TEXAS INSTRUMENTS

DSP/BIOS VPBE Device Driver

User's Manual

June 2008 ***Platform Software Group***

Document Version 1.12

IMPORTANT NOTICE

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address:
Texas Instruments
Post Office Box 655303, Dallas, Texas 75265

Preface

Read This First

About This Manual

The API reference guide serves as a software programmer's handbook for working with the VPBE device driver modules. This reference guide provides necessary information regarding how to use these modules in user systems and applications.

Abbreviations

Table of Abbreviations

Abbreviation	Description
API	Application Programming Interface
DDC	Device Driver Core
IOM	Device Driver Adapter
ISR	Interrupt Service Routine
OS	Operating System
ROM	Read Only Memory
SOC	System On Chip

Revision History

Date	Author	Comments	Version
September 5,2006	Maulik Desai	Created the document	1.0
December 1, 2006	Maulik Desai	Modified for the release 0.3.0	1.1
January 2, 2007	Maulik Desai	Modified for the release 0.4.0	1.2
January 16, 2007	Maulik Desai	Bios version modified	1.3
February 3, 2007	Maulik Desai	CCS version modified	1.4
April 25, 2007	Maulik Desai	FVID layer modifications	1.5
May 5, 2007	Maulik Desai	IOCTL Description Added	1.6
May 15, 2007	Maulik Desai	FVID layer modifications	1.7
June 22, 2007	Anuj Aggarwal	Bios version modified	1.8
June 29,2007	Amit Chatterjee	Modified Release Version	1.9
June 30, 2007	Maulik Desai	Added Digital YCC16 support and Video Encoder(THS8200)	1.10
November 29, 2007	Sivaraj R	PSP merge package changes - directory structure changes, FVID_allocBuffer and FVID_freeBuffer functions are implemented as GIO control commands	1.11
January 24, 2008	Sivaraj R	Added TCI file driver initialization illustration and added dependent libraries for building video application	1.12
May21, 2008	Chandan Nath	Updated for adding compiler switches in build options	1.13

TABLE OF CONTENTS

List Of Figures

CHAPTER 1

INTRODUCTION

This document is an API reference guide on DSP/BIOS VPBE Device Driver for DM6437 SOC.

1.1 H/W S/W Support

This VPBE Device driver has been developed for the DSP/BIOS operating system using the TI supplied Chip Support Library. For more details on the version numbers refer to the release notes in the root of the installation.

1.2 Driver Components

The driver is constituted of following sub components:

VPBE IOM – Application facing, OS Specific Adaptation of VPBE Device Driver

VPBE DDC –OS Independent part of VPBE Driver Core

VPBE CSLR –The low-level VPBE h/w abstraction module

System components:

PALOS – DSP/BIOS Abstraction

CSLR– Non-Functional h/w abstraction.

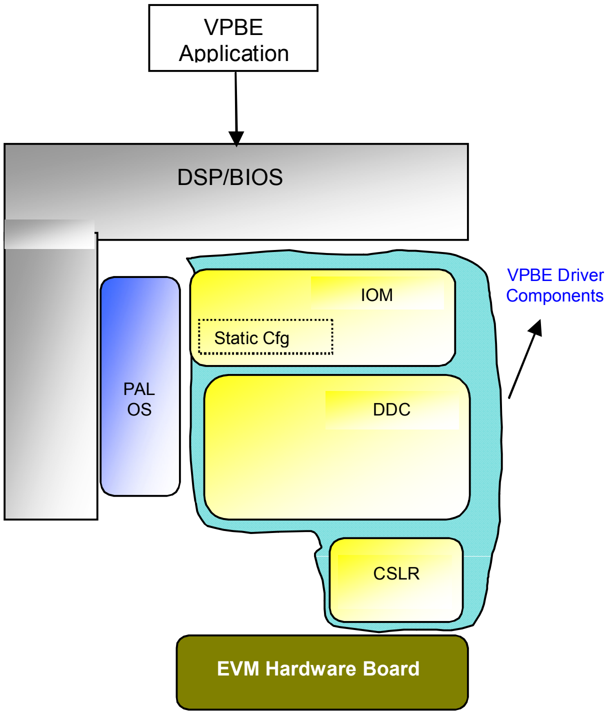


Figure 1 Device Driver Functional Decomposition

1.3 Default Driver Configuration

VPBE driver does not have any default configuration support. Before using the driver application should configure the driver with valid configurations. In case the driver recognizes invalid configuration the handle for the corresponding channel shall not be created and will returned NULL.

