TII DM6437 VPSS Drivers

H3A Design Specifications

TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

Instructions:

This document captures the design of H3A driver for DMDM6437.

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.

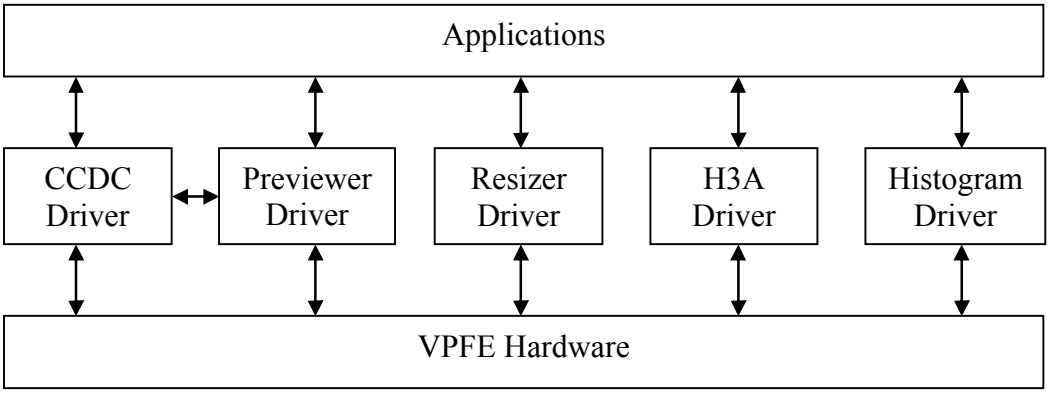

Figure 1. VPFE Driver Stack

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
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 H3A Driver is to develop an integrated driver for AF and AEW modules of VPSS. It should be fully programmable & configurable and all hardware features shall be supported.H3A receives the raw image/video data from the video port (CCDC). Input data is 10bits wide.

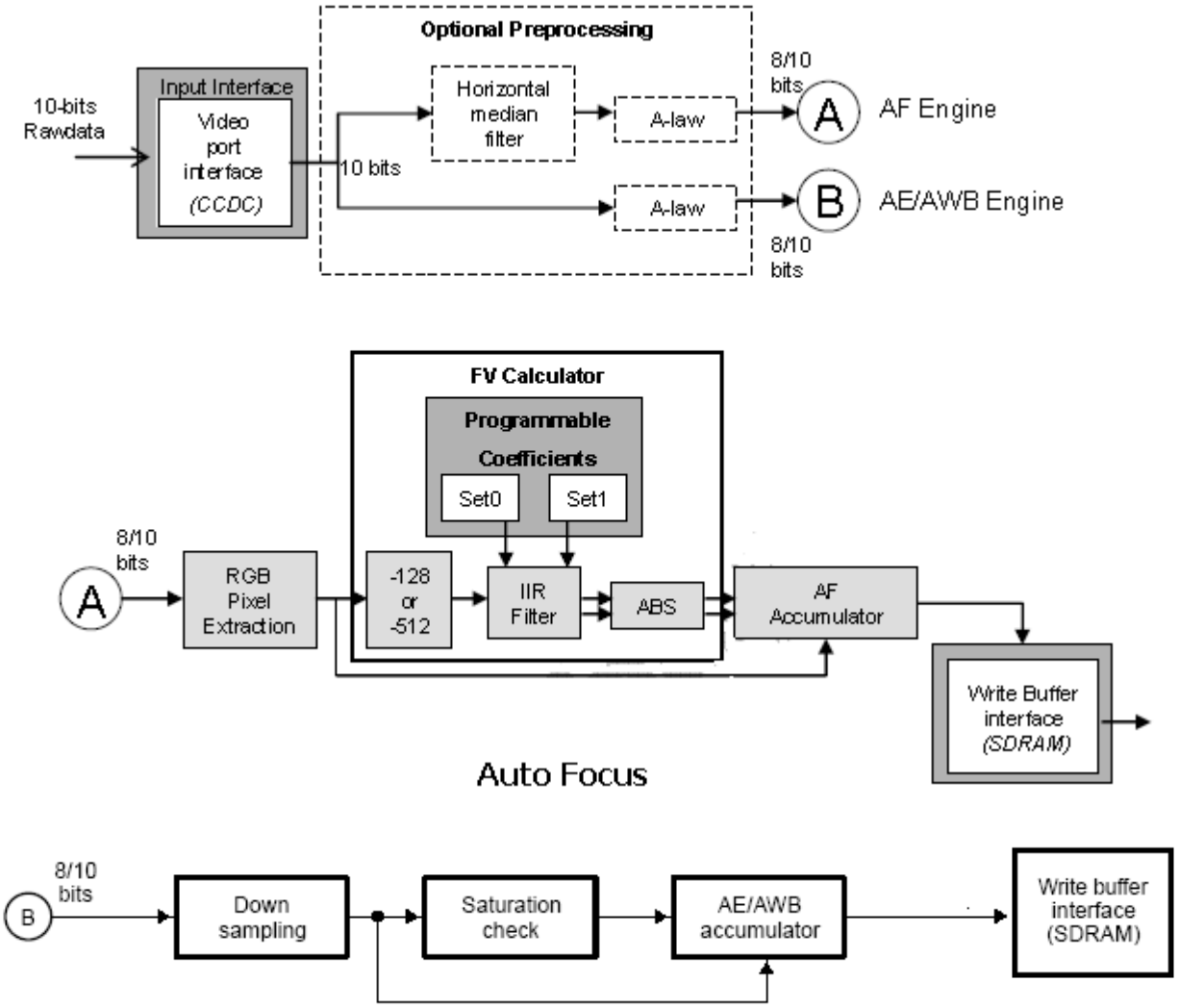

Figure 2. Block Diagram of H3A

2 Requirements

The DM6437 H3A driver comprises of Auto Focus and Auto Exposure/Auto White Balance modules.

The DM6437 AF module Functional requirements are as follows:

- AF Module shall support Auto Focus by providing the peak/sum values of the configured Paxels.
- AF Module shall support configuration of HMF.
- AF Module shall support configuration of A-law.
- AF Module shall support configuration of RGB Color positioning to perform RGB pixel extraction.
- AF Module shall support configuration of IIR Filter Coefficients.
- AF Module shall add time stamp (using BIOS API) to the captured statistics.
- AF Module shall manage the buffers using Queue/Dequeue mechanism. It should automatically enable/disable HW engine depending on the buffer availability.
- Module shall support input from CCDC
- Module shall support output to DDR.

The DM6437 AEW Module Functional requirements are as follows:

- AEW Module shall support Auto white Balance and Auto Exposure by providing the peak/sum values of the configured window.
- AEW Module shall support configuration of A-law.
- AEW Module shall support configuration of saturation check to set the clipping value.
- AEW Module shall add time stamp (using BIOS API) to the captured statistics.
- AEW Module shall manage the buffers using Queue/Dequeue mechanism. It should automatically enable/disable HW engine depending on the buffer availability.
- Module shall support input from CCDC
- Module shall support output to DDR.

2.1 Assumptions

None

2.2 Constraints

Raw Pattern Type

- H3A will provide support for only Bayer Pattern Input.

AEW Output Address

- AEW Output address must be on 64 byte boundaries.
- Windows cannot overlap

AEW Window Configuration

- The width and height of the windows must be an even number.
- Sub-sampling windows can only start on even numbers.
- The minimum width of the AE/AWB windows must be 6 pixels.
- Support for up to 36 windows in horizontal direction
- Support for up to 128 windows in vertical direction
- Windows cannot overlap
- All windows must be accommodated in the image.

(Windows Horizontal Start + (Width of window * Windows Horizontal Count)) should be less then or equal to Image width

(Windows Vertical Start + (Height of window * Windows Vertical Count)) should be less then or equal to Image Height

AF Horizontal Median Filter

- If the median filter is enabled then the first two and last two pixels in the frame are not in the valid region. Therefore the Paxel start/end and IIR filter start positions should not be set within the first two and last two pixels.

AF Paxel Configuration

- The minimum width of the Paxel must be 6 pixels. The width and height of the Paxels must be an even number.
- Paxels cannot overlap and must be adjacent to one another.
- Paxel horizontal start value must be greater than or equal to the (IIR horizontal start position + 2)
- Support for up to 36 paxels in horizontal direction
- Support for up to 128 paxels in vertical direction
- All paxels must be accommodated in the image.

(Paxel Horizontal Start + (Width of Paxel * Paxel Horizontal Count)) should be less then or equal to Image width

(Paxel Vertical Start + (Height of Paxel * Paxel Vertical Count)) should be less then or equal to Image Height

AF Window Size

- The minimum width of the window must be 6 pixels. The width and height of the window must be an even number.

AF Output Address

- AF Output address must be on 64 byte boundaries.

AF IIR Filter

- Horizontal Start Position for IIR Filter must be even.

3 Design Description

This chapter gives detail on the overall architecture of TI DM6437 H3A device driver. This includes the static view explaining the functional decomposition and dynamic view explaining the deployment scenario of the H3A 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 H3A driver.

The various functionalities of H3A driver are as follows:

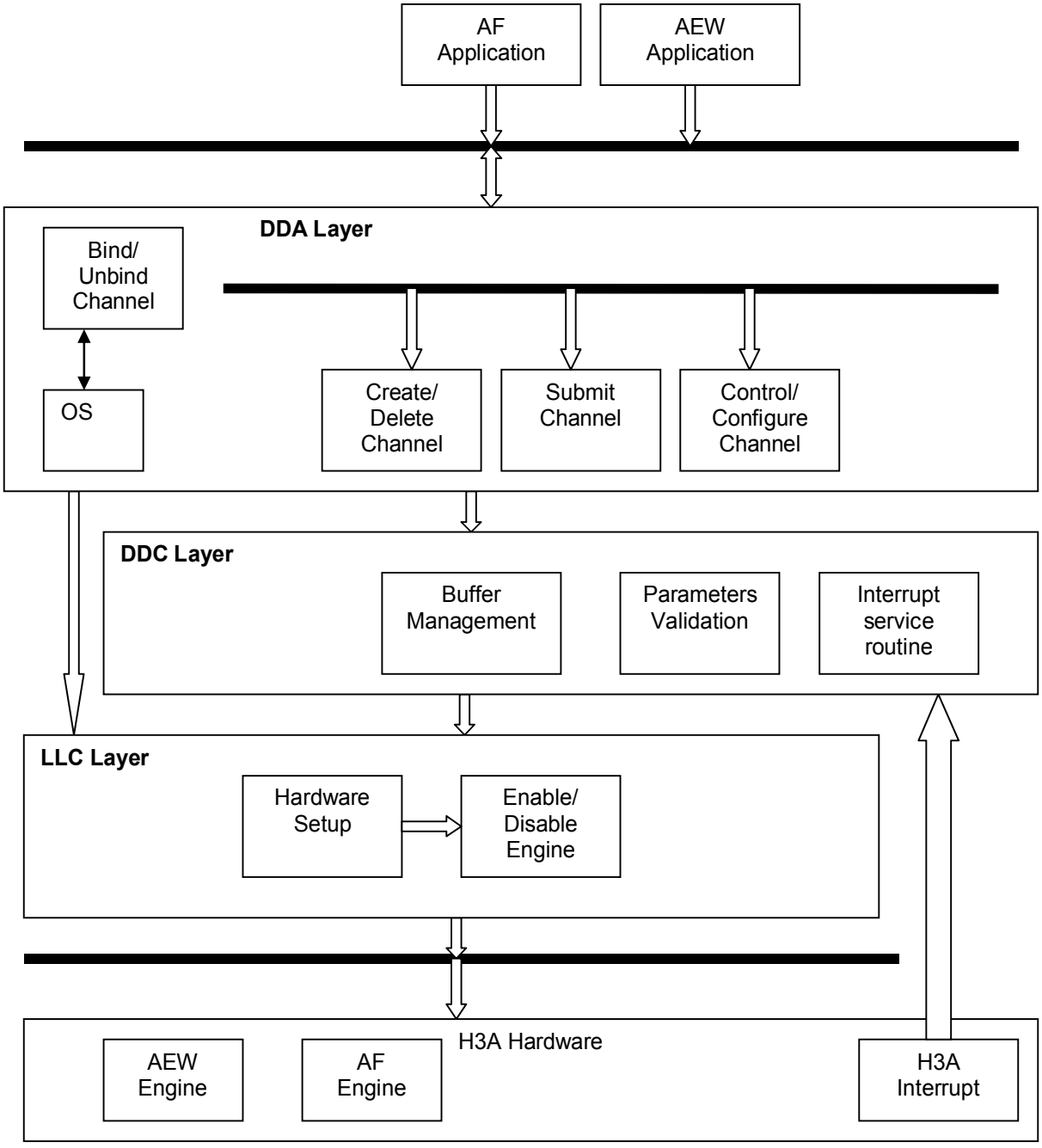

Figure 3. Detailed Design Diagram for H3A

Various functionalities of different layers in H3A driver are as below:

DDA (IOM) layer

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

DDC Layer

- Multiple Application Handling
- Parameters Validation
- Interrupt Service Routine

LLC Layer

- Hardware setup
- Enable / Disable engine

3.1.2.1 Driver Creation/deletion

Driver Creation

The sequence diagram below depicts the creation phase of the BIOS H3Adriver. 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 H3A_mdBindDev (), way up in the application startup phase, perhaps in a central driver initialization function.

The H3A_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.

The figure for AF Channel Creation is as follows:

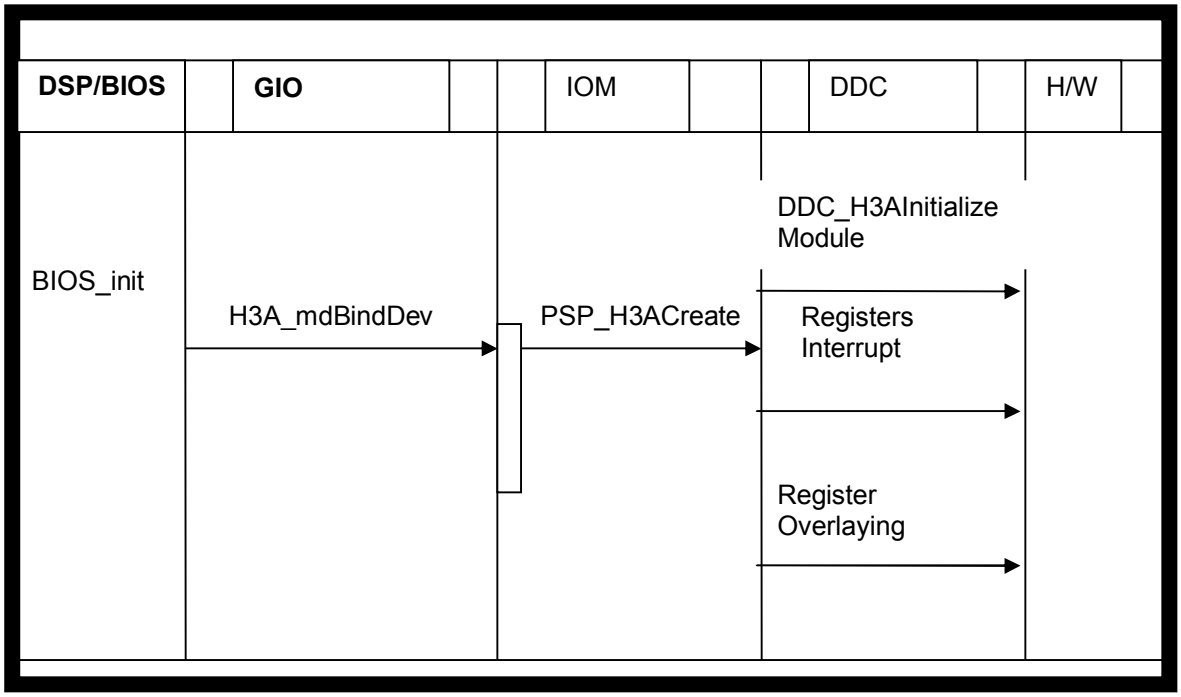

Figure 4. H3A Driver Creation overview

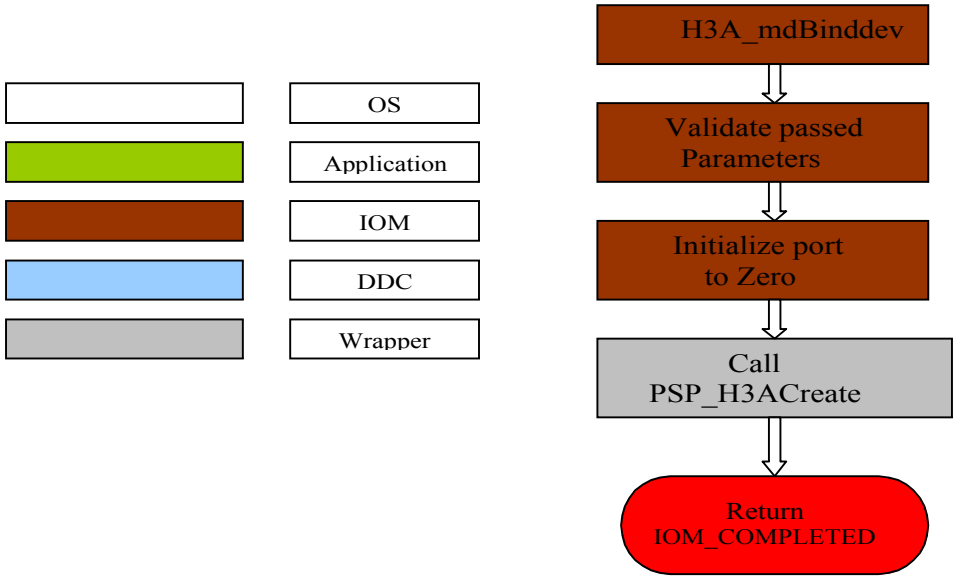

Figure 5. Driver Creation detail flow diagram-1

PSP_H3ACreate

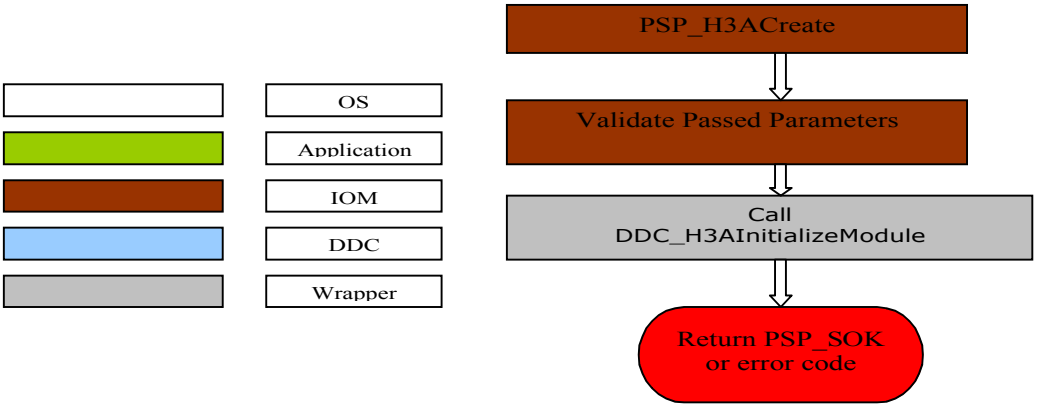

Figure 6. Driver Creation Detail flow diagram -2

DDC_H3AInitializeModule

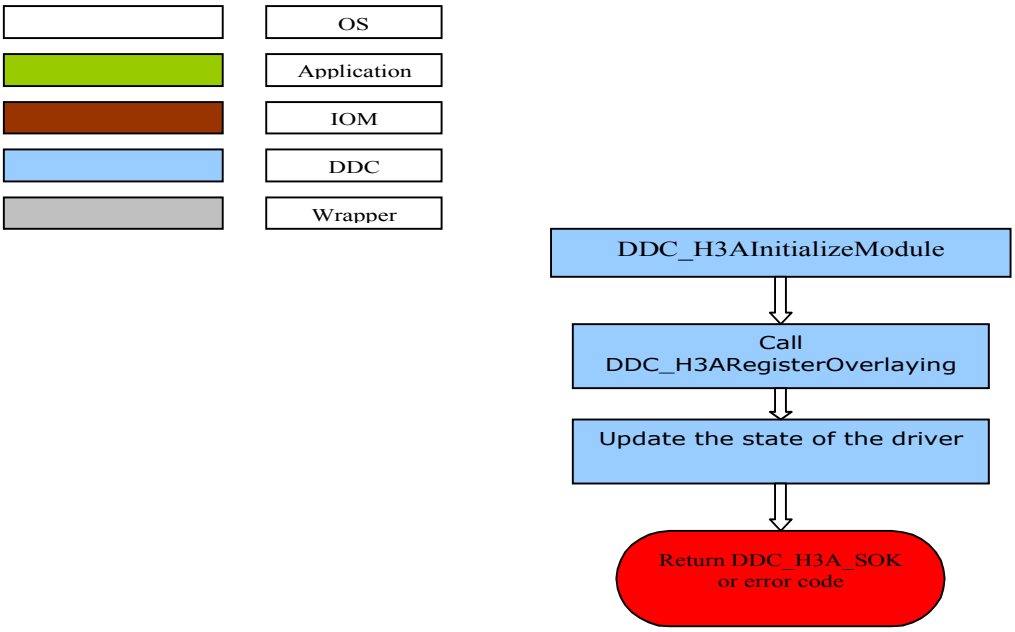

Figure 7. Driver Creation Detail flow diagram -3

Driver Deletion

Following the call mdUnBindDev () one is required to restart from beginning over an mdBindDev () call to bring driver back to life. The driver de-initialize and delete function de-initialize the Driver and delete if any OS resources originally allocated through mdBindDev.

