TII DM6437 VPSS Drivers
H3A API Specifications

Release Version: 1.10.00

January 14, 2008

Table of Contents

Revision History

Date	**Version**	**Changes**	**Author**
October 9, 2006	Draft 0.01	Created	EI4
October 11, 2006	Issue 1.00	Updated for technical review comments	EI4
		1) Change in the mdCreateChan Function. 2) Change in Submit Command 3) Change in the buffer structure	EI4
November 17, 2006	Issue 1.01	1) Modified PSP_H3ABuffer structure 2) Added description for behavior of Submit call	EI4
November 20, 2006	Pre-silicon Release 0.3.0	Release to TI	EI4
November 30,2006	Post-silicon Release 0.3.0	Release to TI	EI4
June 22, 2007	1.00.01	Updated the release version for GA Patch Release 1	Anuj Aggarwal
June 29 ,2005	1.00.02	Updated Release Version	Amit Chatterjee
July 18, 2007	1.00.03	Updated Release Version	Maulik Desai
November 29, 2007	1.00.04	Updated Release Version	Sivaraj R
January 14, 2008	1.00.05	PSP_H3A_IOCTL_SET_SEM_TIMEOUT IOCTL added	Sivaraj R

1. Overview

Purpose and Scope

This document provides APIs for the proposed driver for the H3a on DM6437 family SOCs. The APIs are based on the requirement document that has been agreed upon by the Catalog/EEE team, the PSP team and e-Infochips.

The intention of this document is to provide guidelines on how the driver should behave from application point of view. However, the actual design of the driver is not covered.

Names and Terminology

The module name of the H3a driver shall be H3A. Hence the name of the top level files which will directly interact with application shall be “dda_h3aIOM.c” and “dda_h3aIOM.h”. These above files will interact with the dda_h3a.c.Thereafter the dda_h3a.c will interact with the ddc_h3a.c. Finally, the files related to hardware block is referred to as the llc_h3a.c and llc_h3a.h