1.4 Driver Capabilities

The significant driver features are:

- Supports individual channels for Video/OSD/VENC and CURSOR.
- Driver is SYNCHRONOUS and operates in INTERRUPT mode only.
- Supports flipping of multiple frame buffers for seamless capture and display of video.
- Easy to maintain & re-target to new platforms.

1.5 System Requirements

Details about the tools and the BIOS version that the driver is compatible with can be found in the system Release Notes.

CHAPTER 2

INSTALLATION GUIDE

2.1 Component Folder

Upon installing the VPBE driver the following directory structure is found in the driver's directory.

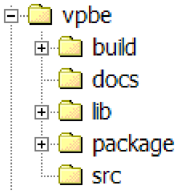


Figure 2 VPBE Driver Directory Structure

This top level vpbe folder contains vpbe driver psp header file and XDC package files (package.bld, package.xdc and package.xs)

- **build:** This folder contains vpbe driver library project file. The generated driver library shall be included in the application where VPBE driver have to be used.
- **docs:** This folder contains architecture document, datasheet, release notes and user guide.

  Architecture document contains the driver details which can be helpful for the developers as well as consumers to understand the driver design.

  Datasheet gives the idea about the memory consumption by the driver and description of the top level APIs.

  Release Note gives the details about system requirements, steps to Install/Uninstall the package.This document list the known issues of the driver.

  User Guide provides information about how to use the driver. It contains description of sample applications which guide the end user to make their applications using this driver.
- **lib:** This folder contains libraries generated in all the configuration modes(debug, idebug, irelease and release)
- **package:** This folder contains files generated by XDC tool.

- **src:** This folder contains vpbe driver source files. It also contains header files that are used by the driver.

2.2 Build

This section describes for each supported target environment, the applicable build options, supported configurations and how selected, the featured capabilities and how enabled, the allowed user customizations for the software to be installed and how the same can be realized.

The component might be delivered to user in different formats:

- Source-less ie., binary executables and object libraries only.
- Source –inclusive.,The entire source code is used to implement the driver is included in the delivered product.
- Source-selective ie., Only a part of the overall source is included. This delivery mechanism might be required either because ;certain parts of the driver require soruce level extensions and/or customization at the user's end or because,specific parts of the driver is exposed to user at the source level to insure user's software development.

When source is included as part of the product delivery, the CCS project file is provided as part of the package. When object format is distributed, the driver header files are part of the "*/drivers/vpbe/*" folder and the driver library is provided in */drivers/vpbe/lib* folder.

2.3 Build Options

This driver does not have any specific build option at the time of writing of this manual.

The build folder contains a CCS project file that builds the driver into a library for debug, idebug, release and irelease mode.

Following compiler switches are used to compile for different options.

- **_DEBUG**
  This is used as a flag to compiler whether to include the debug statements inserted in the code into the final image. This flag helps to build DEBUG image of the program. For RELEASE images this is not passed to the compiler.
- **CHIP_XXXX**
  The CSL layer is written in a common file for all the variants of a SOC. This flag differentiates the variant we are compiling for, for e.g. - CHIP_DM648, and the CSL definitions for that variant appropriately gets defined for register base addresses, num of ports of a peripheral etc.
- **VPBE_INSTRUMENTATION_ENABLED**
  This flag is passed to the compiler to include the instrumentation code parts into the final image/lib of the program. This helps build the iRelease/iDebug versions of the image/lib with a common code base.