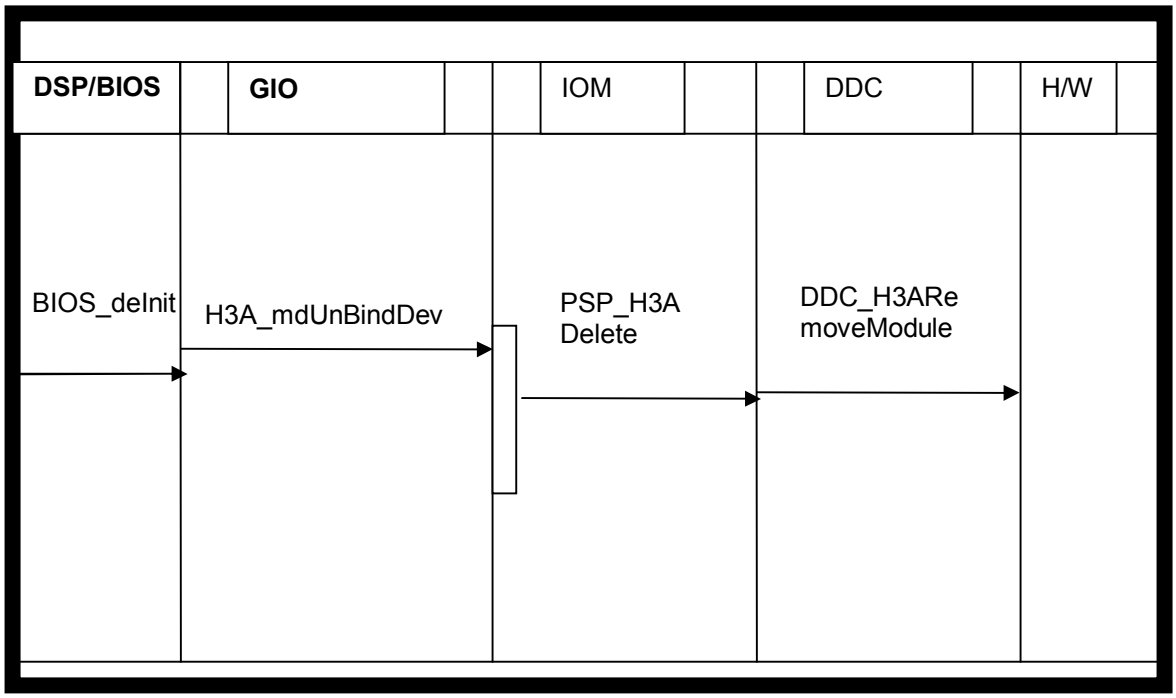

Figure 8. Driver deletion overview

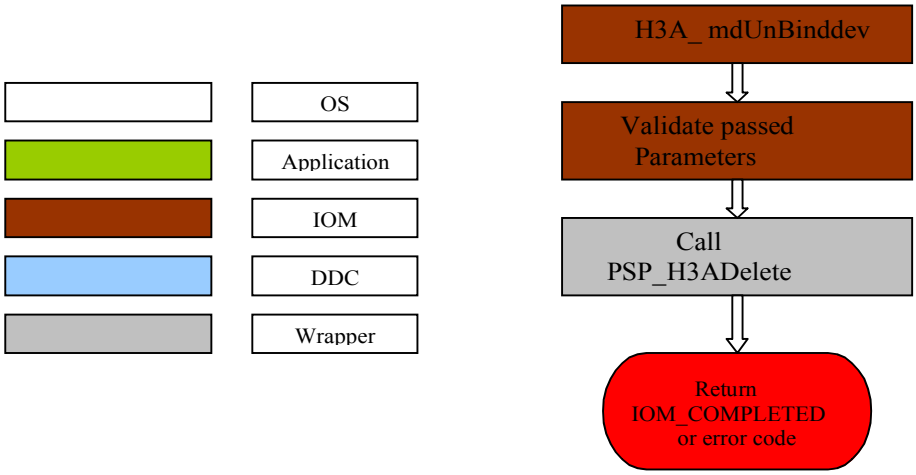

Figure 9. Driver Deletion detail Flow -1

PSP_H3ADelete

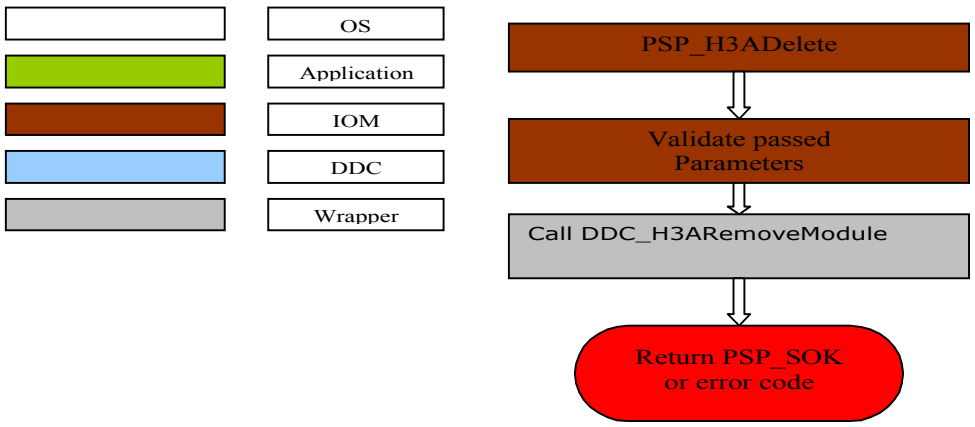

Figure 10. Driver deletion detail flow-2

DDC_H3ARemoveModule

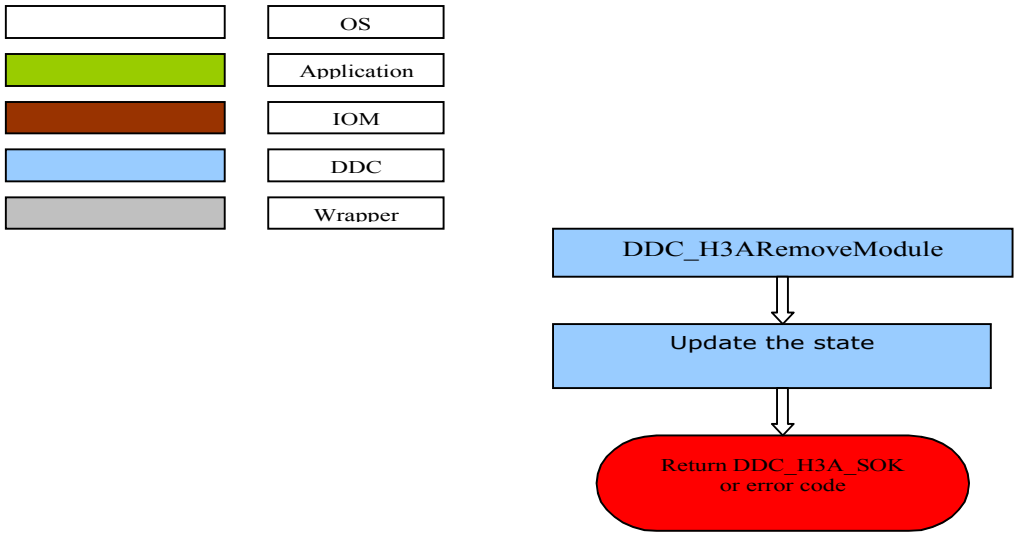

Figure 11. Driver deletion detail flow-3

3.1.2.2 Driver Open/Close

Driver_Open

When the application calls the H3A_mdCreateChan ( ), driver entry point is created. The chanParams field in mdCreateChan determines the channel type. Channel Type Field is set in the Channel Object. Driver will fetch the Function Table based on the type of channel. Driver will invoke Functions using Function Table.

The driver creates channel overview for AF Channel is as follows:

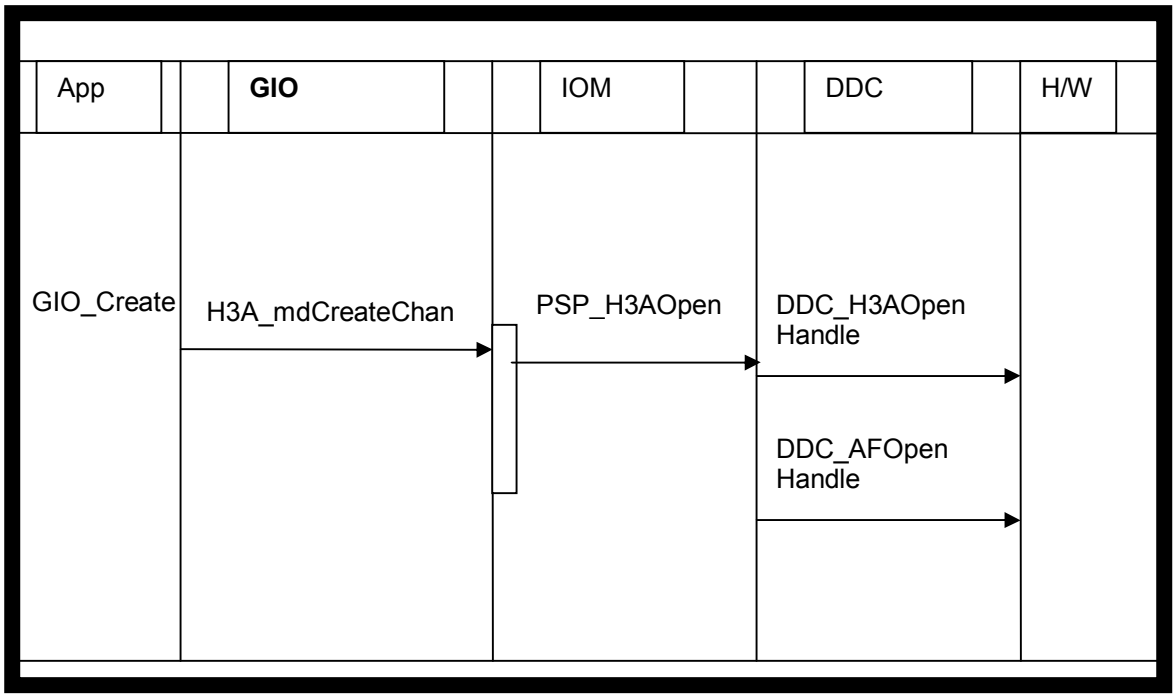

Figure 12. Driver create channel overview for AF Channel

The driver creates channel overview for AEW Channel is as follows:

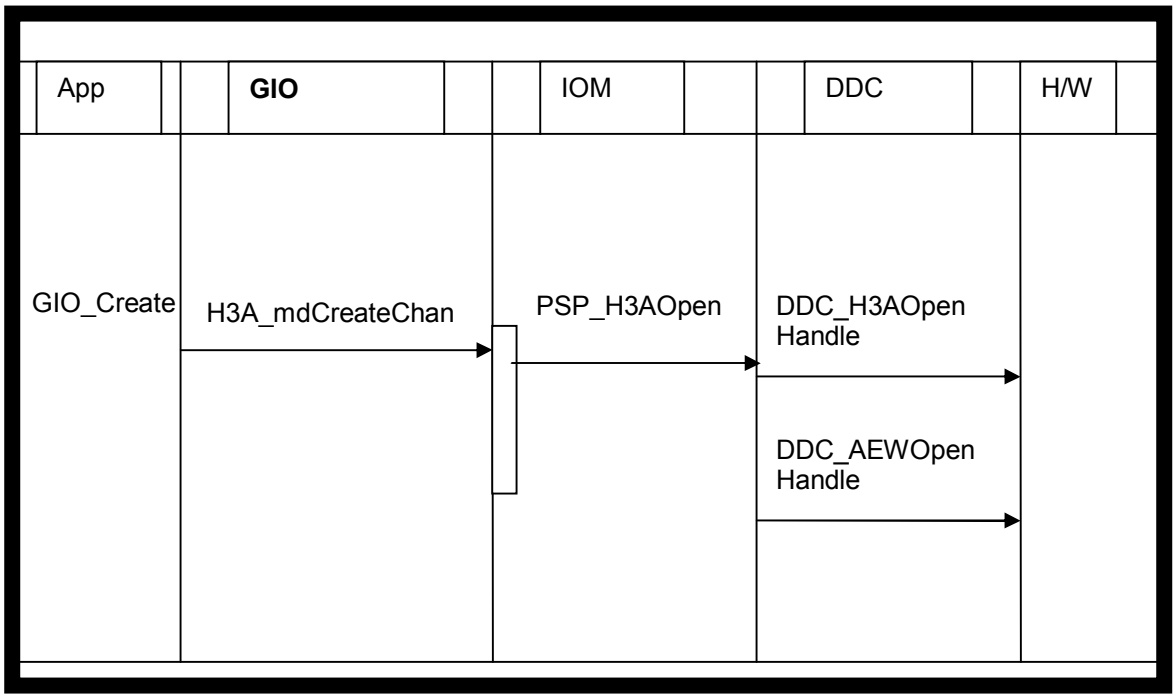

Figure 13. Driver create channel overview for AEW Channel

H3A_mdCreateChan

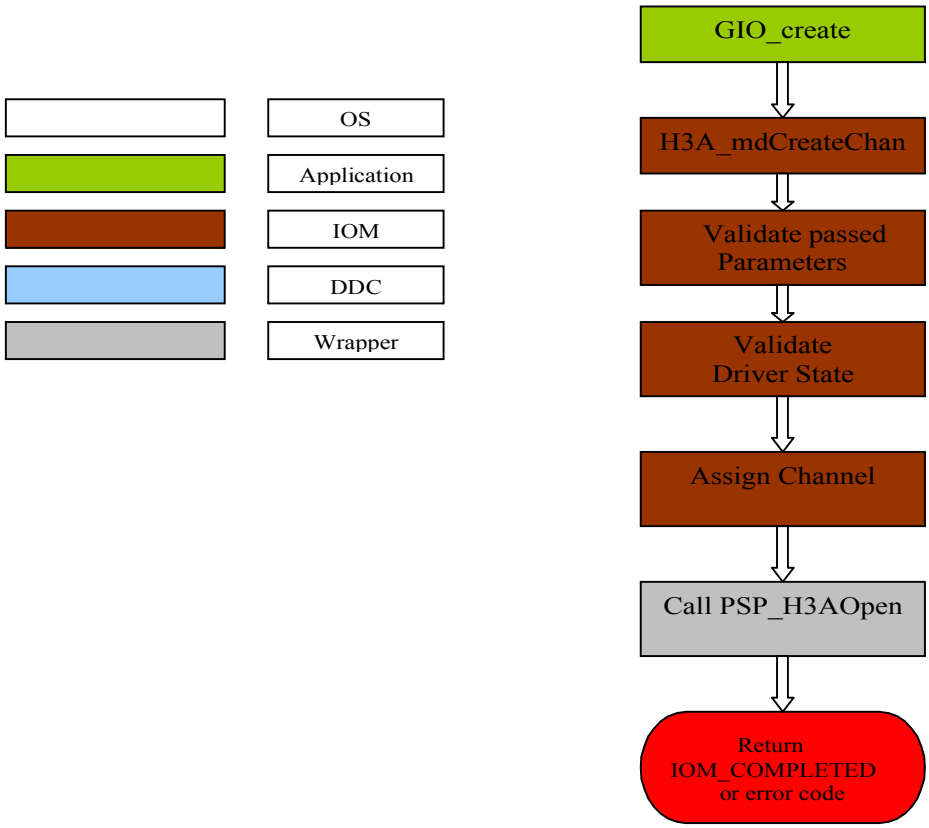

Figure 14. Driver Open Detail flow -1

PSP_H3AOpen

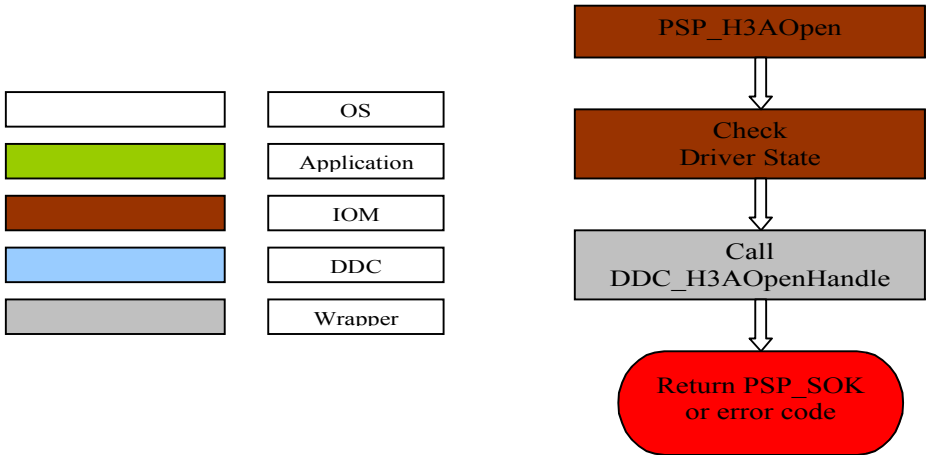

Figure 15. Driver Open detail flow -2

DDC_H3AOpenHandle

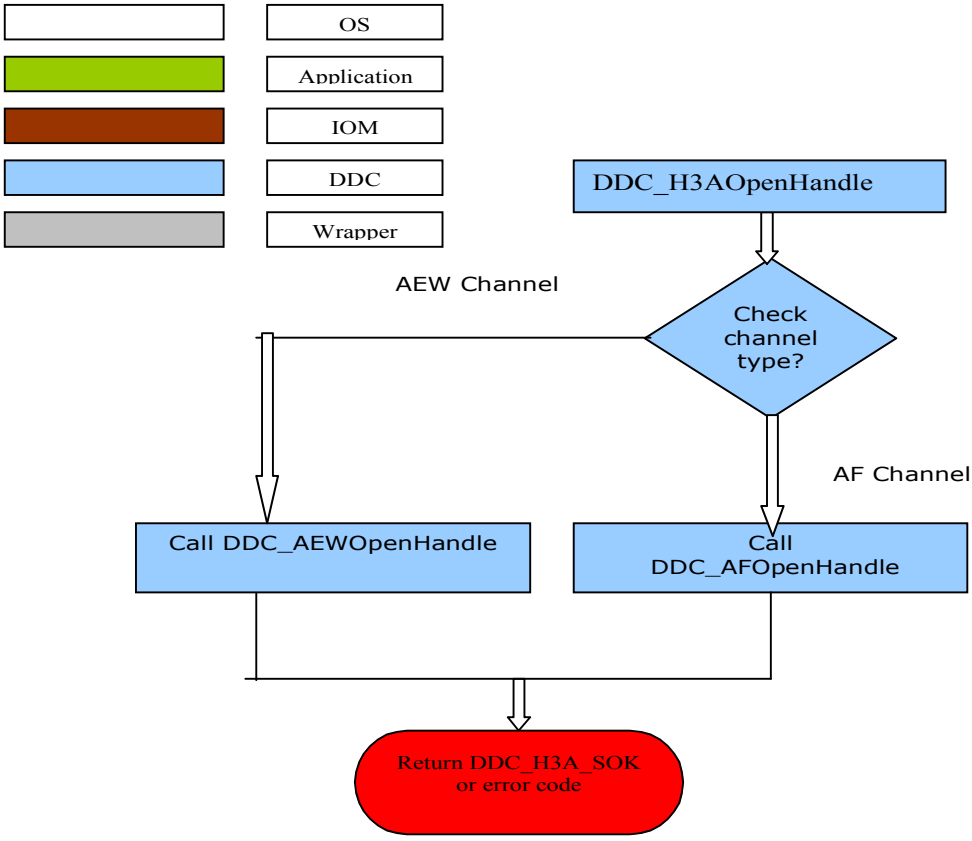

Figure 16. Driver Open detail flow -3

DDC_AFOpenHandle

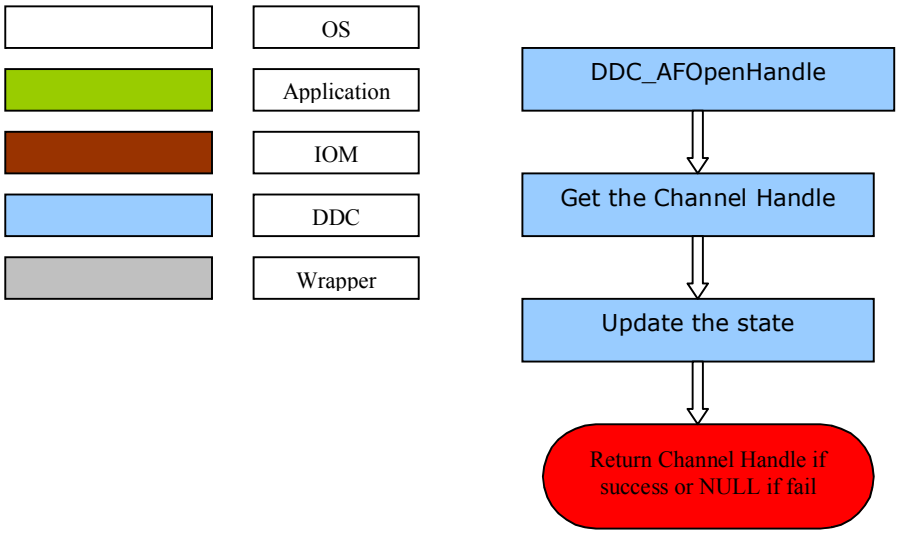

Figure 17. Driver Open detail flow -3 for AF Channel

DDC_AEWOpenHandle

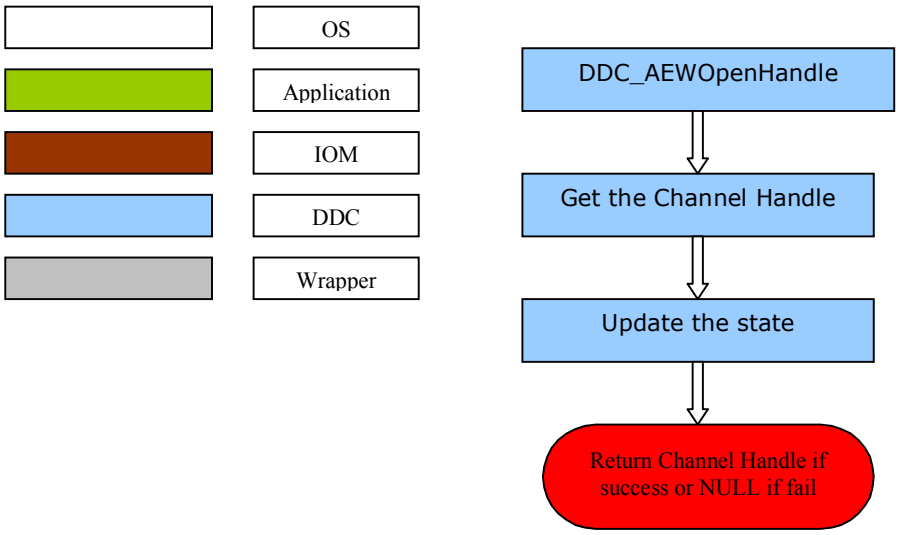

Figure 18. Driver Open detail flow -4 for AEW Channel

Driver Close

The application invokes the mdDeleteDev () function to close the channel of the VPFE 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 mdCreateChan ().The device will be deleted depending upon the device id.