Architecture

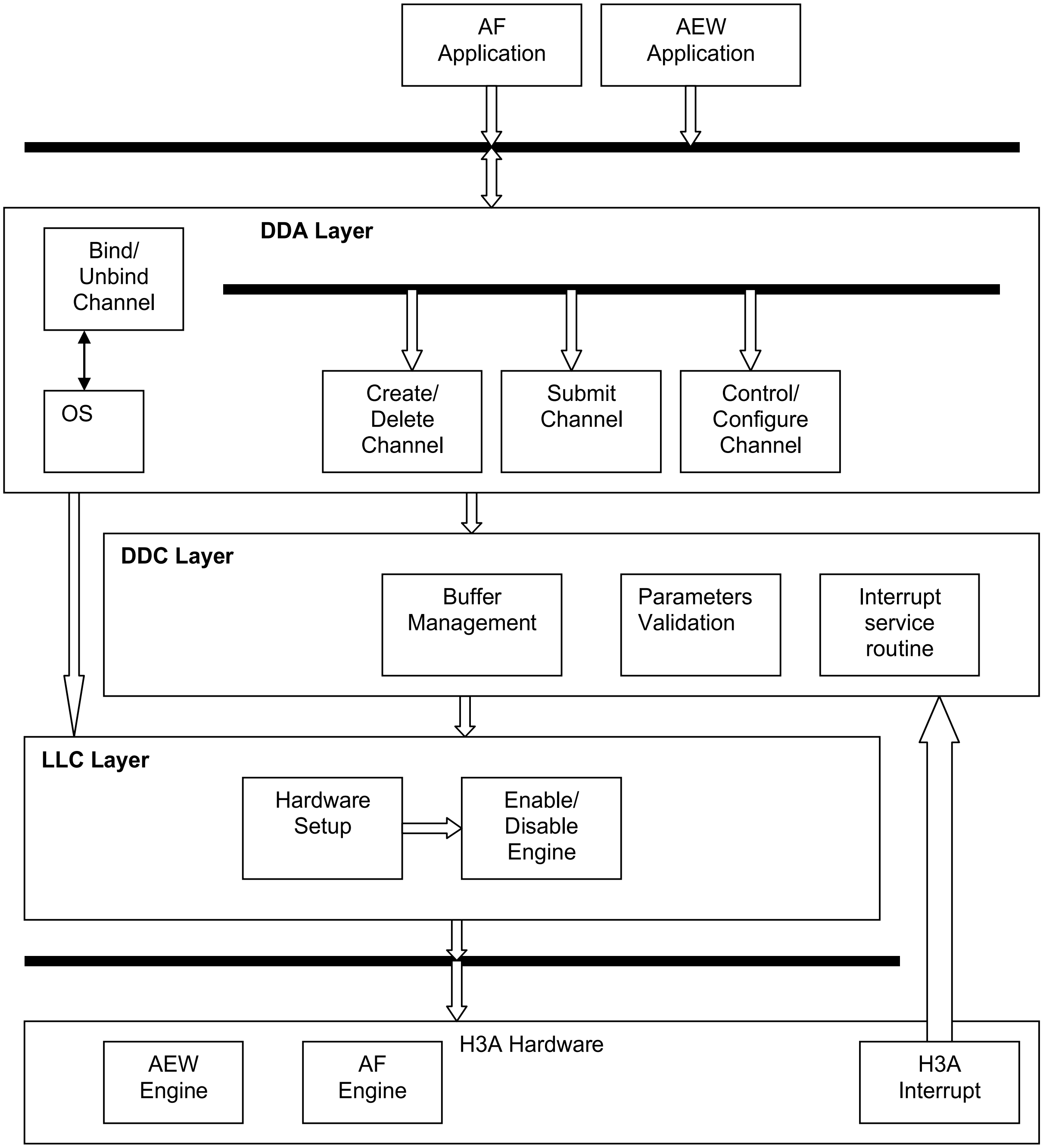

Figure 1. Top-Level Block Diagram of H3A

The H3A driver is sub-divided into following horizontal layers:

DDA Layer

This layer handles all OS level driver API implementations. This Layer will differentiate between AF and AEW Channel. This layer is OS centric and hardware agnostic. It does following functionalities:

- Registration/Unregistration
- Open/Close
- Various controls to configure HW

DDC Layer

This layer is mainly responsible for doing the statistical analysis on the captured frame & buffer management. It validates the parameters sent by the application. It also handles the ISR.

LLC Layer

This layer is responsible for the actual configuration of the Auto focus and Auto Exposure/White Balance hardware by writing to the H3A MMRs. It is pretty much OS agnostic and hardware centric. The layer enables the H3A module after the specific hardware is configured.

Critical Features and Implementation

H3A Driver will support Auto Focus and Auto Exposure/Auto White Balance Functionality.

2. Application Level APIs

The following GIO Class driver API shall be supported by the H3a driver.

- GIO_create()
- GIO_delete()
- GIO_control()
- GIO_submit()

GIO_CREATE

Synopsis

GIO_Handle GIO_create (String name, int mode, int Status, Ptr optargs, GIO_Attrs * attrs);

Arguments

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. Channel

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.

Status

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

Optargs

The optargs 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. Channel parameters will differentiate between AF and AEW. Application must pass the channel type.

The enumeration for channel type is as follows:

```
typedef enum _PSP_H3AChannelType
{
        PSP_H3A_AF = 0,
        PSP_H3A_AEW
} PSP_H3AChannelType;
```

Attrs

The attrs parameter is a pointer to a structure of type GIO_Attrs.

Description

Open a logical channel. Two open calls shall be supported by the H3A driver, one each for AF and AEW.