CHAPTER 3

DSP/BIOS VPBE

This chapter describes the functions, data structures, enumerations and macros for the List module.

3.1 Functions

This section lists the functions available in the PSP module. FVID layer is implemented as a simple wrapper on top of the GIO class driver and provides an application-specific interface.

3.1.1. GIO_create/FVID_create

```
GIO_Handle GIO_create      (  String          name
                              Int             mode
OR                            Int *           status
                              Ptr             chanParams
                              GIO_Attrs*      attrs
FVID_Handle FVID_create    )
```

This function is called by the application to create the various VPBE channel. The channels that the VPBE supports are VIDEO (2 instance of Channel), OSD (2 instance of Channel), CURSOR (1 instance of Channel) and VENC (1 instance of Channel).

GIO_create () populates static settings in driver object; formally creates/registers driver entry points with DSP/BIOS. This call also registers interrupt for VENC.

Parameters:

name	[INOUT] Name of the device to open
mode	[INOUT] Mode in which device is to be opened (IGNORED AND NOT USED)
status	[OUT] Address to which driver returns status (IGNORED AND NOT USED)
chanParams	[IN] Valid Parameters required for creating channel handle .If the application passes invalid parameters then VPBE driver shall return NULL and the corresponding channel shall not be created
attrs	[IN] pointer to GIO_Attrs structure

Returns:

GIO_Handle – if the operation is successful
NULL – if the operation is failed

Example:

```
Prior to FVID_create call application should make sure that the VPBE
instance is created .The VPBE instance is created in the configuration
file (*.tcf) file in the User-defined devices.

The example below shows creation of Video-0 Channel for VPBE

FVID_Handle Vid0Handle;
GIO_Attrs gioAttrs = GIO_ATTRS;

PSP_VPBEChannelParams   beinitParams;
PSP_VPBEOsdConfigParams vid0Params;

static PSP_VPBEOsdConfigParams vid0Params =
{
    FVID_FRAME_MODE,                    /* ffmode       */
    FVID_BPP_BITS16,                    /* bitsPerPixel */
    FVID_YCbCr422_INTERLEAVED,          /* colorFormat  */
    (720 * (16/8u)),                    /* pitch        */
    {
        0,                              /* leftMargin   */
        0,                              /* topMargin    */
        720,                            /* width        */
        480,                            /* height       */
    },
    0,                                  /* segId        */
    PSP_VPBE_ZOOM_IDENTITY,             /* hScaling     */
    PSP_VPBE_ZOOM_IDENTITY,             /* vScaling     */
    PSP_VPBE_EXP_IDENTITY,              /* hExpansion   */
    PSP_VPBE_EXP_IDENTITY,              /* vExpansion   */
    NULL                                /* appCallback  */

};
beinitParams.id         = PSP_VPBE_VIDEO_0;
beinitParams.params     = (PSP_VPBEOsdConfigParams    *) &vid0Params;
Vid0Handle  = FVID_create ("/VPBE0", IOM_OUT, NULL,
                            & beinitParams, &gioattrs);
```

Name is passed as VPBE0 whose instance is declared in the configuration file

Note: Channel handle should not be shared across multiple tasks. Sharing of handle across multiple tasks might cause corruption in the GIO layer.

3.1.2. GIO_delete/FVID_delete

Int GIO_delete **(GIO_Handle gioChan,**
OR
Int FVID_delete)

This function deletes the driver channel

Parameters:

gioChan	[IN] GIO Handle for the channel

Returns:
IOM_EBADARGS – If the parameters passed are not correct or the channel has never been created

Example:

```
FVID_Handle     vpbeVid0Handle;

FVID_delete (vpbeVid0Handle);
```

3.1.3. GIO_control/FVID_control

Int FVID_control **(FVID_Handle *gioChan***
Int *cmd*,
Ptr *cmdArg*
OR
Int GIO_control)

This function handles the IOCTLs for the VPBE driver.