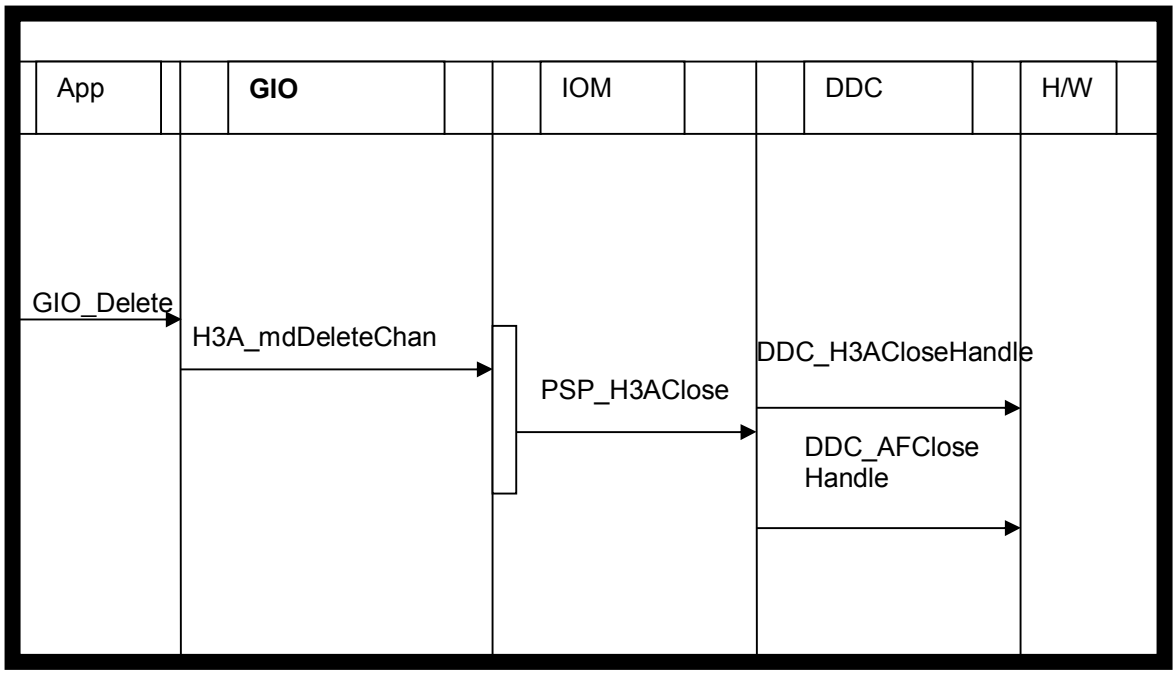

Figure 19. Driver channel close overview for AF Channel

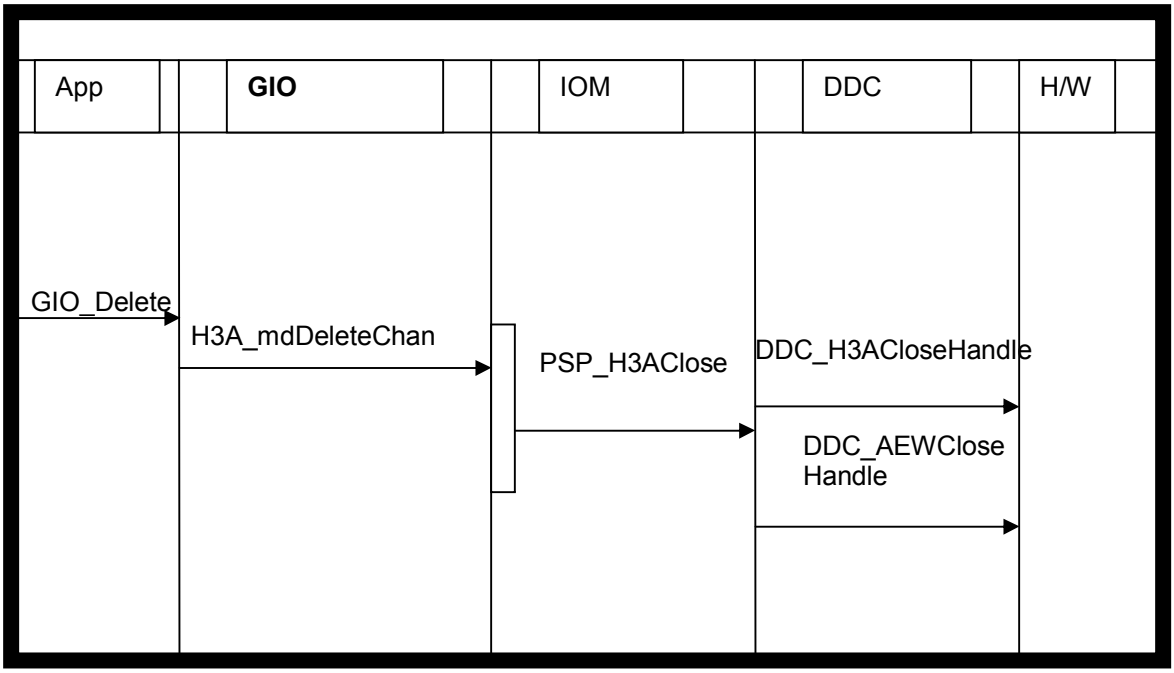

Figure 20. Driver channel close overview for AEW Channel

H3A_mdDeleteChan

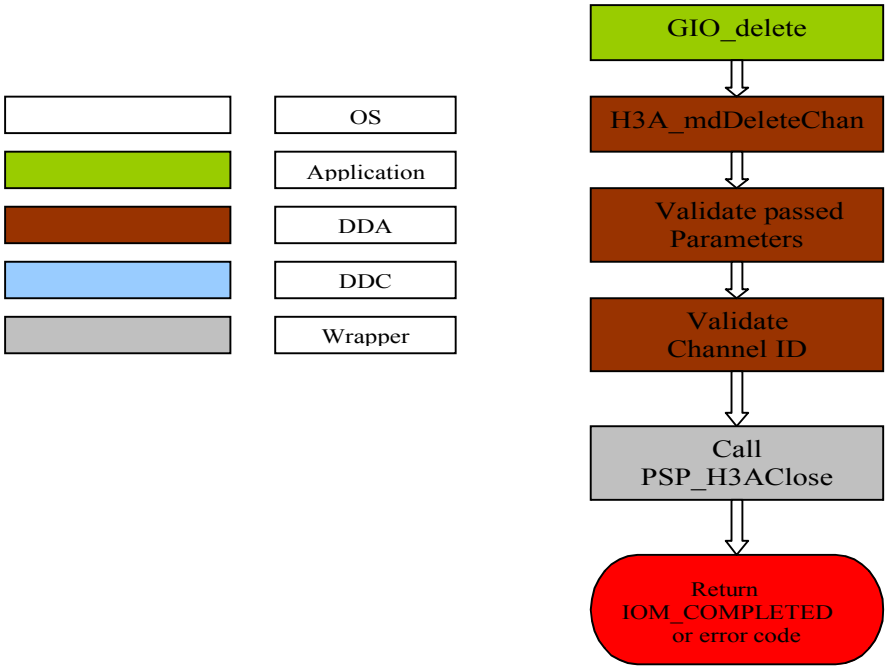

Figure 21. Driver close channel detail flow -1

PSP_H3AClose

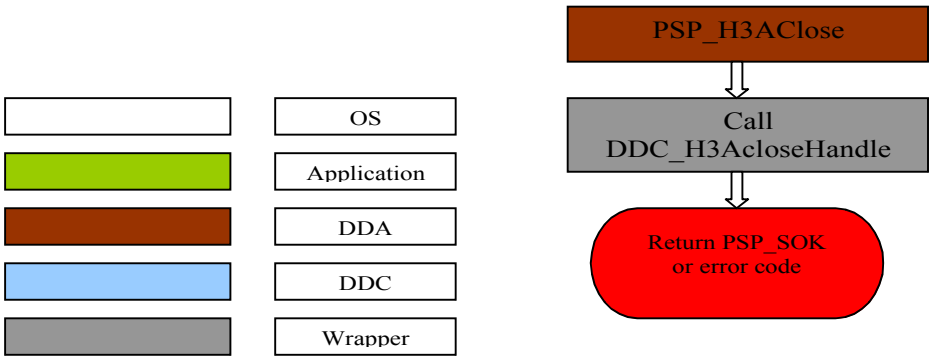

Figure 22. Driver close channel detail flow diagram -2

DDC_H3ACloseHandle

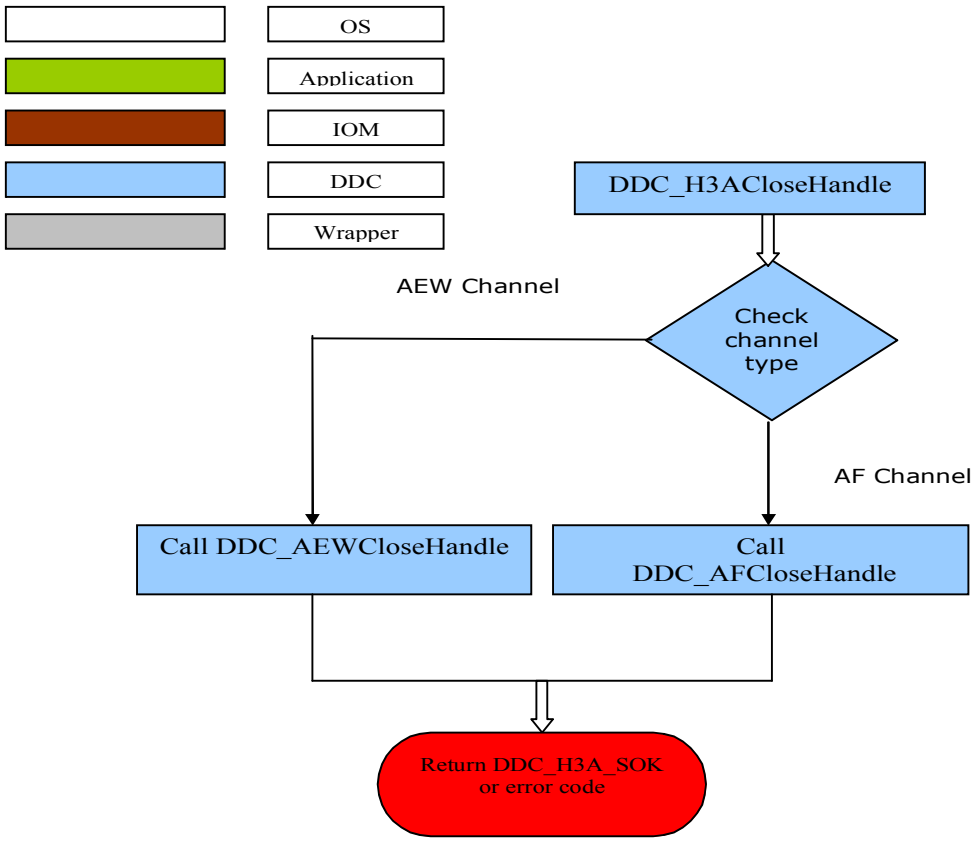

Figure 23. Driver close channel detail flow-3

DDC_AFCloseHandle

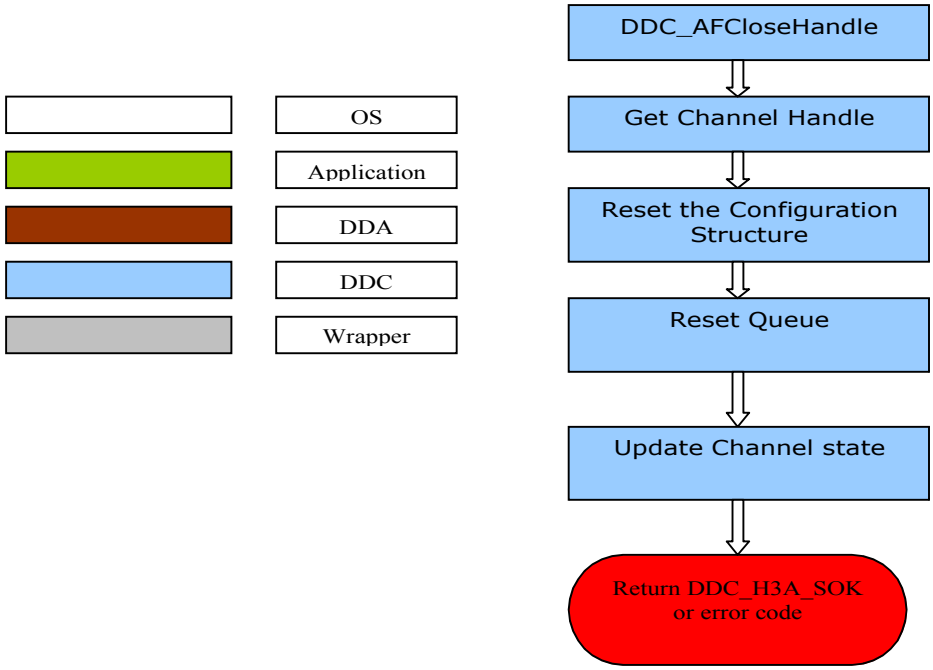

Figure 24. Driver close Channel for AF Channel

DDC_AEWCloseHandle

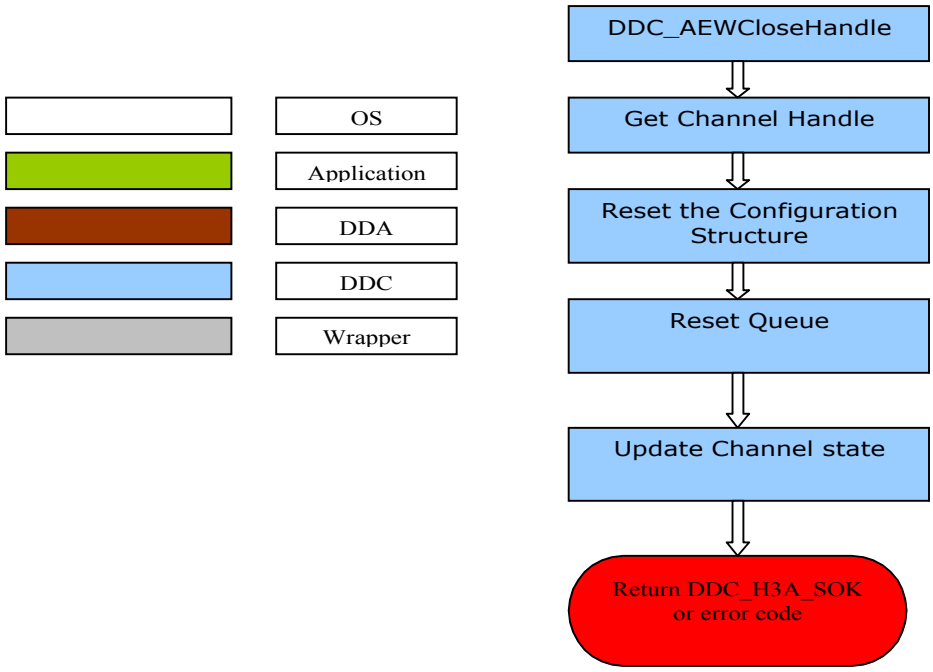

Figure 25. Driver close Channel for AEW Channel

3.1.2.3 Various Controls

The H3A Driver provides mdControlChan ( ) to set/get common configuration parameters on the driver at run time through the corresponding DDC IOCTL function, ddc_H3AIoctl ( ). 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.

