enumeration. 16) Enumeration for EDMA creation information added. 17) EDMA parameter structure added.
Pre-silicon Release 0.3.0	20 NOV 2006	EI5	Release to TI
Post-silicon Release 0.3.0	29 NOV 2006	EI5	Release to TI
Post-silicon Release 0.3.0	4 DEC 2006	EI5	1) Two parameters cfaPattern & ipSource are removed from PSP_HISTparam structure 2) Section of CFA pattern & Input source macro are removed.
Post-silicon Release 0.3.0	11 DEC 2006	EI5	Description of section 3.1.1 is removed. Appropriate Reference of document is given
Post-silicon Release 0.3.0	26 DEC 2006	EI5	1) Modified after removing PSP_HISTedmaobj structure.
Post-silicon Release 0.4.0	2 JAN 2007	EI5	1) Modified after buffer management design change. 2) PSP_HISTOGRAM_STATE is renamed as DDC_HISTOGRAM_STATE
Post-silicon Release 0.6.0	28 FEB 2007	EI5	1) Figure-19 modified. 2) Section-1.4 modified. 3) Section-2.2 modified.
1.00.01	June 22, 2007	Anuj Aggarwal	GA Patch Release 1
1.00.02	June 29, 2007	Amit Chatterjee	Modified Release Version
1.00.03	July 18, 2007	Maulik Desai	Modified Release Version