Parameters:

gioChan	[IN] Handle to FVID layer.
cmd	[IN] IOCTL Command
cmdArg	[INOUT] Argument for the IOCTL

Returns:
IOM_COMPLETED if successful or else suitable error code is given.
IOM_EBADARGS– *gioChan* is not valid or if the state is not appropriate or the argument passed is not appropriate
IOM_EBADMODE– if the cmdArg is not appropriate or if the FIFO is not enabled or if the mode is not supported

Example:

```
FVID_Handle                     vid0Handle;
PSP_VPBEOsdConfigParams vid0Params;
```

```
vid0Params.hExpansion =  PSP_VPBE_EXP_NINE_BY_EIGHT;
vid0Params.vExpansion =  PSP_VPBE_EXP_IDENTITY;
vid0Params.hScaling   =  PSP_VPBE_ZOOM_IDENTITY;
vid0Params.vScaling   =  PSP_VPBE_ZOOM_IDENTITY;

status = FVID_control (vid0Handle, PSP_VPBE_IOCTL_OSD_ZOOM, & vid0Params);
```

3.1.4. GIO_submit

Int GIO_submit	**(**	**GIO_Handle**	***gioChan***
		Uns	**cmd**
		Ptr	**bufp**
		Uns*	**pSize**
		GIO_AppCallBack*	**appCallback**
	)		

This function is called by the application to perform the read/write operation.

Parameters:

gioChan	[IN] Handle to GIO
bufp	[IN] Pointer to the Surfaceparams for Queuing and Dequeuing
psize	[IN] pointer to the Number of bytes (IGNORED NOT USED –make it 1)
cmd	Whether Queue/Dequeue cmd
appCallback	callback for the application

Returns:
IOM_COMPLETED if success else suitable error code
IOM_EBADARGS – if the arguments passed are not valid

Following are the FVID APIs which in ways calls GIO_submit API.

1. **FVID_queue** Queue the Frame Buffer in to Driver

   Syntax status = FVID_queue (gioChan, bufp);

   Parameters
   FVID_Handle giochan **/*** Handle to an instance of the driver */
   Ptr bufp /* pointer to allocated buffer by application */

   Description VPBE driver has its own queue maintain in the driver. An application will call FVID_queue to queue frame buffer in the driver queue. Application allocates memory for frame buffer.

   The giochan argument is the handle of the VPBE driver channel that was created with a call of FVID_create.

   The bufp argument is a pointer which points to surface params structure. Frame buffer is an element inside surface params structure whose memory is allocated by FVID_allocBuffer.

FVID_queue returns IOM_COMPLETED when it returns successfully. If an error occurs, a negative value will be returned.

Constraints This function can only be called after the device driver has been loaded and initialized. The handle supplied as an argument to the function should have been obtained with a previous call of FVID_create. The bufp supplied as an argument to the function should have been points to memory allocated by the application.

Example

```
Refer to example code to Queue the Frame Buffer using FVID_queue
function.

FVID_Handle   Vid0Handle;
FVID_Frame *VidallocFB = NULL;

   /* Allocate memory to Frame Buffer          */
FVID_allocBuffer (Vid0Handle, &VidallocFB);
   /* Queue the Frame Buffer after allocation */
FVID_queue (Vid0Handle, VidallocFB);
```

2. FVID_dequeue Dequeue the Frame Buffer from the Driver

Syntax status = FVID_dequeue (gioChan, bufp);

Parameters

FVID_Handle giochan /* Handle to an instance of the driver */
Ptr bufp /* pointer to allocated buffer by application */

Description VPBE driver has two queue maintained in the driver. Application will call FVID_queue which queues the Frame buffer in the driver’s first queue. Driver will fill the data in this frame buffer and keeps it in the second queue. Application can call FVID_dequeue to get the frame buffer filled with information.

The giochan argument is the handle of the VPBE driver channel that was created with a call of FVID_create.

The bufp argument is an out parameter that fills with a pointer to Frame buffer present in second queue of the driver.