H3A_mdControlChan

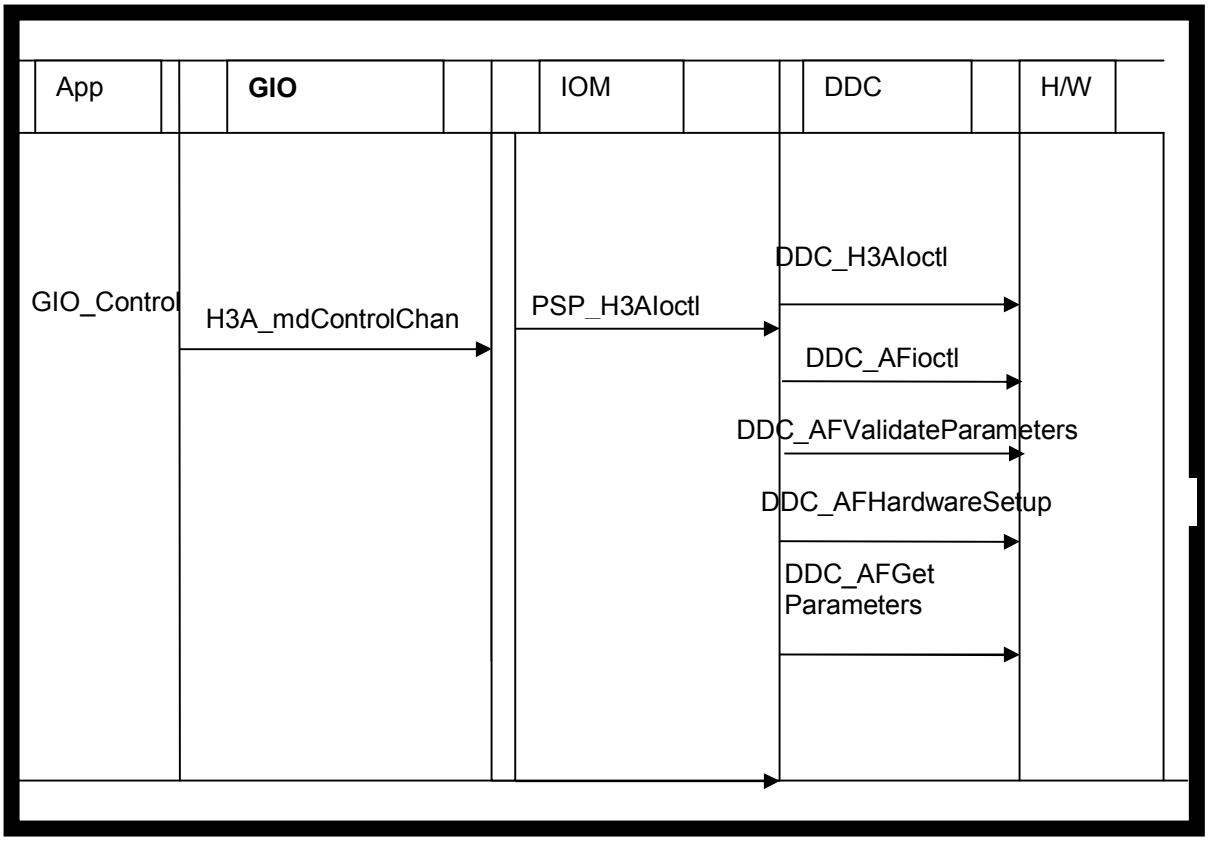

Figure 26. Control Command overview for AF Channel

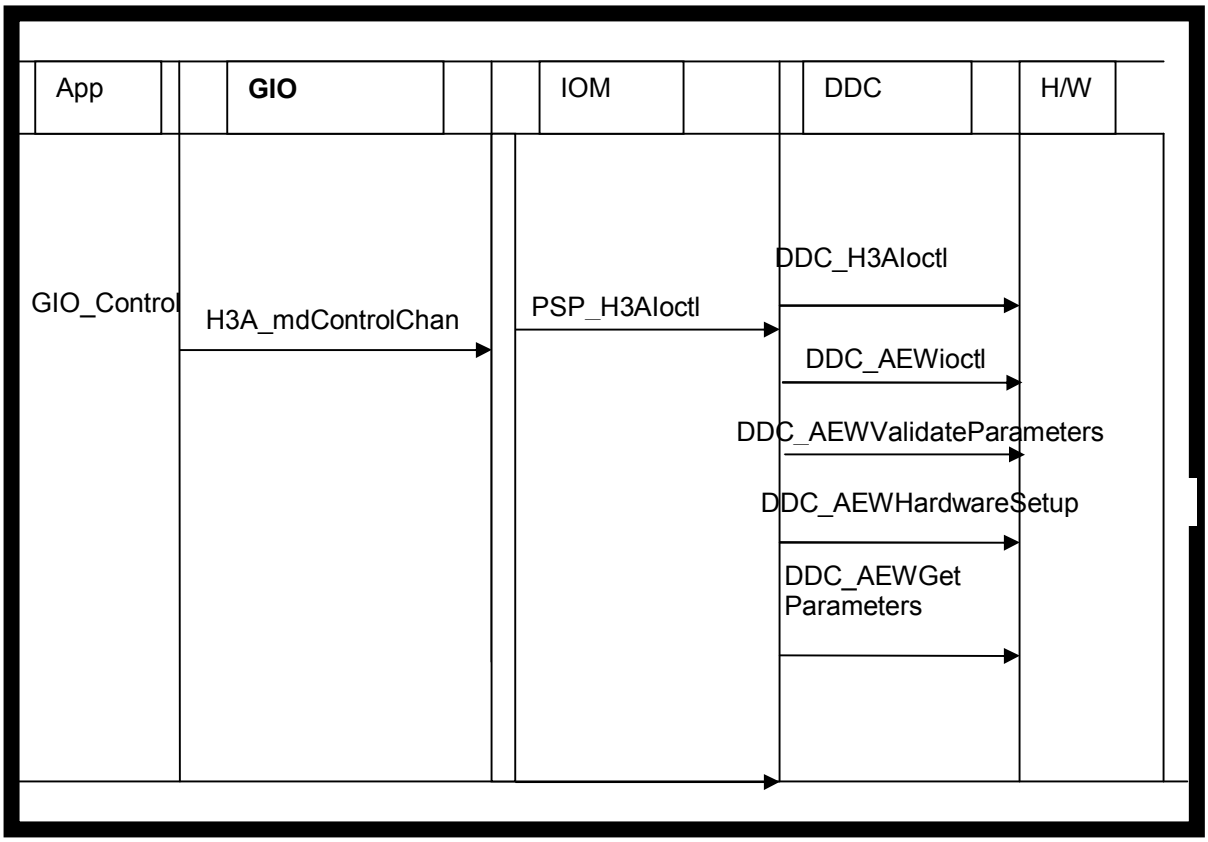

Figure 27. Control Command overview for AEW Channel

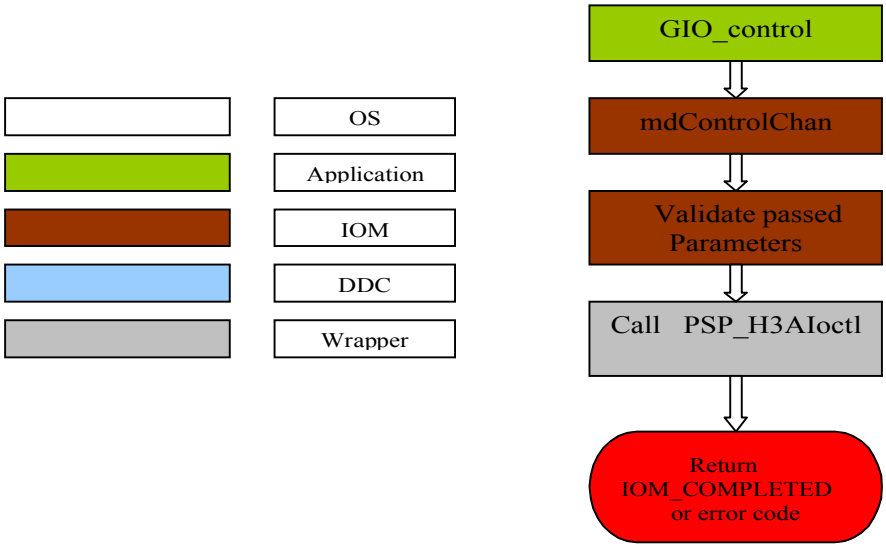

Figure 28. Overview of control command

PSP_H3AIoctl

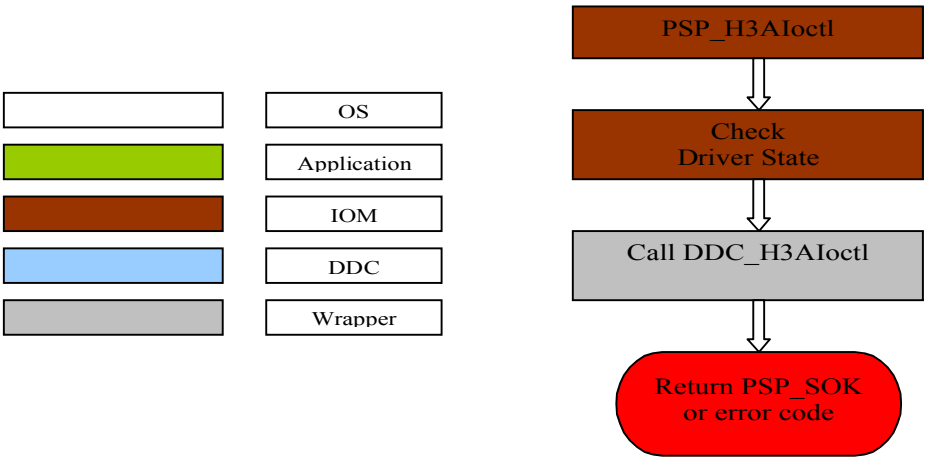

Figure 29. Overview of control command

DDC_H3AIoctl

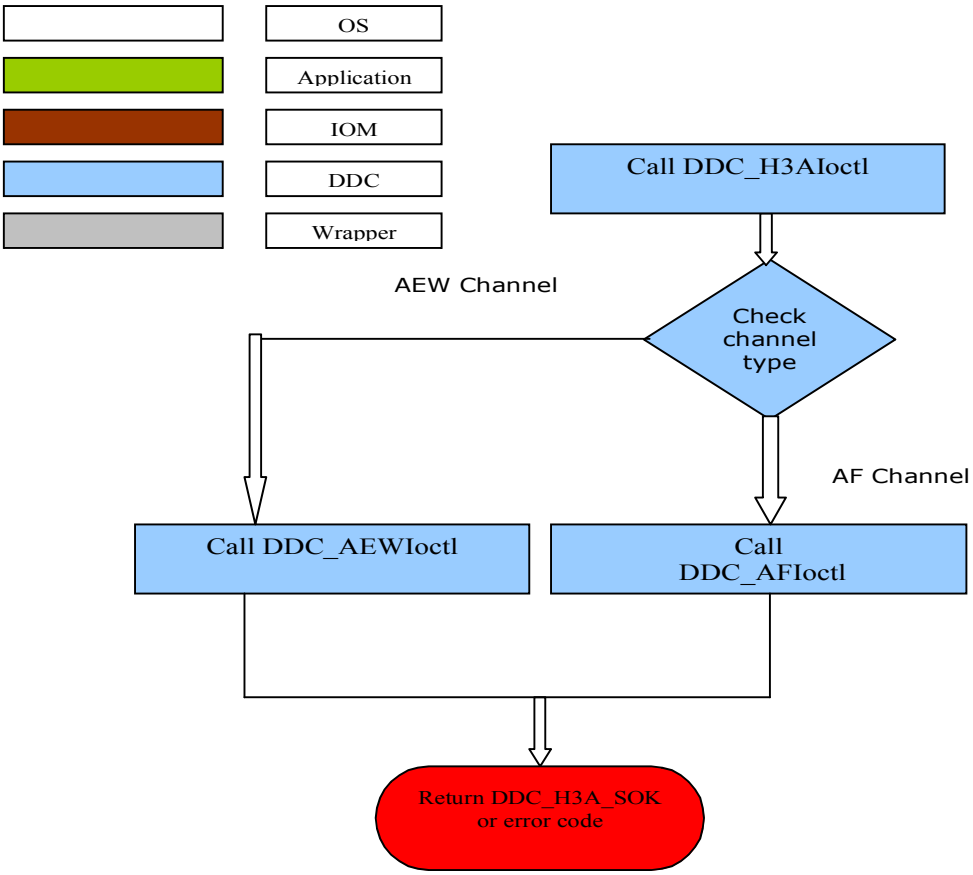

Figure 30. Control Command detail flow-1

DDC_AFIoctl

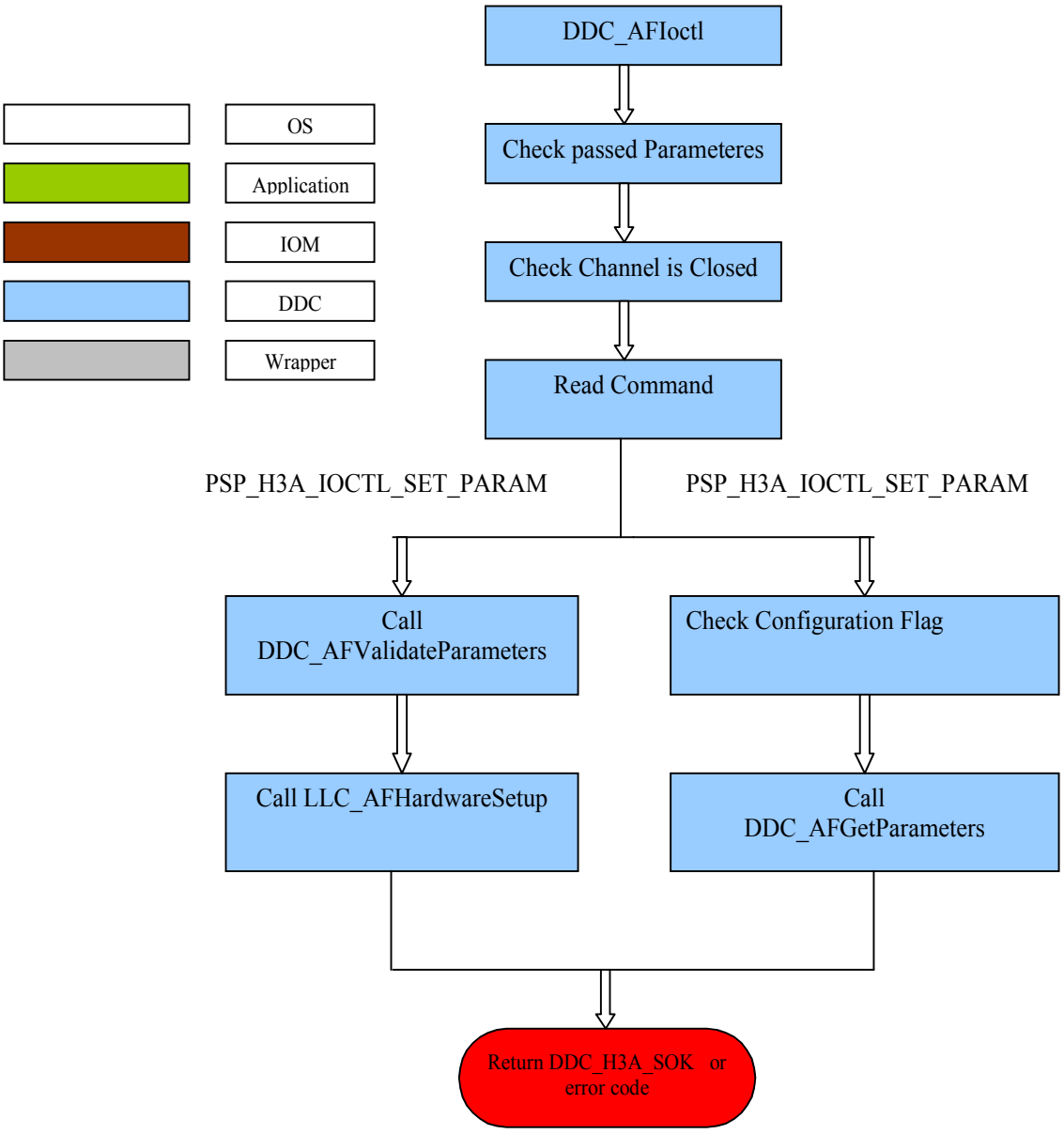

Figure 31. Driver close Channel detail flow -3 for AF Channel

DDC_AEWIoctl

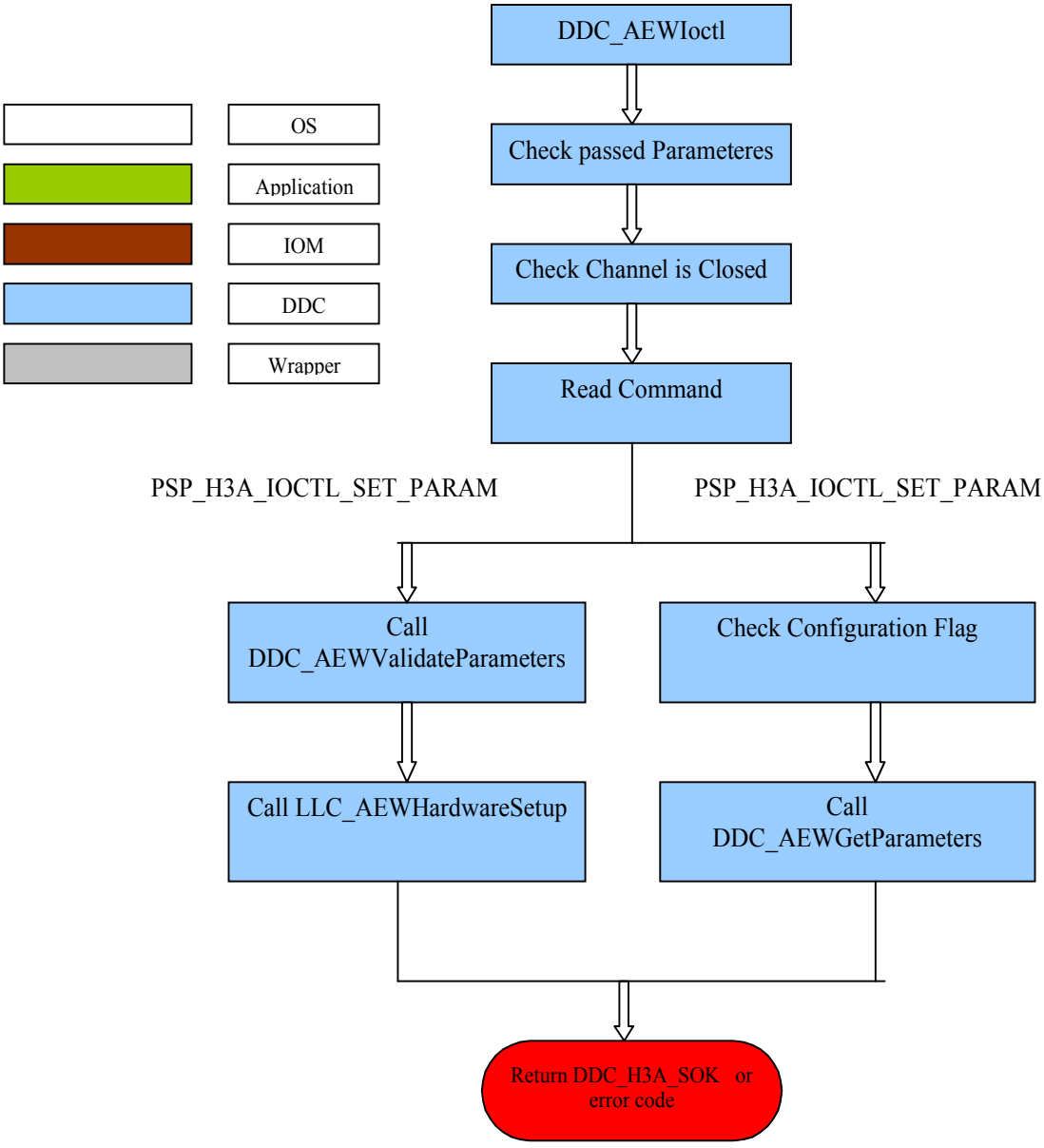

Figure 32. Driver close Channel detail flow -3 for AEW Channel

3.1.2.4 IO access

The Application will access H3A driver H3A_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.