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;
```

Return Value

It returns the handle of type GIO_Handle on successful opening of a device. It returns NULL if it is unable to open the device.

GIO_DELETE

Synopsis
int GIO_delete(GIO_Handle gioChan);

Arguments
gioChan
Handle to device instance to be closed
Description
Close the logic channel associated with gioChan.

Return Value
IOM_COMPLETED on success, or negative value if an error occurred

GIO_CONTROL

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

GioChan
Handle to an instance of the device
cmd
Control functionality to perform
args
Data structure to pass control information

Description
An application calls GIO_control to configure or perform control functionality on the communication channel. Macros and defines specifying H3A control requests are located in the psp_h3a.h header file

Return Value
IOM_COMPLETED on success and negative value if error.

GIO_SUBMIT

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

Arguments
GioChan
Handle to an instance of the device
Cmd
Specified mini-driver command
bufp
Pointer to data structure for buffer data
pSize
Pointer to size of bufp structure

appCallback

Pointer to callback structure

Description

Submit a GIO packet to the mini-driver. The buffer structure and the commands for the buffer are mentioned in section 4 of this document.

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.

3. H3A module Controls

PSP_H3A_IOCTL_SET_PARAM

Name
PSP_H3A_IOCTL_SET_PARAM – set the AF/AEW channel hardware parameters associated with the channel

Synopsis
AF Channel
int GIO_control(GIO_Handle gioChan, int cmd, PSP_AFParams *argp);
AEW Channel
int GIO_control(GIO_Handle gioChan, int cmd, PSP_AEWParams*argp);
Arguments
gioChan
Handle to an instance of the device
Request
PSP_H3A_IOCTL_SET_PARAM
Argp
AF Channel
Pointer to PSP_AFParams structure
AEW Channel
Pointer to PSP_AEWParams structure
Description
The common structure used by H3A and its fields as defined in the psp_h3a.h header file as shown below.

```
/* H3A Ioctl commands */
enum _PSP_H3AIoctlCmd
{
   PSP_H3A_IOCTL_SET_PARAM = 128,
   /* Set H3A config params, cmdArg = parameter structure    */
   PSP_H3A_IOCTL_GET_PARAM,
   /* Get H3A config params, cmdArg = parameter structure    */
} PSP_H3AIoctlCmd;
```

AF Channel
This ioctl is used to set the following parameters/modules of the AF Channel

- AF Paxel parameters
- AF IIR Filter Parameters
- AF Horizontal Median Filter Parameters
- A-Law compression module
- AF Accumulator mode
- RGB Position

The PSP_AFParams structure and its fields as defined in the psp_af.h header file as shown below. The psp_af.h header file will be included in the psp_h3a.h header file.

The defines are as follows:

```
#define AF_NUMBER_OF_COEF 11
/* Enumeration definition for status of A law */
enum _PSP_AF_Alaw
```

```
{
   PSP_AF_ALAW_DISABLE     = 0, /* Disable A law */
   PSP_AF_ALAW_ENABLE      = 1 /* Enable A law */
};

/* Enumeration definition for status of Median Filter Law */
enum _PSP_AF_HMF_law
{
   PSP_AF_HMF_DISABLE = 0, /* Disable HMF Law*/
   PSP_AF_HMF_ENABLE   = 1 /*Enable HMF Law*/
};

/* Enumeration definition for status of Accumulator*/
enum _PSP_AF_mode
{
          PSP_AF_ACCUMULATOR_SUMMED = 0,        /* Summed mode of Accumulator */
          PSP_AF_ACCUMULATOR_PEAK    = 1/* Peak Mode of the accumulator*/
};

/* Enumeration definition for RGB Position */
enum _PSP_AF_rgbpos{
   PSP_AF_GR_GB_BAYER      =   0,       /*GR and GB as Bayer pattern*/
   PSP_AF_RG_GB_BAYER         =   1,    /* RG and GB as Bayer pattern*/
   PSP_AF_GR_BG_BAYER         =   2,    /* GR and BG as Bayer pattern*/
   PSP_AF_RG_BG_BAYER         =   3,    /* RG and BG as Bayer pattern*/
   PSP_AF_GG_RB_CUSTOM    =   4,     /* GG and GB as Custom pattern*/
   PSP_AF_RB_GG_CUSTOM    =   5      /* RB and GG as Custom pattern*/
};