FVID_dequeue returns IOM_COMPLETED when it returns successfully. If an error occurs, a negative value will be returned.

Constraints This function can only be called after the device driver has been loaded and initialized. The handle supplied as an argument to the function should have been obtained with a previous call of FVID_create. The bufp pointer supplied as an argument to the function should be NULL which will be filled by the driver.

Example

```
Refer to example code to dequeue the FrameBuffer using FVID_queue
function.

FVID_Handle   Vid0Handle;
FVID_Frame *allocBF = NULL;

status = FVID_dequeue (Vid0Handle, &allocBF);
```

3. FVID_exchange Exchange the Buffer between Application and Driver

Syntax status = FVID_exchange (gioChan, bufp);

Parameters

FVID_Handle giochan	/* Handle to an instance of the driver */
Ptr bufp	/* pointer to allocated buffer by application */

Description An Application will call FVID_exchange whose functionality is equivalent to serial calls of FVID_queue and FVID_dequeue, but the same thing can be done in a single API call. The application has to call FVID_queue once prior to FVID_exchange.

The giochan argument is the handle of the VPBE driver channel that was created with a call of FVID_create.

The bufp argument is an in/out parameter that points to frame buffer that is to be relinquished by the driver. After the call returns successfully, this function fills bufp with the pointer to the frame buffer that was previously queued in the device driver.

FVID_exchange returns IOM_COMPLETED when it returns successfully. If an error occurs, a negative value will be returned.

Constraints This function can only be called after the device driver has been loaded and initialized. The handle supplied as an argument to the function should have been obtained with a previous call of FVID_create. The FVID_queue must be called once prior to this function call.

Example

```
Refer to example code to exchange the FrameBuffer using FVID_exchange
function.

FVID_Handle   Vid0Handle;
FVID_Frame    *FBAddr = NULL;

/* Allocate memory to Frame Buffer              */
   FVID_allocBuffer (Vid0Handle, & FBAddr);
/* Exchange the Frame Buffers                   */
status = FVID_exchange (Vid0Handle, &FBAddr);
```

4. FVID_allocBuffer Allocates Memory to FrameBuffer

Syntax status = FVID_allocBuffer (gioChan, bufp);

Parameters

FVID_Handle giochan /* Handle to an instance of the driver */
Ptr bufp /* pointer to allocate buffer for application */

Description An Application will call FVID_allocBuffer to allocate the memory for Frame buffer.

The giochan argument is the handle of the VPBE driver channel that was created with a call of FVID_create.

The bufp argument is an out parameter that the function fills with a pointer to the memory allocated by driver.

FVID_allocBuffer returns IOM_COMPLETED when it returns successfully. If an error occurs, a negative value will be returned.

Constraints This function can only be called after the device driver has been loaded and initialized. The handle supplied as an argument to the function should have been obtained with a previous call of FVID_create. The bufp pointer supplied as an argument to the function should be NULL which will be filled by the driver.

Example

```
Refer to example code to allocate memory to the FrameBuffer using
FVID_allocBuffer function.

FVID_Handle           vid0Handle;
FVID_Frame *FBAddr = NULL;

status = FVID_allocBuffer (vid0Handle, & FBAddr);
```

5. FVID_freeBuffer — Frees the allocated Memory of Frame Buffer

Syntax status = FVID_freeBuffer (gioChan, bufp);

Parameters

FVID_Handle	giochan	/* Handle to an instance of the driver */
Ptr	bufp	/* Frame buffer to free the allocated memory */

Description An Application will call FVID_freeBuffer to free the memory allocated by the application.

The giochan argument is the handle of the VPBE driver channel that was created with a call of FVID_create.

FVID_freeBuffer returns IOM_COMPLETED when it returns successfully. If an error occurs, a negative value will be returned.

Constraints This function can only be called after the device driver has been loaded and initialized. The handle supplied as an argument to the function should have been obtained with a previous call of FVID_create.