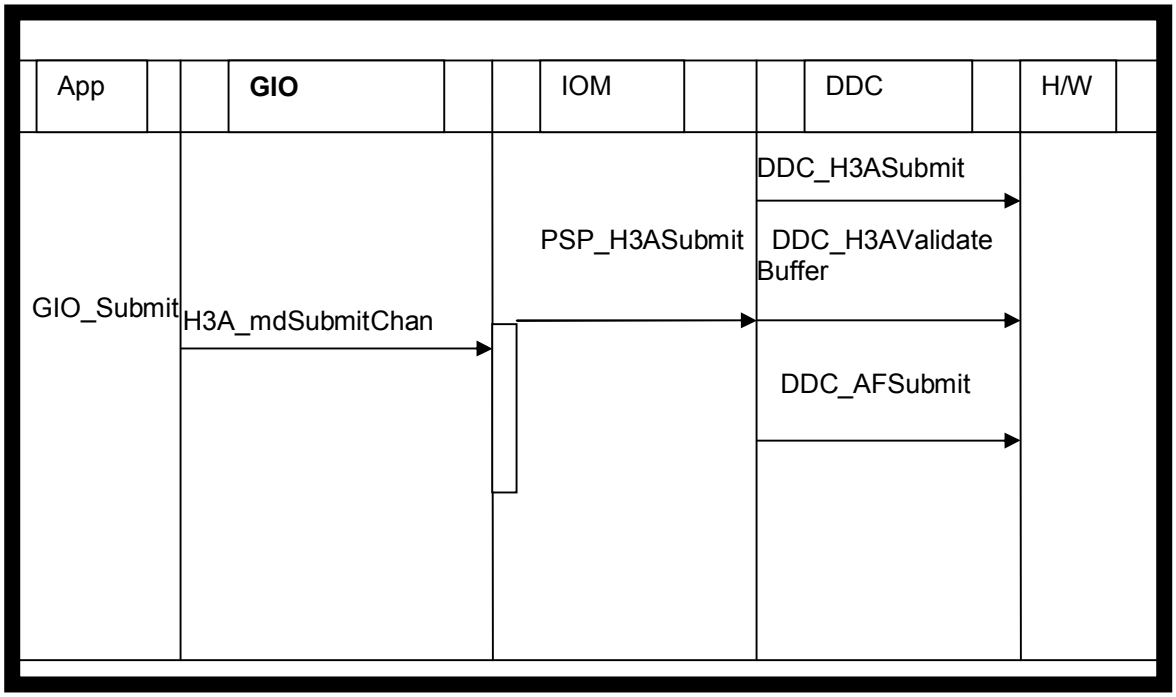

Figure 33. Driver Submit Overview for AF Channel

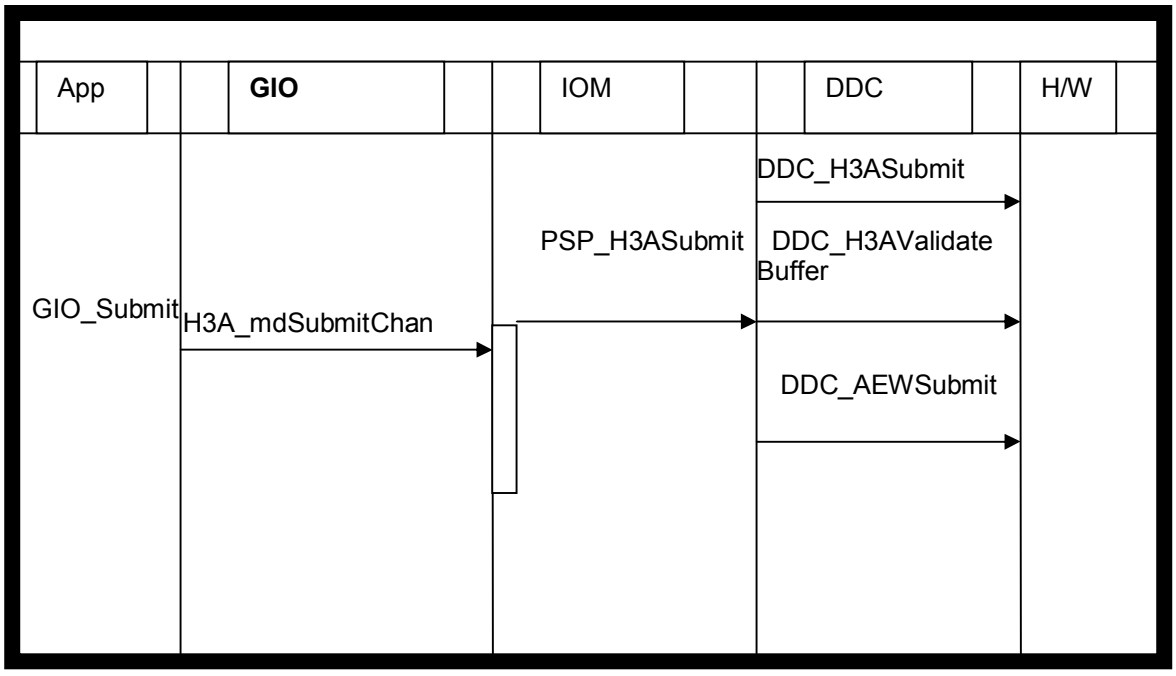

Figure 34. Driver Submit Overview for AEW Channel

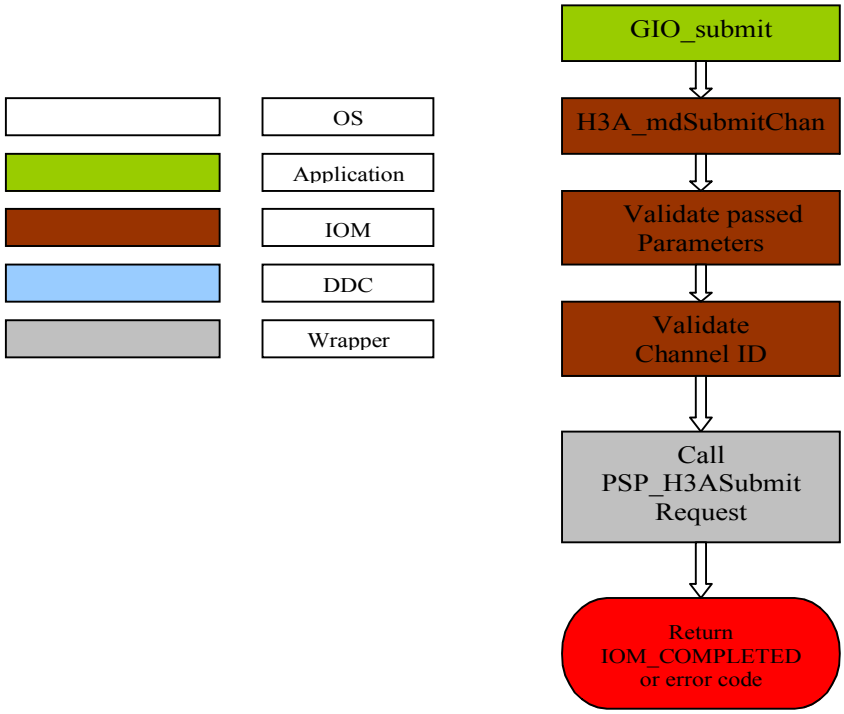

Figure 35. Driver Submit Detailed Flow Diagram – 1

PSP_H3ASubmitRequest

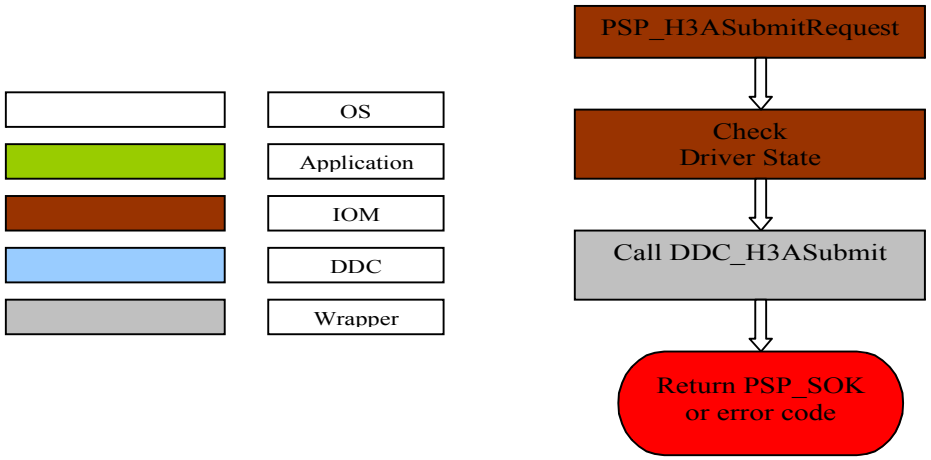

Figure 36. Driver Submit Detailed Flow Diagram – 1

DDC_H3ASubmit

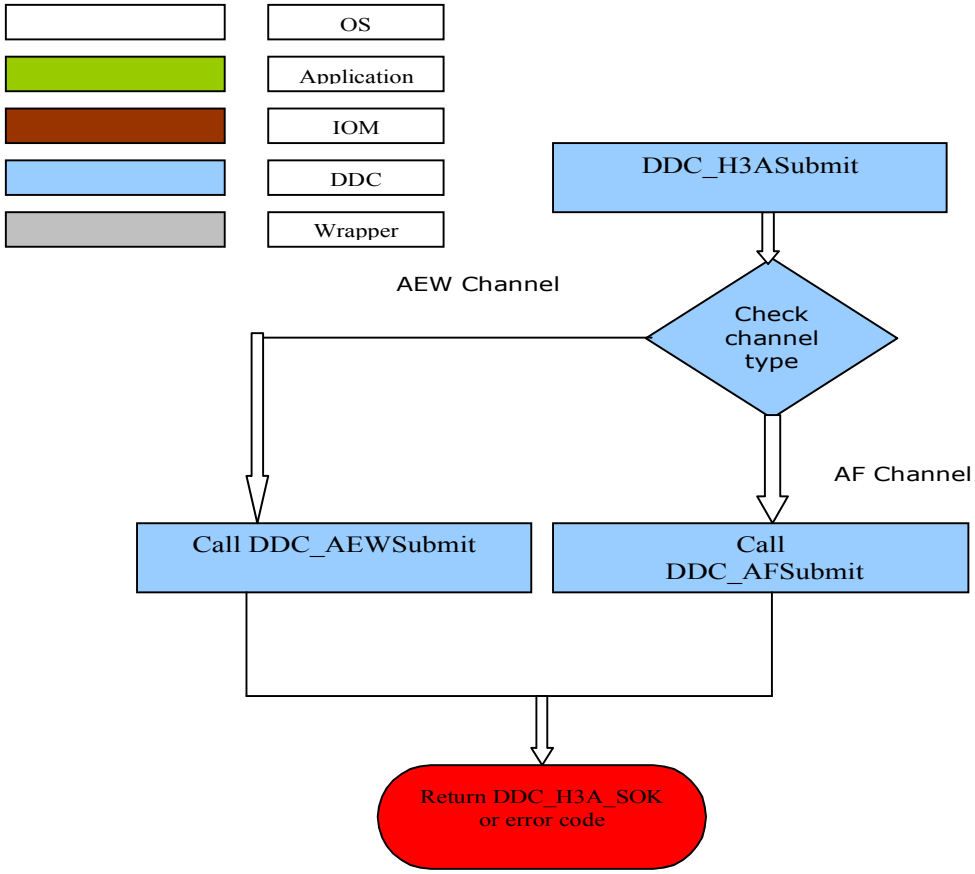

Figure 37. Driver Submit Detailed Flow Diagram – 2

DDC_AFSubmit

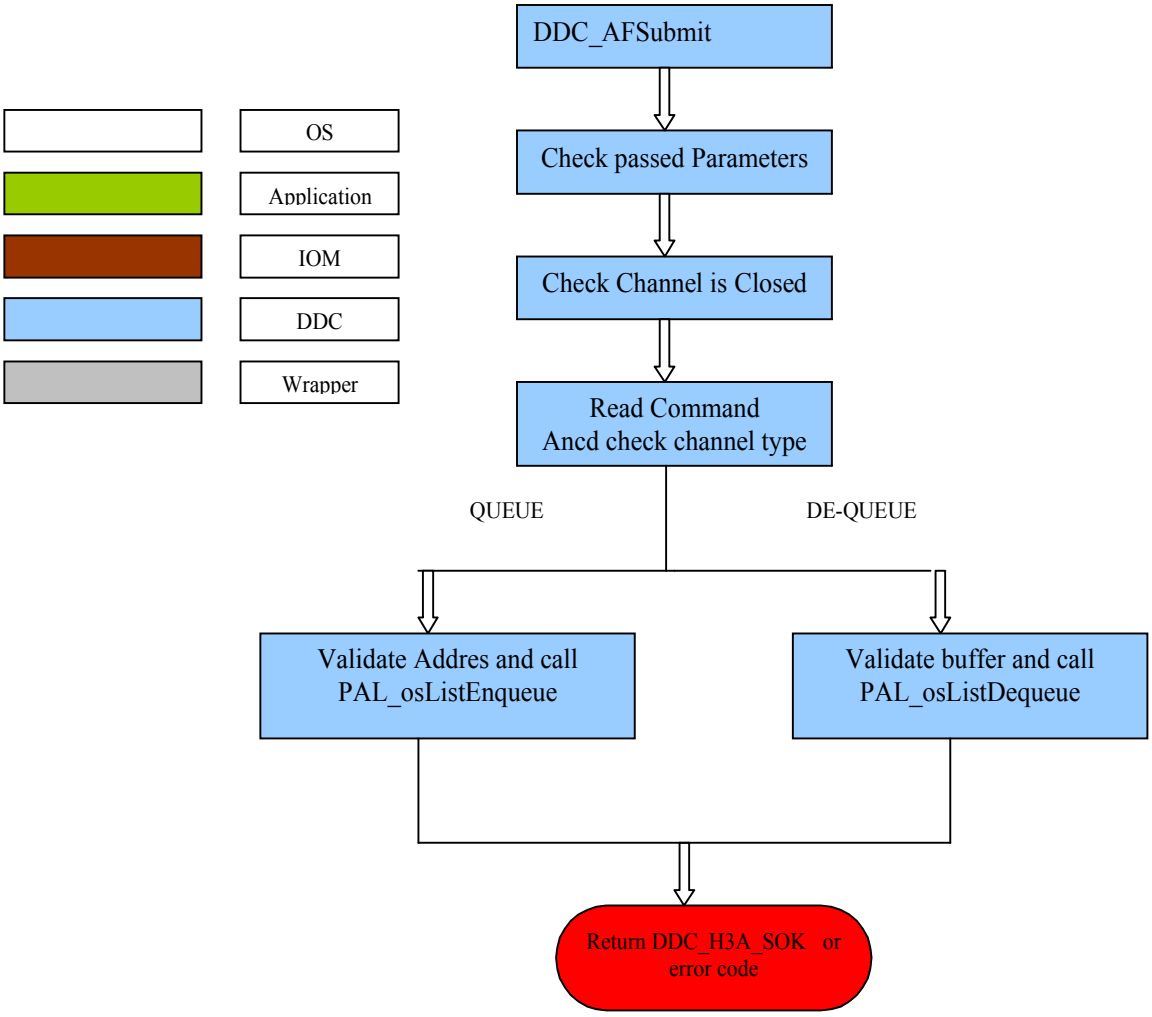

Figure 38. Driver Submit Detailed Flow Diagram – 2

DDC_AEWSubmit

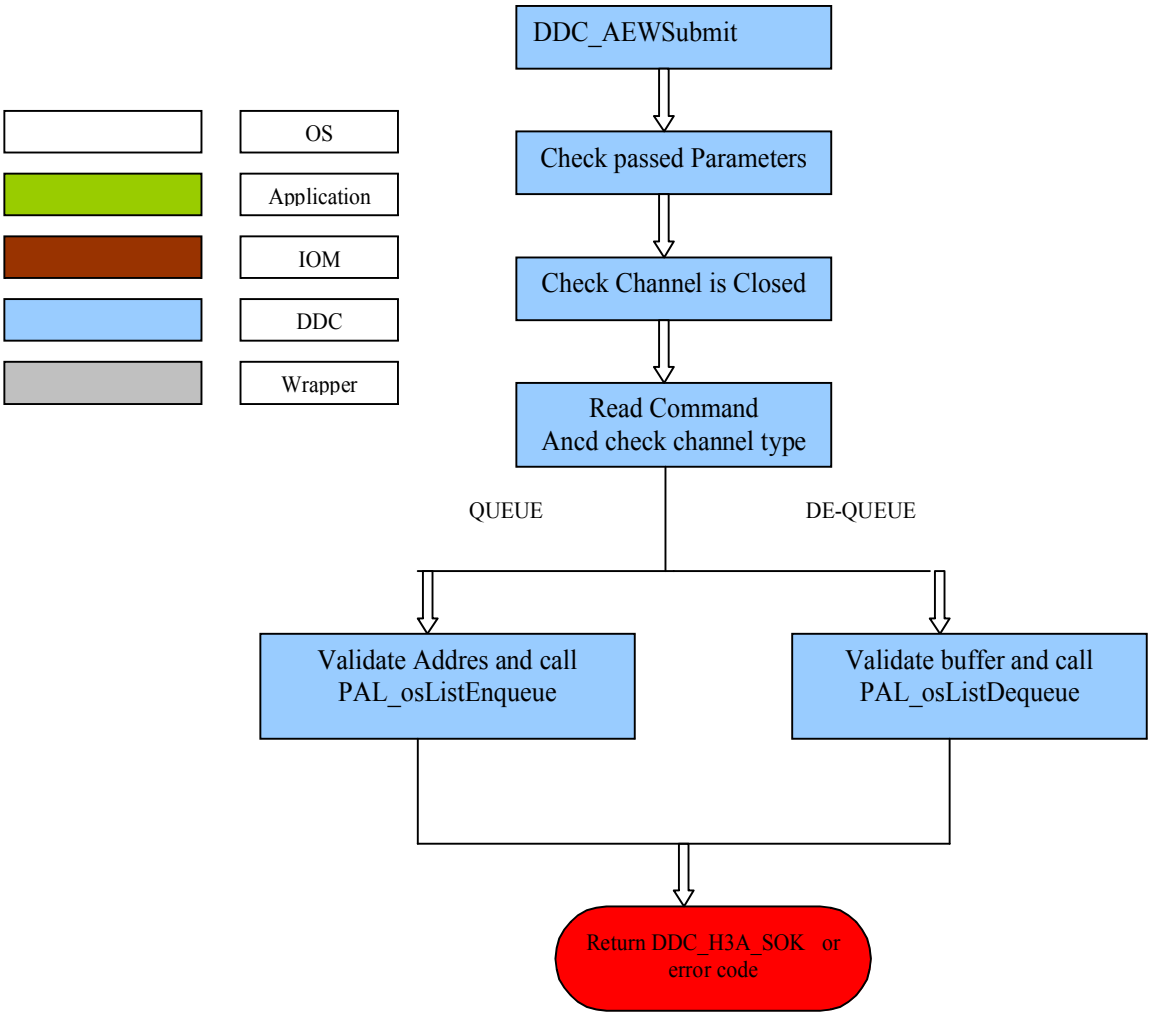

Figure 39. Driver Submit Detailed Flow Diagram – 2

DDC Layer Functions

3.1.2.5 Buffer management

When application puts a buffer in a queue for any of AF or AEW Channels, the driver will automatically enabled & start capturing frame. After completion of a frame the driver will copy the data into queued buffer and if another buffer is available in the in AF or AEW ACTIVE queue, interrupt will remain enabled otherwise it will be disabled. Enqueue/Dequeue will follow the FIFO mechanism.

3.1.2.6 Interrupt Service Routine

H3A driver Interrupt Handles will be attached with the H3A Hardware Interrupt of DM6437. H3A driver will implement interrupt dispatcher for AF and AEW Channels which will handle interrupts generated for the completion of a frame. On receipt of interrupt, this H3A interrupt dispatcher will read the busy bit of PCR register to verify the source of interrupt and copy the data into available queued empty 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_IOCTL_H3A_PARAMS command.

4 Low Level Definitions

This section describes low level details of H3A Driver.

4.1 Constants & Enumerations

CONSTANTS:

4.1.1 Number of Coefficients

Definition

```
#define PSP_AF_NUMBER_OF_COEF 11
```

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 H3A devices that can be created.

Definition

```
#define PSP_H3A_NUM_INSTANCES 1u
```

Comments

None

Constraints

Only one H3A device can be created.

4.1.3 Time out for semaphore

It describes the timeout value for semaphore in terms of millisecond.

Definition

```
#define DDC_H3A_TIMEOUT 150u
```

Comments

None

Constraints

None

4.1.4 Maximum Horizontal Count for Paxels for AF

Maximum no of paxels in the horizontal direction

Definition

```
#define DDC_AF_MAX_HZCOUNT (36u)
```

Comments

None

Constraints

Maximum no of paxels in the horizontal direction should not be greater than 36

4.1.5 Minimum Horizontal Count for Paxels for AF

Minimum no of paxels in the horizontal direction

Definition

```
#define DDC_AF_MIN_HZCOUNT (2u)
```

Comments

None

Constraints

The number of the paxels in horizontal direction must be 1 or greater.

4.1.6 Minimum Vertical Count for Paxels for AF

Minimum no of paxels in the vertical direction

Definition

```
#define DDC_AF_MIN_VTCOUNT (1u)
```

Comments

None

Constraints

The number of the paxels in vertical direction must be 1 or greater.

4.1.7 Maximum Vertical Count for Paxels for AF

Maximum no of paxels in the vertical direction

Definition

```
#define DDC_AF_MAX_VTCOUNT (128u)
```

Comments

None

Constraints

Maximum no of paxels in the vertical direction should not be greater than 128

4.1.8 Minimum Horizontal Start for AF

It describes the maximum number of H3A devices that can be created.

Definition

```
#define DDC_AF_MIN_HZSTART (2u)
```

Comments

None

Constraints

Minimum value of the horizontal start

4.1.9 Minimum width for AF

It describes the maximum number of H3A devices that can be created.

Definition

```
#define DDC_AF_MIN_WIDTH (6u)
```

Comments

None

Constraints

Minimum width

4.1.10 Maximum horizontal Start of Paxel for AF

Maximum horizontal start

Definition

#define DDC_AF_MAX_HZSTART (4094u)

Comments

None

Constraints

Maximum value of the horizontal start

4.1.11 Maximum width of paxel

Maximum width of paxel

Definition

```
#define DDC_AF_MAX_WIDTH      (256u)
```

Comments

None

Constraints

Maximum width of paxel

4.1.12 Minimum height of the paxel

Minimum height of paxel

Definition

#define DDC_AF_MIN_HEIGHT (2u)

Comments

None

Constraints

Minimum height of paxel.

4.1.13 Maximum height of the paxel

Maximum height of paxel

Definition

```
#define DDC_AF_MAX_HEIGHT       (256u)
```

Comments

None

Constraints

Maximum height of paxel.

4.1.14 Minimum Line Increment of the paxel

Minimum line increment of paxel

Definition

```
#define DDC_AF_MIN_LINE_INCR    (2u)
```

Comments

None

Constraints

Minimum line increment value for Paxel.

4.1.15 Maximum Line Increment of the paxel

Maximum Line increment

Definition

```
#define DDC_AF_MAX_LINE_INCR    (32u)
```

Comments

None

Constraints

Maximum Line increment value for Paxel.

4.1.16 Maximum vertical start of the paxel

Maximum vertical start

Definition

```
#define DDC_AF_MAX_VTSTART      (4095u)
```

Comments

None

Constraints

Maximum vertical start value for paxel.

4.1.17 Maximum IIR Filter Horizontal start Position

Maximum IIR Filter Horizontal Start Position

Definition

```
#define DDC_AF_MAX_IIRSH        (4094u)
```

Comments

None

Constraints

Maximum value of IIR Filter horizontal Start Position.

4.1.18 Maximum threshold value for Horizontal Median filter

Maximum threshold value for Horizontal Median Filter

Definition

```
#define DDC_AF_HMF_MEDTH_MAX    (255u)
```

Comments

None

Constraints

Maximum threshold value for Horizontal Median Filter.

4.1.19 Minimum value for IIR filter coefficients

Minimum value for IIR Filter Coefficients

Definition

```
#define DDC_AF_MIN_COEF         ((Int32)0xFFFFF800)
```

Comments

None

Constraints

Minimum value for IIR Filter Coefficients.

4.1.20 Maximum value for IIR filter coefficients

Maximum value for IIR Filter Coefficients

Definition

```
#define DDC_AF_MAX_COEF         ((Int32)0x000007FF)
```

Comments

None

Constraints

Maximum value for IIR Filter Coefficients.

4.1.21 Maximum Horizontal Count for Paxels for AEW

Maximum no of paxels in the horizontal direction

Definition

```
#define DDC_AEW_MAX_HZCOUNT (36u)
```

Comments

None

Constraints

Maximum no of paxels in the horizontal direction should not be greater than 36

4.1.22 Minimum Horizontal Count for Paxels for AEW

Minimum no of paxels in the horizontal direction

Definition

#define DDC_AEW_MIN_HZCOUNT (2u)

Comments

None

Constraints

The number of the windows in horizontal direction must be 2 or greater.

4.1.23 Minimum Vertical Count for Paxels for AEW

Minimum no of paxels in the vertical direction

Definition

```
#define DDC_AEW_MIN_VTCOUNT (1u)
```

Comments

None

Constraints

The number of the paxels in vertical direction must be 1 or greater.

4.1.24 Maximum Vertical Count for Windows for AEW

Maximum no of windows in the vertical direction

Definition

```
#define DDC_AEW_MAX_VTCOUNT (128u)
```

Comments

None

Constraints

Maximum no of windows in the vertical direction should not be greater than 128

4.1.25 Maximum horizontal Start of window

Maximum horizontal start

Definition

#define DDC_AEW_MAX_HZSTART (4095u)

Comments

None

Constraints

Maximum value of the horizontal start

4.1.26 Minimum width of window

Minimum width of window

Definition

```
#define DDC_AEW_MIN_WIDTH     (6u)
```

Comments

None

Constraints

Minimum width of window

4.1.27 Maximum width of window

Maximum width of window

Definition

```
#define DDC_AEW_MAX_WIDTH     (256u)
```

Comments

None

Constraints

Maximum width of window

4.1.28 Minimum height of the window

Minimum height of window

Definition

```
#define DDC_AEW_MIN_HEIGHT    (2u)
```

Comments

None

Constraints

Minimum height of window.

4.1.29 Maximum height of the window

Maximum height of window

Definition

```
#define DDC_AEW_MAX_HEIGHT       (256u)
```

Comments

None

Constraints

Maximum height of window.

4.1.30 Minimum height of the Black window

Minimum height of Black window

Definition

```
#define DDC_AEW_MIN_BLKWIN_HEIGHT   (2u)
```

Comments

None

Constraints

Minimum height of Black window.

4.1.31 Maximum height of the Black window

Maximum height of Black window

Definition

```
#define DDC_AEW_MAX_BLKWIN_HEIGHT   (256u)
```

Comments

None

Constraints

Maximum height of Black window.

4.1.32 Minimum Horizontal Line Increment of the window

Minimum Horizontal line increment of window

Definition

```
#define DDC_AEW_MIN_HZLINEINCR    (2u)
```

Comments

None

Constraints

Minimum Horizontal line increment value for window.

4.1.33 Maximum Horizontal Line Increment of the window

Maximum Horizontal Line increment

Definition

```
#define DDC_AEW_MAX_HZLINEINCR    (32u)
```

Comments

None

Constraints

Maximum Horizontal Line increment value for window.

4.1.34 Minimum Vertical Line Increment of the window

Minimum Vertical line increment of window

Definition

```
#define DDC_AEW_MIN_VTLINEINCR    (2u)
```

Comments

None

Constraints

Minimum Vertical line increment value for window.

4.1.35 Maximum Vertical Line Increment of the window

Maximum Vertical Line increment

Definition

```
#define DDC_AEW_MAX_VTLINEINCR    (32u)
```

Comments

None

Constraints

Maximum Vertical Line increment value for window.

4.1.36 Maximum vertical start of the window

Maximum vertical start

Definition

```
#define DDC_AEW_MAX_VTSTART      (4095u)
```

Comments

None

Constraints

Maximum vertical start value for window.

4.1.37 Maximum vertical start of the Black window

Maximum vertical start

Definition

```
#define DDC_AEW_MAX_BLKWIN_VTSTART  (4095u)
```

Comments

None

Constraints

Maximum vertical start value for Black window.

4.1.38 Maximum saturation limit of window

Maximum saturation limit of window

Definition

```
#define DDC_AEW_AVE2LMT_MAX   (1023u)
```

Comments

None

Constraints

Maximum saturation limit of window

4.1.39 Minimum saturation limit of window

Minimum vertical start

Definition

```
#define DDC_AEW_AVE2LMT_MIN   (0)
```

Comments

None

Constraints

Minimum saturation limit of window

4.1.40 Enable Value for Engine

Enable Value for Engine

Definition

```
#define DDC_H3A_ENGINE_ENABLE (1u)
```

Comments

None

Constraints

Maximum no of paxels in the vertical direction should not be greater than 128

4.1.41 Disable Value for Engine

Disable Value for Engine

Definition

```
#define DDC_H3A_ENGINE_DISABLE (0u)
```

Comments

None

Constraints

None

4.1.42 Busy Value for Engine

Busy Value for Engine

Definition

```
#define DDC_H3A_ENGINE_BUSY   (1u)
```

Comments

None

Constraints

Maximum no of paxels in the vertical direction should not be greater than 128

4.1.43 Status Value for Engine

Status Value indicates that engine is not busy

Definition

```
#define DDC_H3A_ENGINE_NOTBUSY (0u)
```

Comments

None

Constraints

None

4.1.44 Macro to check whether value is even or not

Macro to check whether value is even or not

Definition

```
#define DDC_H3A_ISEVEN(x) ((x) % (2u))
```

Comments

None

Constraints

None

ENUMARATORS:

H3A Enumerations

4.1.45 Controls Commands at DDA Layer

Describes various controls commands

Definition

```
typedef enum _PSP_H3AIoctlCmd
{
    PSP_H3A_IOCTL_SET_PARAM = 128,
    PSP_H3A_IOCTL_GET_PARAM
}PSP_H3AIoctlCmd;
```

Comments

None

Constraints

None

4.1.46 Status of data contained in the buffer

Status of the data contained in the buffer

Definition

```
typedef enum _PSP_H3ABufferStatus

{

    PSP_H3A_BUFFER_DATA_VALID = 0,

    /**< Get H3A config params, cmdArg = parameter structure    */

    PSP_H3A_BUFFER_DATA_CORRUPTED

    /**< Set H3A config params, cmdArg = parameter structure    */


} PSP_H3ABufferStatus;
```

Comments

None

Constraints

None

4.1.47 Parameters State

Describes state of the hardware configuration.

Definition

```
typedef enum _DDC_ParamsConfigDone
{
    STATE_NOT_CONFIGURED = 0,
    STATE_CONFIGURED

} DDC_ParamsConfigDone;
```

Comments

None

Constraints

None

4.1.48 Device state

Describes State of the device

Definition

```
typedef enum _PSP_H3A_DEVICE_STATE
{
      PSP_H3A_DELETED = 0,
      PSP_H3A_CREATED,
      PSP_H3A_OPENED,
      PSP_H3A_CLOSED
} PSP_H3AState;
```

Comments

None

Constraints

None

4.1.49 Error codes

Contains DDC related error codes that will be passed to the DDA layer

Definition

```
typedef enum _DDC_H3AStatus
{
      DDC_H3A_SOK=0,
      DDC_H3A_E_FAIL,
      DDC_H3A_INVALID_STATE,
      DDC_H3A_INVALID_PARAM,
      DDC_H3A_NOT_SUPPORTED,
      DDC_H3A_NO_DATA
```

```
} DDC_H3AStatus;
```

Comments

None

Constraints

None

4.1.50 Channel state

Describes State of the Channel

Definition

```
typedef enum _DDC_H3AChanState
{
      DDC_H3ACHANCLOSED = 0,
      DDC_H3ACHANOPENED

} DDC_H3AChanState;
```

Comments

None

Constraints

None

4.1.51 Channel Type

Describes type of the Channel

Definition

```
typedef enum _PSP_H3AChannelType
{
      PSP_H3A_AF = 0,
      PSP_H3A_AEW

} PSP_H3AChannelType;
```

Comments

None

Constraints

None

Auto Focus Enumerations

4.1.52 AF A law Enable/Disable

Describes status of A Law for AF.

Definition

```
typedef enum _PSP_AF_Alaw
{
      PSP_AF_ALAW_DISABLE     = 0,
```

```
        PSP_AF_ALAW_ENABLE      = 1
}PSP_AF_Alaw;
```

Comments

None

Constraints

None

4.1.53 Accumulator Mode

Describes mode of the accumulator

Definition

```
typedef enum _PSP_AF_mode
{
        PSP_AF_ACCUMULATOR_SUMMED  = 0,
        PSP_AF_ACCUMULATOR_PEAK    = 1
} PSP_AF_mode;
```

Comments

None

Constraints

None

4.1.54 Horizontal Median Filter Enable/Disable

Describes status of Horizontal Median Filter

Definition

```
typedef enum _PSP_AF_HMF_law
{
        PSP_AF_HMF_DISABLE  = 0,
        PSP_AF_HMF_ENABLE   = 1
} PSP_AF_HMF_law;
```

Comments

None

Constraints

None

4.1.55 RGB Position

Describes RGB Position

Definition

```
typedef enum _PSP_AF_rgbpos
{
        PSP_AF_GR_GB_BAYER            =   0,
        PSP_AF_RG_GB_BAYER            =   1,
        PSP_AF_GR_BG_BAYER            =   2,
        PSP_AF_RG_BG_BAYER            =   3,
```

```
        PSP_AF_GG_RB_CUSTOM             =    4,
        PSP_AF_RB_GG_CUSTOM             =    5
}PSP_AF_rgbpos;
```

Comments

None

Constraints

None

Auto Exposure / Auto White Balance Enumerations

4.1.56 AEW A law Enable/Disable

Describes status of A Law for AEW.