/* Structure definition for Horizontal Median Filter */
typedef struct _PSP_AFHmf
{
   PSP_AF_HMF_law enable;     /*status of Horizontal Median Filter Law*/
   Unsigned int          threshold;  /* Threshold Value for Horizontal Median Filter */
} PSP_AFHmf;

/* Structure definition for IIR Filter
typedef struct  _PSP_AFIir
{
          Unsigned int    hzStarPos;                          /*IIR Start Register Value*/
          Int     coeffSet0 [AF_NUMBER_OF_COEF]; /* IIR Filter Coefficient for Set 0*/
          Int     coeffSet1 [AF_NUMBER_OF_COEF]; /* IIR Filter Coefficient for Set 1*/
 } PSP_AFIir;

/* Structure definition contains information regarding Paxels
typedef struct    _PSP_AFPaxel
{
 unsigned int    width;     /*Width of the Paxel*/
 unsigned int    height;    /* Height of the Paxel*/
 unsigned int    hzStart;  /*Horizontal Start Position*/
 unsigned int    vtStart;   /*Vertical Start Position*/
 unsigned int    hzCnt;     /*Horizontal Count */
 unsigned int    vtCnt;     /*vertical Count */
 unsigned int    lineIncr;  /*Line Increment */
} PSP_AFPaxel;
```

```
/* AF Parameter Structure */
typedef struct  _PSP_AFParams
{
PSP_AF_Alaw  alaw_enable;      /*ALWAW status*/
PSP_AFHmf    hmfConfig;        /*HMF configurations*/
PSP_AF_rgbpos      rgbPos;     /*RGB Positions*/
af_iir_t   iir_config;         /*IIR filter configurations*/
PSP_AFPaxel paxelConfig;       /*Paxel parameters*/
PSP_AF_mode mode;              /*Accumulator mode*/
} PSP_AFParams;
```

AEW Channel

The _PSP_AEWParams structure and its fields as defined in the psp_aew.h header file as shown below. The psp_aew.h header file will be included in the psp_h3a.h header file.

```
enum _PSP_AEW_Alaw
{
        PSP_AEW_ALAW_DISABLE     = 0, /* Disable A Law*/
        PSP_AEW_ALAW_ENABLE      = 1/* Enable A Law*/

};

/* Structure definition for Window Structure in AEW Engine
typedef struct _PSP_AEWWindow
{
        Unsigned int    width;           /* Width of the window  */
        Unsigned int    height;          /* Height of the window*/
        Unsigned int    hzStart;         /*Horizontal Start of the window*/
        Unsigned int    vtStart;         /*Vertical Start of the window*/
        Unsigned int    hzCnt;           /* Horizontal Count*/
        Unsigned int    vtCnt;           /* Vertical Count*/
        Unsigned int    hzLineIncr;      /* Horizontal Line Increment*/
        Unsigned int    vtLineIncr;      /* Vertical Line Increment*/
} PSP_AEWWindow;

/* Structure definition for AEW Black Window*/
typedef struct _PSP_AEWBlkWindow
{
        Unsigned int    height;    /* Height of the Black Window*/
        Unsigned int    vtStart;   /* Vertical Start of the black Window*/
} PSP_AEWBlkWindow;