Example

```
Refer to example code to free the Frame Buffer using FVID_freeBuffer
function.

FVID_Handle vid0Handle;
FVID_Frame *FBAddr = NULL;

status = FVID_allocBuffer (vid0Handle, & FBAddr);

status = FVID_freeBuffer (vid0Handle, FBAddr);
```

3.2 Control Commands

3.2.1. PSP_VPBE_IOCTL_OSD_VALIDATE_BUFFER

PSP_VPBE_IOCTL_OSD_VALIDATE_BUFFER – Validate the OSD's any channel framebuffer pitch. It will return an error if the framebuffer pitch is not proper as expected by the driver.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel
  - **cmd** – PSP_VPFE_IOCTL_CCDC_VALIDATE_BUFFER
  - **cmdArg** – Pointer to PSP_VPBEValidateParams structure
- **RETURN VALUE**
  - IOM_COMPLETED – Success if the pitch is validated
  - IOM_EBADARGS – if args passed is NULL or if the pitch is not validated
- **DESCRIPTION**

  This Ioctl is used to validate the frame buffer pitch. It will return an error if the framebuffer pitch is not proper as expected by the driver. In case of error, the driver will fill the expected value of framebuffer pitch in second element of PSP_VPBEValidateParams structure. An application can read the second element of PSP_VPBEValidateParams structure to know the expected value of framebuffer pitch.

  The structure is described as below:

```
typedef struct _PSP_VPBEValidateParams
{
   FVID_Frame      InParams;        /**< Pass the Surface params as input to validate   */
                                    /**< Presently Pitch validation is only done          */
   FVID_Frame      OutParams;       /**< If the Pitch given as Input doesn't validate then
                                          appropriate pitch is given as this Params.
                                     Pass NULL params to this element                 */
} PSP_VPBEValidateParams;
```

- **LIMITATIONS/CONSTRAINTS**

  None

3.2.2. PSP_VPBE_IOCTL_OSD_BOUNDARY

PSP_VPBE_IOCTL_OSD_BOUNDARY –This Ioctl is used to change the boundaries of any active OSD windows. It will return an error if the boundaries of OSD windows other than video-0 window is more than video-0 windows boundary.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);

- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel
  - **cmd** – PSP_VPBE_IOCTL_OSD_BOUNDARY
  - **cmdArg** – Pointer to PSP_VPBEOsdConfigParams structure

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the boundary parameters are valid
  - IOM_EBADARGS – if args passed is NULL or boundary values of other active windows are more than boundary values of video-0 window.

- **DESCRIPTION**
  This Ioctl is used to change height, width, top margin and left margin of any active window. It will return an error if the boundary values of other active windows are more than boundary values of video-0 window. The structure is described in section 3.3 table 2.

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.3. PSP_VPBE_IOCTL_OSD_ZOOM

PSP_VPBE_IOCTL_OSD_ZOOM –This Ioctl is used to change the zoom of any active OSD windows. User can zoom the window by 1x, 2x and 4x using these ioctl. User can also expand horizontal by 9/8 and vertical expansion by 6/5 dynamically using these ioctl.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);

- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel
  - **cmd** – PSP_VPBE_IOCTL_OSD_ZOOM
  - **cmdArg** – Pointer to PSP_VPBEOsdConfigParams structure

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the boundary parameters are valid
  - IOM_EBADARGS – if args passed is NULL or arg passed are different than enum values.

- **DESCRIPTION**
  This Ioctl is used to change horizontal scaling, vertical scaling, horizontal expansion and vertical expansion of any active window. It will return an error if the values passed other than defined in the enum. The structure is described in section 3.3 table 2. The enum values used are described below:

```
typedef enum _PSP_VPBE_Scalingfactor
{
  /**< No Scaling                                */
```

```
    PSP_VPBE_ZOOM_IDENTITY = 1,
    /**< Scaling by 2x                              */
    PSP_VPBE_ZOOM_2X,
    /**< Scaling by 4x                              */
    PSP_VPBE_ZOOM_4X

} PSP_VPBEScalingFactor;
typedef enum _PSP_VPBE_Expansionfactor
{
    /**< No expansion                                                              */
   PSP_VPBE_EXP_IDENTITY   = 1,
    /**< Square Pixel Expansion by 9/8(Horizontal Direction only)          */
    PSP_VPBE_EXP_NINE_BY_EIGHT,
    /**< Square Pixel Expansion by 6/5(Vertical Direction only)            */
    PSP_VPBE_EXP_SIX_BY_FIVE

} PSP_VPBEExpansionFactor;
```