Definition

```
typedef enum _PSP_AEW_Alaw
{
        PSP_AEW_ALAW_DISABLE    = 0,
        PSP_AEW_ALAW_ENABLE     = 1
} PSP_AEW_Alaw;
```

Comments

None

Constraints

None

4.2 Typedefs & Data Structures

Interface level structures

H3AStructure

4.2.1 H3A Buffer

This structure contains buffer pointer & timestamp information.

Definition

```
typedef struct _PSP_H3ABuffer{
      PAL_OsListNodeHeader    nodeEntry;
      Ptr                     ramIpAddr;
      Uint                    timeStamp;
      Uint                    size;
      PSP_H3ABufferStatus     buffStatus;

} PSP_H3ABuffer;
```

Fields

nodeEntry	Node for queue entry
ramIpAddr	Buffer address to store data
timeStamp	Tells the time(Milliseconds) when data was read from H3A
size	Size of the enqueued buffer
buffStatus	Status of the buffer containing statistics

Comments

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

Constraints

None

Auto Focus Structure

4.2.2 Horizontal Median Filter Parameters

This structure is used to configure Horizontal Median Filter enhancement parameters

Definition

```
typedef struct _PSP_AFHmf
{
    PSP_AF_HMF_law            enable;
    Uint                      threshold;
} PSP_AFHmf;
```

Fields

Enable	status of Horizontal Median Filter

Threshold	Threshold Value for Horizontal Median Filter

Comments

If PSP_AF_HMF_law type is disabled than all others fields are ignored.

Constraints

None.

4.2.3 IIR Filer Parameters

This structure is used to configure the various IIR Filter parameters

Definition

```
typedef struct  _PSP_AFIir
{
      Uint    hzStarPos;
      Int     coeffSet0[PSP_AF_NUMBER_OF_COEF];
      Int     coeffSet1[PSP_AF_NUMBER_OF_COEF];
} PSP_AFIir;
```

Fields

hzStarPos	IIR Start Register Value
coeffSet0	IIR Filter Coefficient for Set 0
coeffSet1	IIR Filter Coefficient for Set 1

Comments

All parameters should be configured.

Constraints

None

4.2.4 Paxel Structure

This structure is used to configure paxel parameters

Definition

```
typedef struct _PSP_AFPaxel
{
      Uint    width;
      Uint    height;
      Uint    hzStart;
      Uint    vtStart;
      Uint    hzCnt;
      Uint    vtCnt;
      Uint    lineIncr;
} PSP_AFPaxel;
```

Fields

width	Width of the Paxel

height	Height of the Paxel
hzStart	Horizontal Start Position
vtStart	Vertical Start Position
hzCnt	Horizontal Count
vtCnt	vertical Count
lineIncr	line Increment

Comments

All parameters should be configured

Constraints

None

4.2.5 AF Parameter Structure

This structure is used to configure AF parameters

Definition

```
typedef struct  _PSP_AFParams
{
      PSP_AF_Alaw       enable;
      PSP_AFHmf         hmfConfig;
      PSP_AF_rgbpos     rgbPos;
      PSP_AFIir         iirConfig;
      PSP_AFPaxel       paxelConfig;
      PSP_AF_mode       mode;
} PSP_AFParams;
```

Fields

enable	ALAW status
hmfConfig	HMF configurations
rgbPos	RGB Positions
iirConfig	IIR filter configurations
paxelConfig	Paxel parameters
mode	Accumulator mode

Comments

All parameters should be configured

Constraints

None

Auto Exposure/Auto White Balance

4.2.6 Window Structure

This structure is used to configure window parameters

Definition

```
typedef struct _PSP_AEWWindow
{
      Uint    width;
      Uint    height;
      Uint    hzStart;
      Uint    vtStart;
      Uint    hzCnt;
      Uint    vtCnt;
      Uint    hzLineIncr;
      Uint    vtLineIncr;
} PSP_AEWWindow;
```

Fields

`width`	Width of the window
`height`	Height of the window
`hzStart`	Horizontal Start Position
`vtStart`	Vertical Start Position
`hzCnt`	Horizontal Count
`vtCnt`	vertical Count
`hzLineIncr`	Horizontal line Increment
`vtLineIncr`	Horizontal line Increment

Comments

All parameters should be configured.

Constraints

None

4.2.7 Black Window Structure

This structure is used to configure black window parameters

Definition

```
typedef struct _PSP_AEWBlkWindow
{
      Uint    height;
      Uint    vtStart;
 } PSP_AEWBlkWindow;
```

Fields

height	Height of the window
vtStart	Vertical Start Position

Comments

All parameters should be configured.

Constraints

None

4.2.8 AEW Parameters Structure

This structure is used to configure AEW parameters.

Definition

```
typedef struct _PSP_AEWParams
{
      PSP_AEW_Alaw            enable;
      Int                     satLimit;
      PSP_AEWWindow           winConfig;
      PSP_AEWBlkWindow        blkWinConfig;
} PSP_AEWParams;
```

Fields

enable	ALAW status
satLimit	Saturation Limit
winConfig	Window Configurations
blkWinConfig	Black Window Configurations

Comments

All parameters should be configured

Constraints

None

IOM Layer Structures

H3A Structures

4.2.9 IOM Layer H3A Channel Object

This structure is used to describe channel structure for H3A.

Definition

```
typedef struct H3A_IOMCHANNELOBJ
{
      H3APortObject           *port;
      PSP_Handle              ddc_Handle;
} H3AChannelObject;
```

Fields

Port	Reverse pointer to Port
ddc_Handle	Handle for the channels

Comments

None

Constraints

None

Global structure

4.2.10 IOM Layer Port Structure

This structure is used to describe Port structure for H3A driver.

Definition

```
typedef struct H3APORTObj
{
      Uint                    portNumber;
      DDC_H3ADeviceObject     *deviceObj;
      PSP_H3AState            state;
      H3AChannelObject        afChan;
      H3AChannelObject        aewChan;
} H3APortObject;
```

Fields

portNumber	Instance number
deviceObj	Pointer to device Object
state	state of driver
afChan	Pointer to AF channel Object
aewChan	Pointer to AEW Channel Object

Comments

None

Constraints

None

DDC Layer Structures

H3A Structures

Global Structure

4.2.11 DDC Layer H3A Device Structure

This structure is used to describe device structure for H3A.

Definition

```
typedef struct _DDC_H3ADeviceObject
{
      Uint                    instanceId;
      Uint                    intNum;
      PSP_H3AState            state;
      Ptr                     regs;
      DDC_AFChannelObject     chanAF;
      DDC_AFChannelObject     chanAEW;
} DDC_H3ADeviceObject;
```

Fields

instanceId	Device Instance
intNum	Interrupt number
Sate	state of H3A Device
regs	Pointer to H3A Registers
chanAF	AF Channel Object
chanAEW	AEW Channel Object

Comments

None

Constraints

None

4.2.12 DDC Layer Buffer Management Structure

This structure is used to describe Queue Structure.

Definition

```
typedef struct _DDC_H3ABufferobject{
      Uint       indexEnQueue;
      Uint       indexDeQueue;
```

```
        PAL_OsListNodeHeader    enQueue;
        PAL_OsListNodeHeader    deQueue;
} DDC_H3ABufferobject;
```

Fields

indexEnQueue	Index indicating the no of buffer pushed on the queue
indexDeQueue	Index indicating the no of buffers containing statistics
enQueue	Queue which keeps track of the buffer sent by the application
deQueue	Queue which keeps track of the buffer containing statistics

Comments

None

Constraints

None

Channel Structure

AutoFocus Structure

4.2.13 DDC Layer AF Channel Structure

This structure is used to describe device structure for AF Engine.

Definition

```
typedef struct _DDC_AFChannelObject
{
      Uint                          channelId;
      DDC_H3AChanState              state;
      PSP_AFParams                  params;
      PAL_OsSemHandle               sem;
      DDC_H3A BufferObject          buffObj;
      DDC_ParamsConfigDone          configFlag;
      Uint                          buffSize;
      PAL_OsSemHandle               semISR;
} DDC_AFChannelObject;
```

Fields

channelId	Channel id for AF object
state	state for the channel
params	Handle for access the AF config parameters
sem	Synchronizing Object
buffObj	Handle for accessing the buffer object
buffSize	Size of the buffer
configFlag	configuration Flag to indicate whether set up is done

semISR	Semaphore for Dequeue

Comments

None

Constraints

None

AutoExposure/White Balance Structure

4.2.14 DDC Layer AEW Channel Structure

This structure is used to describe device structure for AEW Engine.

Definition

```
typedef struct _DDC_AEWChannelObject
{
      Uint                          channelId;
      DDC_H3AChanState              state;
      PSP_AEWParams                 params;
      PAL_OsSemHandle               sem;
      DDC_H3ABufferObject           buffObj;
      Uint                          buffSize;
      DDC_ParamsConfigDone          configFlag;
      PAL_OsSemHandle               semISR;

} DDC_AEWChannelObject;
```

Fields

channelId	Channel id for AF object
state	state for the channel
params	Handle for access the AF config parameters
sem	Synchronizing Object
buffObj	Handle for accessing the buffer object
buffSize	Size of the buffer
configFlag	configuration Flag to indicate whether set up is done
semISR	Semaphore for Dequeue

Comments

None

Constraints

None

4.3 API Definition

4.3.1 GIO_CREATE

Syntax

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

Arguments

IN String name

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

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. The Channel Parameters will determine the type of the channel. The application should pass the channel type.

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 H3A 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.

4.3.4 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 points 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 structure. 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.

The behavior of submit call for the ENQUEUE request is as follows.

- If the application issues ENQUEUE request with the buffer size less than size of available statistics, then the driver will return error.

The behavior of submit call for the DEQUEUE request is as follows.

- If the application issues DEQUEUE request without performing ENQUEUE operation driver will return error.
- If the application issues DEQUEUE request after performing ENQUEUE operation the driver will block the DEQUEUE request for some time. If statistics are available before time out occurs, then driver will return the statistics to the application otherwise error will be returned.

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 H3A_mdBindDev

Function	H3A_mdBindDev()
Function Prototype	Int H3A_mdBindDev (Ptr *devp, Int devid, Ptr devParams)
Input Parameters	devid – number of device instances devParams – would be the H/W configuration information pointer variable. It will be NULL. 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 two instances. 2) The driver should be opening for the first time at the OS initialization time.
Design	Logic in steps. 1) Check the number of instance 2) set the initial global port values to zero 3) call PSP_H3ACreate function 4) configure devp

4.4.2 H3A_mdCreateChan

Function	H3A_mdCreateChan
Function Prototype	Int H3A_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. The name will differentiate between the AF and AEW driver. mode – mode of the driver i.e. INPUT or OUTPUT chanParams – additional parameters needed to initialize the channel cbFxn – callback function to the GIO cbArg – callback arguments
Output Parameters	Int according to the IOM errors
Description	This function 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 depending on the channel type. 4) Call PSP_H3AOpen function. 5) Update the port state

4.4.3 H3A_mdControlChan

Function	H3A_mdControlChan
Function Prototype	Int H3A_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 input parameters 2) Call a function PSP_H3A_Ioctl to Execute the IOCTL command

4.4.4 H3A_mdSubmitChan

Function	mdSubmitChan
Function Prototype	static Int H3A_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 would result in the call to the various other APIs depending upon the command issued the packet.
Preconditions	The device driver state should be opened.
Design	Logic in steps 1) Check submit request 2) Call Function PSP_H3ASubmit

4.4.5 H3A_mdUnbindDev

Function	H3A_mdUnbindDev

Function Prototype	Int H3A_mdUnBindDev(Ptr devp)
Input Parameters	Devp – Handle to H3A 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) Check the state of driver. 2) Check devp. 3) Call the PSP_H3ADelete function.

4.4.6 H3A_mdDeleteChan

Function	H3A_mdDeleteChan
Function Prototype	Int H3A_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 function of DDC Layer using Function Table.
Preconditions	The driver should be opened
Design	Logic in steps 1) Check the State of the port. 2) Call PSP_H3A_close Function

4.4.7 PSP_H3AOpen

Function	PSP_H3AOpen

Function Prototype	PSP_Handle PSP_H3AOpen (PSP_H3AChannelType Chantype)
Input Parameters	Chantype - Channel Type
Output Parameters	Int according to the PSP errors
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 Input Parameters 2) Check the State of the Driver 3) Call DDC_H3AOpenHandle

4.4.8 PSP_H3AClose

Function	PSP_H3AClose
Function Prototype	PSP_Result PSP_H3AClose(PSP_Handle handle)
Input Parameters	PSP_Handle – Handle to close the channel
Output Parameters	Int according to the IOM error code
Description	This function would remove the channel instance.
Preconditions	Driver must be in created State. Channel should be opened.
Design	Logic in steps 1) Check the State of the Channel 2) Call DDC_H3ACloseHandle

4.4.9 PSP_H3Aloctl

Function	PSP_H3AIoctl
Function Prototype	PSP_Result PSP_H3AIoctl(PSP_Handle handle,PSP_H3AIoctlCmd 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 input parameters 2) Check Channel is Closed or not. 3) Call DDC_H3AIoctl Function

4.4.10 PSP_H3ASubmit

Function	PSP_H3ASubmit
Function Prototype	PSP_Result PSP_H3ASubmit(PSP_Handle handle,PSP_VPSSSubmitCommand cmd, Ptr cmdArg);
Input Parameters	handle-Pointer to channel handle Cmd –Submit Command CmdArg – Pointer to 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 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_H3ASubmit

4.4.11 PSP_H3ACreate

Function	PSP_H3ACreate
Function Prototype	PSP_Handle PSP_H3ACreate(Uint devid)
Input Parameters	Devid - Device Id
Output Parameters	Int according to the PSP errors
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 DDC_H3AInitializeModule depending on the device id 5) Call DDC_H3ARegisterOverlaying Function. 6) Call DDC_H3ASetInterruptNumber function. 7) Call DDA_H3ARegisterIntrHandler

4.4.12 PSP_H3ADelete

Function	PSP_H3ADelete
Function Prototype	PSP_Result PSP_H3ADelete(Uint devid)
Input Parameters	Int –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 Input Parameters 2) Call DDC_H3ARemoveModule Function 3) Call DDA_H3AUnRegisterIntrHandler

4.4.13 DDA_H3ARegisterIntrHandler

Function	_DDA_H3ARegisterIntrHandler
Function Prototype	void _DDA_H3ARegisterIntrHandler(Uint intrNum);
Input Parameters	intrNum - Interrupt Number
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

4.4.14 DDA_H3AUnRegisterIntrHandler

Function	DDA_H3AUnRegisterIntrHandler
Function Prototype	void DDA_H3AUnRegisterIntrHandler(Uint intrNum);
Input Parameters	intrNum - Interrupt Number

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

H3A Functions

4.5.1 DDC_H3AInitailizeModule

Function	DDC_H3AInitializeModule
Function Prototype	Void * DDC_H3AInitializeModule (Uint devid)
Input Parameters	Device ID
Output Parameters	Pointer to DDC layer device structure
Description	It will perform the register overlaying and set the interrupt number and update the state of device.
Preconditions	None
Design	Logic in steps 1) Update the state of driver 2) Return the H3A device handle

4.5.2 DDC_H3ARemoveModule

Function	DDC_H3ARemoveModule
Function Prototype	Void * DDC_H3ARemoveModule (Uint devid)
Input Parameters	devid- Devide ID
Output Parameters	Pointer to DDC layer device structure
Description	It will update the update the state of device.
Preconditions	None
Design	Logic in steps 1) Update the state of the device driver 2) Return the H3A device handle

4.5.3 DDC_H3AValidateBuffer

Function	DDC_H3AValidateBuffer
Function Prototype	DDC_H3AStatus DDC_H3AValidateBuffer((Ptr cmdArg,PSP_VPSSSubmitCommand cmd)
Input Parameters	handle-Pointer to channel handle Cmd –Submit Command CmdArg – Pointer to Command Argument
Output Parameters	NULL
Description	Driver will validate the buffers.
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps 1) Check the buffer Parameters.

4.5.4 DDC_H3ASetInterruptNumber

Function	DDC_H3ASetDeviceInterruptNumber
Function Prototype	void DDC_H3ASetDeviceInterruptNumber()
Input Parameters	NULL
Output Parameters	NULL
Description	It will assign the interrupt number to the H3A driver.
Preconditions	Driver must not be in opened state.
Design	Logic in steps 1) Get the H3A device handle. 2) Assign the interrupt number to the IntNum member of the device handle.

4.5.5 DDC_H3AGetDeviceHandle

Function	DDC_H3AGetDeviceHandle
Function Prototype	Void * DDC_H3AGetDeviceHandle(void);
Input Parameters	NULL

Output Parameters	NULL
Description	It will return the handle for H3A.
Preconditions	Driver must be in opened state.
Design	Logic in steps 1) Return the device object.

4.5.6 DDC_H3AOpenHandle

Function	DDC_H3AOpenHandle
Function Prototype	PSP_Handle DDC_H3AOpenHandle(PSP_H3AChannelType ChannelId)
Input Parameters	channelId - Channel Id
Output Parameters	Return channel handle
Description	It will return the Channel handle.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Check channel type 2) Call the corresponding open Function 3) It will return the channel handle

4.5.7 DDC_H3ACloseHandle

Function	DDC_H3ACloseHandle
Function Prototype	DDC_H3AStatus DDC_H3ACloseHandle(PSP_Handle handle);
Input Parameters	Channel Id
Output Parameters	NULL
Description	It will close the channel.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Check channel type 2) Call the corresponding close Function

4.5.8 DDC_H3ARegisterOverlaying

Function	DDC_H3ARegisterOverlaying
Function Prototype	void DDC_H3ARegisterOverlaying(void)
Input Parameters	NULL
Output Parameters	NULL
Description	It will Perform register overlaying.
Preconditions	Driver must not be already opened.