/* Structure definition for AEW engine configuration*/
typedef struct _PSP_AEWParams
{
   PSP_AEW_Alaw            alaw_enable;    /* A-law status*/
   int                     satLimit;       /* Saturation Limit*/
   PSP_AEWWindow           winConfig;      /* Window for AEW Engine */
   PSP_AEWBlkWindow        blkWinConfig;   /* Black Window */
} PSP_AEWParams;
```

Return Value

IOM_COMPLETED on success and negative value if error.

PSP_H3A_IOCTL_GET_PARAM

Name
PSP_H3A_IOCTL_GET_PARAM– get the hardware parameters associated with AF/AEW channel.

Synopsis
AF Channel
int GIO_control(GIO_Handle gioChan, int cmd, PSP_AFParams);
AF Channel
int GIO_control(GIO_Handle gioChan, int cmd, PSP_AEWParams);
Arguments
gioChan
Handle to an instance of the device
request
PSP_H3A_IOCTL_GET_PARAM
Argp
AF Channel
Pointer to _PSP_AFParams structure
AEW Channel
Pointer to _PSP_AEWParams structure

Description
This ioctl is used to get the AF and AEW hardware settings associated with the current logic channel represented by gioChan.

Return Value
IOM_COMPLETED on success and negative value if error.

PSP_H3A_IOCTL_SET_SEM_TIMEOUT

Name
PSP_H3A_IOCTL_SET_SEM_TIMEOUT – set the timeout values used in semaphore operation in the driver. Values are in milliseconds.

Synopsis
int GIO_control(GIO_Handle gioChan, int cmd, Int32 *timeout);

Arguments
gioChan
Handle to an instance of the device
Request
PSP_H3A_IOCTL_SET_SEM_TIMEOUT
Argp
Pointer to Int32 – timeout in milliseconds; -1 should be provided for infinite timeout.

Description
This control command is used to set the timeout values used in semaphore operation in the driver associated with the current logic channel represented by gioChan.

Return Value
IOM_COMPLETED on success and negative value if error.

4. H3A Buffer Management

PSP_VPSS_QUEUE

Name

PSP_VPSS_QUEUE– Provides the buffer to capture statistics of the frame.

Synopsis

GIO_submit (GIO_Handle gioChan, Uns cmd, PSP_H3ABuffer * bufp, Uns *pSize, NULL);

GioChan
Handle to an instance of the device
cmd
PSP_VPSS_QUEUE

bufp
Pointer to _PSP_H3ABuffer structure
pSize
size of _PSP_H3ABuffer structure.
appCallback
NULL

Description
It provides the buffer and buffer size to the driver to store the data. The structure for the buffer is given below. Queuing of a new buffer will automatically enable the AF/AEW engine if required. Driver will also disable AF/AEW engine when all the buffers given by application are consumed. The 'ramIpAddr' field of _PSP_H3ABuffer structure must be 64 bit aligned address.

Enum definition for Buffer Status

```
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;

typedef struct _PSP_H3ABuffer{
PAL_OsListNodeHeader          nodeEntry;
Ptr                           ramIpAddr;
Uint                          timeStamp;
Uint                          size;
PSP_H3ABufferStatus           buffStatus;
} PSP_H3ABuffer;
```

Return Value
IOM_COMPLETED on success and negative value if error.

PSP_VPSS_DEQUEUE

Name

PSP_VPSS_DEQUEUE – Provides the statistics generated by AF / AEW Engine to the application.

Synopsis

GIO_submit (GIO_Handle gioChan, Uns cmd, PSP_H3ABuffer * bufp, Uns *pSize, NULL);

GioChan
Handle to an instance of the device
cmd
PSP_VPSS_DEQUEUE
bufp
Pointer to _PSP_H3ABuffer structure
pSize
size of _PSP_H3ABuffer structure.
appCallback
NULL

Description
It provides the captured data & number of byte read to the application. It will also set timestamp field in PSP_H3ABuffer. The timestamp will be expressed in terms of the milliseconds. Driver will set the “buffStatus” field in the structure to indicate whether buffer data is valid or corrupted.

Return Value
IOM_COMPLETED on success and negative value if error.

5. Usage Examples

This section provides some example code showing how to use the RSZ module.

Registration of H3A driver

To configure a mini-driver in the DSP/BIOS Configuration Tool, follow these steps:

1. Create a new device object by right-clicking on User-Defined Devices (in the Input/Output tree) and selecting Insert UDEV from the pop-up menu.

2. Rename the object as H3A.

3. Right-click on the UDEV object you created and choose Properties.

4. In the Properties dialog, specify the Initialize function name, name of the function table and function table type .See below

The Function table is as below:-