Design	Logic in steps 1) Get the H3Adevice handle. 2) Assign the base address H3A register 3) Reset all the H3A register 4) Clear write buffer Overflow bit

4.5.9 DDC_H3Aisr

Function	DDC_H3Aisr
Function Prototype	Void DDC_H3Aisr(void)
Input Parameters	NULL
Output Parameters	NULL
Description	It will be called when AF or AEW completes processing of one frame.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Check the BUSYAF and BUSYAEW 2) If BUSYAF is 0 and AF_EN is 1 then call DDC_AFisr 3) If BUSYAEW is 0 and AEW_EN is 1 then call DDC_AEWisr

4.5.10 DDC_H3Aioctl

Function	DDC_H3AIoctl
Function Prototype	DDC_H3AStatus DDC_H3AIoctl(PSP_Handle handle, PSP_H3AIoctlCmd cmd, Ptr cmdArg, Ptr params);

Input Parameters	Handle - Pointer to Channel Object Cmd - Control Command cmd-arg - Pointer to Command Argument Params – pointer to params structure
Output Parameters	NULL
Description	Function to execute the control command.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps : 1) Check Channel Type 2) Call the corresponding function

4.5.11 DDC_H3ASubmit

Function	DDC_H3ASubmit
Function Prototype	DDC_H3AStatus DDC_H3ASubmit(PSP_Handle handle,PSP_VPSSSubmitCommand cmd, Ptr cmdArg);
Input Parameters	handle-Pointer to channel handle Cmd –Submit Command CmdArg – Pointer to 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) Check Channel Type 2) Call the corresponding function

4.5.12 DDC_H3AIsChannelHandleClosed

Function	DDC_H3AIsChannelHandleClosed
Function Prototype	DDC_H3AStatus DDC_H3AIsChannelHandleClosed (PSP_Handle handle);
Input Parameters	Handle – Channel Handle
Output Parameters	NULL
Description	It will return the status of the handle. If the handle is opened it will return DDC_H3A_SOK else it will return DDC_H3A_E_FAIL.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Check the channel type 2) Call the corresponding function

Auto Focus Functions

4.5.13 DDC_AFGetChannelHandle

Function	DDC_AFGetChannelHandle

Function Prototype	PSP_Handle DDC_AFGetChannelHandle(void);
Input Parameters	NULL
Output Parameters	Channel Handle
Description	It will return the Channel handle for AF.
Preconditions	Driver must be in opened state.
Design	Logic in steps 1) Return the channel specific object.

4.5.14 DDC_AFSubmit

Function	DDC_AFSubmit
Function Prototype	DDC_H3AStatus DDC_H3ASubmit(PSP_Handle handle,PSP_VPSSSubmitCommand cmd, Ptr cmdArg);
Input Parameters	handle-Pointer to channel handle Cmd –Submit Command CmdArg – Pointer to 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) Lock sem 2) Call DDC_H3AValidateBuffer 3) Read the command 4) if command is PSP_VPSS_QUEUE then perform the enqueue operation using pal layer Function. 5) Check whether address is 64 bit aligned or not 6) If the enqueue contains buffer enable the engine 7) if command is PSP_VPSS_DEQUEUE then perform the dequeue operation using pal layer Function. 8) If the buffer is queued or the statistics are available block ISR semaphore 9) Unlock the sem

4.5.15 DDC_AFOpenHandle

Function	DDC_AFOpenHandle
Function Prototype	Void * DDC_AFOpenHandle(void);
Input Parameters	NULL
Output Parameters	NULL
Description	It will return the Channel handle and update the state of AF channel.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Call the function DDC_AFGetDeviceHandle 2) Initialize the channel configuration structure 3) Initialize the queue 4) Create the Channel semaphore 5) Update the Channel state to OPENED 6) Reset all the AF registers 7) Return Channel Handle

4.5.16 DDC_AFIsChannelHandleClosed

Function	DDC_AFIsChannelHandleClosed
Function Prototype	DDC_H3AStatus DDC_AFIsChannelHandleClosed (PSP_Handle 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_H3A_SOK else it will return DDC_H3A_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

4.5.17 DDC_AFValidateParameters

Function	DDC_AFValidateParameters
Function Prototype	DDC_H3AStatus DDC_AFValidateParameters(PSP_AFParams *)
Input Parameters	Pointer to PSP_AFParams
Output Parameters	NULL
Description	It will Validate the af 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) Check the range of all parameters 2) Call LLCAFHardwareSetup function

4.5.18 DDC_AFGetParameters

Function	DDC_AFGetParameters
Function Prototype	DDC_H3AStatus DDC_AFGetParameters(DDC_AFChannelObject *, PSP_AFParams *)
Input Parameters	Pointer to DDC_AFChannelObject Pointer to PSP_AFParams
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 AF parameters from the channel specific structure.

4.5.19 DDC_AFCloseHandle

Function	DDC_H3ACloseHandle
Function Prototype	DDC_H3AStatus DDC_H3ACloseHandle(PSP_Handle handle);
Input Parameters	Pointer to Channel 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_AFGetDeviceHandle. 2) Free the channel configuration structure 3) Delete the Channel semaphore 4) Reset the queues 5) Update the Channel state to CLOSED

4.5.20 DDC_AFisr

Function	DDC_AFisr
Function Prototype	Void DDC_AFisr(void)
Input Parameters	NULL
Output Parameters	NULL
Description	It will be called when AF or AEW completes processing of one frame.
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps 1) Check if the "ENQUEUE" is empty 2) If there is no buffer disable the engine 3) If there is buffer in ENQUEUE Get next buffer from the enQueue. Set AFBUFST Register to that buffer address. 4) Dequeue first buffer from ENQUEUE 5) Set TimeStamp 6) move the first buffer from enQueue to dequeue queue. 7) Decrement enqueue index 8) Increment the dequeue index 9) Unblock the ISR Semaphore 10) Check the write overflow bit and set the buffer status 11) Call LLC_AFClearWriteBuffer

4.5.21 DDC_AFIoctl

Function	DDC_H3AQueueFrameBuffer
Function Prototype	DDC_H3AStatus DDC_AFIoctl(PSP_Handle handle,PSP_H3AIoctlCmd cmd, Ptr cmdArg,Ptr params);
Input Parameters	Pointer to Channel Object Control Command Command Argument Params
Output Parameters	NULL
Description	Function to execute the control command.
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps : 1) Lock on Sem 2) Read the command. 3) Call the Functions that will execute the command 4) Unlock Sem E.g. For H3A_SET_PARAM will call DDC_AF_ValidateParameters. H3A_GET_PARAM will call DDC_AFGetParamters.

Auto Exposure/Auto White balance Functions

4.5.22 DDC_AEWGetChannelHandle

Function	DDC_AEWGetChannelHandle
Function Prototype	PSP_Handle DDC_AEWGetChannelHandle(void);
Input Parameters	NULL
Output Parameters	NULL
Description	It will return the Channel handle for AEW.
Preconditions	Driver must be in opened state.
Design	Logic in steps 1) Return the channel specific object.

4.5.23 DDC_AEWOpenHandle

Function	DDC_AEWOpenHandle
Function Prototype	Void * DDC_AEWOpenHandle(void);
Input Parameters	NULL
Output Parameters	NULL
Description	It will return the Channel handle and update the state of AEW channel.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Initialize Channel semaphore 2) Update the Channel state to AEW_OPENED 3) Reset all the registers 4) Return the channel handle

4.5.24 DDC_AEWIsChannelHandleClosed

Function	DDC_AEWIsChannelHandleClosed
Function Prototype	DDC_H3AStatus DDC_AEWIsChannelHandleClosed (PSP_Handle 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_H3A_SOK else it will return DDC_H3A_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

4.5.25 DDC_AEWValidateParameters

Function	DDC_AEWValidateParameters
Function Prototype	Int DDC_AEWValidateParameters (PSP_AEWParams *)
Input Parameters	Pointer to PSP_AEWParams
Output Parameters	NULL
Description	It will Validate the af 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) Check the range of all parameters 2) Call LLC_AEWHardwareSetup function

4.5.26 DDC_AEWGetParameters

Function	DDC_AEWGetParameters

Function Prototype	Int DDC_AEWGetParameters (DDC_AEWChannelObject *, PSP_AEWParams *)
Input Parameters	Pointer to DDC_AEWChannelObject Pointer to PSP_AEWParams
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 AEW parameters from the channel specific structure.

4.5.27 DDC_AEWCloseHandle

Function	DDC_H3ACloseHandle
Function Prototype	DDC_H3AStatus DDC_H3ACloseHandle(PSP_Handle handle);
Input Parameters	Pointer to Channel 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_AEWGetDeviceHandle. 2) Decrement the no of channels created 3) Free the channel configuration structure 4) Delete the Channel semaphore 5) Reset the queues 6) Update the Channel state

4.5.28 DDC_AEWSubmit

Function	DDC_AEWSubmit
Function Prototype	DDC_H3AStatus DDC_H3ASubmit(PSP_Handle handle,PSP_VPSSSubmitCommand cmd, Ptr cmdArg);
Input Parameters	handle-Pointer to channel handle Cmd –Submit Command CmdArg – Pointer to 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) Lock sem 2) Call DDC_H3AValidateBuffer 3) Read the command 4) if command is PSP_VPSS_QUEUE then perform the enqueue operation using pal layer Function. 5) If the enqueue contains buffer enable the engine 6) if command is PSP_VPSS_DEQUEUE then perform the dequeue operation using pal layer Function. 7) If the buffer is queued or the statistics are available block ISR semaphore 8) Unlock sem

4.5.29 DDC_AEWisr

Function	DDC_AEWisr	
Function Prototype	Void ()	DDC_AEWisr
Input Parameters	NULL	
Output Parameters	NULL	
Description	It will be called when AF or AEW completes processing of one frame.	
Preconditions	Driver must be in opened state and channel handle should not be NULL.	

Design	Logic in steps 1) Check if the "ENQUEUE" is empty 2) If there is no buffer disable the engine 3) If there is buffer ins ENQUEUE Get next buffer from the enQueue. Set AEWBUFST Register to that buffer address. 4) Dequeue first buffer from ENQUEUE 5) Set TimeStamp 6) move the first buffer from enQueue to dequeue queue. 7) Decrement enqueue index 8) Increment the dequeue index 9) Unblock the ISR Semaphore 10) Check the write overflow bit and set the buffer status 11) Call LLC_AEWClearWriteBuffer

4.5.30 DDC_AEWIoctl

Function	DDC_H3AQueueFrameBuffer
Function Prototype	DDC_H3AStatus DDC_AEWIoctl(PSP_Handle handle, PSP_H3AIoctlCmd cmd, Ptr cmdArg, Ptr params);
Input Parameters	Pointer to Channel Object Control Command Command Argument Params
Output Parameters	NULL
Description	Function to execute the control command.
Preconditions	Driver must be in opened state and channel handle should not be NULL.

Design	Logic in steps : 1) Lock on Sem 2) Read the command. 3) Call the Functions that will execute the command 4) Unlock Sem E.g. H3A_SET_PARAM will call DDC_AF_ValidateParameters. H3A_GET_PARAM will call DDC_AFGetParamters.

4.6 LLC Layer Functions

Auto Focus Functions

4.6.1 LLC_AFHardwareSetup

Function	LLC_AFHardwareSetup
Function Prototype	CSL_Status LLC_AFHardwareSetup(PSP_AFParams * params, Ptr regs);
Input Parameters	Pointer to PSP_AFParams Structure Pointer to register overlaying structure
Output Parameters	This function will return the status indicating whether register are configured or not
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) From the channel config parameter structure fill every value into the hardware register by doing appropriate masking.

4.6.2 LLC_SetAFEngine

Function	LLC_SetAFEngine
Function Prototype	CSL_Status LLC_SetAFEngine(Uint32 value, Ptr regs);
Input Parameters	Value to enable / Disable Engine Pointer to register overlaying structure
Output Parameters	This function will return the status indicating whether register are configured or not
Description	This Function will enable the H3A 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) Write the AF enable bit into PCR register.

4.6.3 LLC_GetAFHWStatus

Function	LLC_GetAFHWStatus
Function Prototype	Int32 LLC_GetAFHWStatus (Ptr regs);

Input Parameters	Pointer to register overlaying structure
Output Parameters	Value of AF Enable Bit
Description	This Function will Check AF Engine Status.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get the AF enable bit into PCR register.

4.6.4 LLC_GetBusyAF

Function	LLC_GetBusyAF
Function Prototype	Int32 LLC_GetBusyAF (Ptr regs);
Input Parameters	Pointer to register overlaying structure
Output Parameters	Value of Busy bit for AF
Description	This function will get the Busy Bit for AF engine.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get AF Busy Bit from PCR Register

4.6.5 LLC_SetAFBUFST

Function	LLC_SetAFBUFST
Function Prototype	CSL_Status LLC_SetAFBUFST (Ptr regs, Uint32 address);
Input Parameters	Address – Address of the buffer Pointer to register overlaying structure
Output Parameters	This function will return the status indicating whether register were configured or not
Description	This function will set the address of buffer in the register. Hardware will fill the statistics in this buffer.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) set the buffer Address

4.6.6 LLC_AFGetWriteBufferStatus

Function	LLC_AFGetWriteBufferStatus
Function Prototype	Int32 LLC_AFGetWriteBufferStatus (Ptr regs);
Input Parameters	Pointer to register overlaying structure
Output Parameters	This Function will return the status of write Buffer Register .

Description	This function will return the status of write buffer overflow bit .
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Return the status of Write Buffer

4.6.7 LLC_AFClearWriteBuffer

Function	LLC_AFClearWriteBuffer
Function Prototype	CSL_Status LLC_AFClearWriteBuffer (Ptr const regs);
Input Parameters	Pointer to register overlaying structure
Output Parameters	This function will return the status whether register were configured or not.
Description	This function will clear write buffer overflow bit.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Clear the Write Buffer Overflow bit

4.6.8 LLC_AFInitRegister

Function	LLC_AFInitRegister

Function Prototype	CSL_Status LLC_AFInitRegister(Ptr regs);
Input Parameters	Pointer to register overlaying structure
Output Parameters	This function will return the status whether register were configured or not.
Description	This function will reset all the registers
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Reset all the registers

Auto Exposure/Auto White balance Functions

4.6.9 LLC_AEWHardwareSetup

Function	LLC_AEWHardwareSetup
Function Prototype	CSL_Status LLC_AEWHardwareSetup(PSP_AEWParams * params, Ptr regs);
Input Parameters	Pointer to PSP_AEWParams structure Pointer to register overlaying structure
Output Parameters	This function will return the status indicating whether register were configured or not
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) From the channel config parameter structure fill every value into the hardware register by doing appropriate masking.

4.6.10 LLC_SetAEWEngine

Function	LLC_SetAEWEngine
Function Prototype	CSL_Status LLC_SetAEWEngine(Uint32 value, Ptr regs);
Input Parameters	Value to enable / Disable Engine Pointer to register overlaying structure
Output Parameters	NULL
Description	This Function will enable the H3A 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)Write the AEW enable bit into the PCR register.

4.6.11 LLC_GetAEWHWStatus

Function	LLC_GetAFHWStatus
Function Prototype	Int32 LLC_GetAEWHWStatus (Ptr regs);

Input Parameters	Pointer to register overlaying structure
Output Parameters	Value of AEW Enable Bit
Description	This Function will Check AEW Engine Status
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get the AEW enable bit into PCR register.

4.6.12 LLC_GetBusyAEW

Function	LLC_GetBusyAEW
Function Prototype	Int32 LLC_GetBusyAEW (Ptr regs);
Input Parameters	Pointer to register overlaying structure
Output Parameters	Value of Busy bit for AEW
Description	This function will get the Busy Bit for AF engine.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Get AEWBusy Bit from PCR Register

4.6.13 LLC_SetAEWBUFST

Function	LLC_SetAEWBUFST
Function Prototype	CSL_Status LLC_SetAEWBUFST (Ptr regs, Uint32 address);
Input Parameters	Address – Address of the buffer Pointer to register overlaying structure
Output Parameters	This function will return the status indicating whether register were configured or not
Description	This function will set the address of buffer in the register. Hardware will fill the statistics in this buffer.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) set the buffer Address

4.6.14 LLC_AEWGetWriteBufferStatus

Function	LLC_AEWGetWriteBufferStatus
Function Prototype	Int32 LLC_AEWGetWriteBufferStatus (Ptr regs);
Input Parameters	Pointer to register overlaying structure
Output Parameters	This Function will return the status of write Buffer Register .
Description	This function will return the status of write buffer overflow bit .

Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Return the status of Write Buffer

4.6.15 LLC_AEWClearWriteBuffer

Function	LLC_AEWClearWriteBuffer
Function Prototype	CSL_Status LLC_AEWClearWriteBuffer (Ptr const regs);
Input Parameters	Pointer to register overlaying structure
Output Parameters	This function will return the status whether register were configured or not.
Description	This function will clear write buffer overflow bit.
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Clear the Write Buffer Overflow bit

4.6.16 LLC_AEWInitRegister

Function	LLC_AEWInitRegister
Function Prototype	CSL_Status LLC_AFInitRegister(Ptr regs);

Input Parameters	Pointer to register overlaying structure
Output Parameters	This function will return the status whether register were configured or not.
Description	This function will reset all the registers
Preconditions	Driver must be in opened state and channel handle should not be NULL.
Design	Logic in steps 1) Reset all the registers

5 Decision Analysis & Resolution

5.1 DAR Criteria

DAR Criteria

The list of DAR Criteria is as follows:

- Interface Simplicity
- Better control to application

Available Alternatives

Alternative 1

H3A driver will use same buffer management as CCDC.

Alternative 2

H3A 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 AF and AEW engine. Engine will be enabled again when ever new buffer is added to the input queue.

Decision

As alternative 2 gives better control over when to capture the statistics and when to not, and it also saves memory BW, we will go with alternative 2.

6 Revision History

Version #	Date	Author Name	Revision History
Draft 1.01	12 OCT 2006	EI4	Initial Draft Created
Draft 1.02	13 OCT 2006	EI4	Updated for technical review comments.
Draft 1.03	25 OCT 2006	EI4	Updated for QA review comments and following technical comments: 1) Added Channel object in device structure of H3A. 2) Added some constants 3) Remove function DDC_H3AEnqueue and DDC_H3ADequeue 4) Added LLC_GetAFHWStatus, LLC_SetAFBUFST ,Same is applicable for AEW. 5) Change in the prototypes of the function DDC_H3ASubmit.
Issue 1.00	25 OCT 2006	EI4	Issued to TII
Issue 1.01	15 NOV 2006	EI4	1) Prototype of all LLC functions are changed 2) LLC_AFGetWriteBufferStatus, LLC_AFClearWriteBuffer, LLC_AFInitRegister, LLC_AEWGetWriteBufferStatus, LLC_AEWClearWriteBuffer, LLC_AEWInitRegister functions are added 3) DDC_AF_MIN_HZCOUNT, DDC_AF_MIN_VTCOUNT, DDC_AEW_MIN_HZCOUNT, DDC_AEW_MIN_VTCOUNT, DDC_H3A_ISEVEN Macros are added 4) Enum PSP_H3ABufferStatus added

			5) Buffer status is added in PSP_H3ABuffer structure
Pre-silicon Release 0.3.0	20 NOV 2006	EI4	Release to TI
Post-silicon Release 0.3.0	30 NOV 2006	EI4	Release to TI
Post-silicon Release 0.3.0	7 DEC 2006	EI4	Description of section 3.1.1 is removed. Appropriate Reference of document is given
Post-silicon Release 0.3.0	21 DEC 2006	EI4	Isr Routine is modified for AF and AEW
Post-silicon Release 0.6.0	06 MAR 2006	EI4	In section 2.2 constraints are changed and new constants are added.
1.00.02	29 June 2006	Amit Chatterjee	Modified Release Version
1.00.03	18 July 2007	Maulik Desai	Modified Release Version