```
IOM_Fxns H3AMD_FXNS =
{
        &H3A_mdBindDev,
        &H3A_mdUnBindDev,
        &H3A_mdControlChan,
        &H3A_mdCreateChan,
        &H3A_mdDeleteChan,
};
```

The name of Initialize function name will be will be H3A_mdBindDev,
The name of function table will be H3AMD_FXNS.
The function table type will be IOM_Fxns.

Driver open and close

```
/* open a logical channel */.
GIO_Handle      afChan;
PSP_H3AChannelType type;
int gioStatus;
type= PSP_H3A_AF;
afHandle = GIO_create("/H3A",IOM_INOUT,&gioStatus ,&type,&gioAttrs);
if(afHandle == NULL) {
        printf("open h3a channel failed.\n")
        exit (-1);
}

/* close the logic channel */
GIO_delete (afChan);
```

Setup AF Engine parameters

```
PSP_AFParams params;

/* setup the parameter here */

/* setup the parameter here */
```

```
params->paxelConfig.width = 16;
params->paxelConfig.height = 12;
        .
        .
        .


/* configure the logic channel */
GIO_control (afChan, PSP_H3A_IOCTL_SET_PARAM, &params);
```

Enqueue the buffer

```
#define PSP_AF_PAXEL_SIZE           (48u)
PSP_H3ABuffer enbuffer;
unsigned int size;
/* Enqueue the buffer */
enbuffer.size =  params.paxelConfig.hzCntparams.paxelConfig.vtCnt * PSP_H3A_PAXEL_SIZE;
/* The address must be 64 bit aligned */
enbuffer.ramIpAddr = (void *)MEM_alloc(0,enbuffer.size,64);
size = sizeof(PSP_H3ABuffer);
GIO_submit (afChan,(Uint)PSP_VPSS_QUEUE,&enbuffer,&size,NULL);
```

Dequeue the buffer

```
unsigned int size;
PSP_H3ABuffer debuffer;
/* Dequeue the buffer */
size = sizeof(PSP_H3ABuffer);
GIO_submit (afChan,(Uint)PSP_VPSS_DEQUEUE,&debuffer,&size,NULL);
```

The name of Initialize function name will be will be H3A_mdBindDev,
The name of function table will be H3AMD_FXNS.
The function table type will be IOM_Fxns.

Driver open and close

```
/* open a logical channel */.
GIO_Handle      aewChan;
Int gioStatus
PSP_H3AChannelType type;
type = PSP_H3A_AEW;
aewHandle = GIO_create ("/H3A",IOM_INOUT,&gioStatus,&type,&gioAttrs);
if (aewHandle == NULL) {
        printf("open h3a channel failed.\n")
        exit (-1);
}

/* close the logic channel */
GIO_delete (aewChan);
```

Setup AEW Engine parameters

```
PSP_AEWParams params;

/* setup the parameter here */
/* setup the parameter here */
params->winConfig.width = 16;
params->winConfig.height = 12;
        .
        .
        .
/* configure the logic channel */
GIO_control (aewChan, PSP_H3A_IOCTL_SET_PARAM, &params);
```

Enqueue the buffer

```
#define PSP_AEW_WIN_SIZE          (18u)
PSP_H3ABuffer enbuffer;
unsigned int size;
/* Enqueue the buffer */
enbuffer.size = params.winConfig.hzCnt * params.winConfig.vtCnt * PSP_AEW_WIN_SIZE
/* The address must be 64 bit aligned */
enbuffer.ramIpAddr = (void *) MEM_alloc(0,enbuffer.size,64);
  size = sizeof(PSP_H3ABuffer);
GIO_submit (aewChan,(Uint)PSP_VPSS_QUEUE,&enbuffer,&size,NULL);
```

Dequeue the buffer

```
unsigned int size;
PSP_H3ABuffer debuffer;
/* Dequeue the buffer */
size = sizeof(PSP_H3ABuffer);
GIO_submit (aewChan,(Uint)PSP_VPSS_DEQUEUE,&debuffer,&size,NULL);
```