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.4. PSP_VPBE_IOCTL_OSD_BLENDING

PSP_VPBE_IOCTL_OSD_BLENDING –This Ioctl is used for blending the graphics windows (OSD-0 and OSD-1) with the Video windows (video-0 and video-1) with some blending factor. The blending factor can be changed dynamically using these ioctl.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);

- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel
  - **cmd** – PSP_VPBE_IOCTL_OSD_BLENDING
  - **cmdArg** – Pointer to PSP_VPBEOsdConfigParams structure

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the blending factor is valid
  - IOM_EBADARGS – Error if args passed is NULL or args passed with value different than enum values.

- **DESCRIPTION**
  This Ioctl is used to change the blending factor of OSD -0 and OSD-1 windows with Video -0 and video -1 windows dynamically. The structure is described in section 3.3 table 2. The enum values for blending factor are described below:

```
typedef enum _PSP_VPBE_BlendingFactor
{
    PSP_VPBE_BLEND0 = 0,
    PSP_VPBE_BLEND1,
    PSP_VPBE_BLEND2,
    PSP_VPBE_BLEND3,
    PSP_VPBE_BLEND4,
    PSP_VPBE_BLEND5,
    PSP_VPBE_BLEND6,
    PSP_VPBE_BLEND7
} PSP_VPBEBlendingFactor;
```

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.5. PSP_VPBE_IOCTL_OSD_TRANSPARENCY

PSP_VPBE_IOCTL_OSD_TRANSPARENCY– This Ioctl is used to make any portion of video windows transparent from graphics windows (OSD-0 and OSD-1).The transparency value should be changed dynamically using these Ioctl.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);

- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel
  - **cmd** – PSP_VPBE_IOCTL_OSD_TRANSPARENCY
  - **cmdArg** – Pointer to PSP_VPBEOsdConfigParams structure

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the blending factor is valid
  - IOM_EBADARGS – Error if args passed is NULL or args passed with value different than enum values.

- **DESCRIPTION**
  This ioctl is used to make any portion of video windows transparent from graphics windows. The transparency value is specified from the graphics windows values. The transparency value varies from 0 to 65535 values. The structure is described in section 3.3 table 2.

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.6. PSP_VPBE_IOCTL_OSD_PROGRAM_MAIN_CLUT

PSP_VPBE_IOCTL_OSD_PROGRAM_MAIN_CLUT – This Ioctl is used to enable the Main Color look up table. User can set the background color using these ioctl.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);

- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel
  - **cmd** – PSP_VPBE_IOCTL_OSD_PROGRAM_MAIN_CLUT
  - **cmdArg** – Pointer to PSP_VPBEMainClutParams structure

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the blending factor is valid
  - IOM_EBADARGS – Error if args passed is NULL or args passed with value different than enum values.

- **DESCRIPTION**

This ioctl is used to enable main color look up table. User can set the background color dynamically using these ioctl. The structure used as cmdargs is described as below:

```
typedef struct _PSP_VPBEMainClutParams
{
  PSP_VPBEColorSpaceParams
                          MainClut_val[PSP_VPBE_MAX_MAIN_CLUT_ENTRIES];
   /**< Main clut values                         */
   PSP_VPBERAMWRMode          mode;
   /**< RAM WR Mode                              */

} PSP_VPBEMainClutParams;
```

- **LIMITATIONS/CONSTRAINTS**

None

3.2.7. PSP_VPBE_IOCTL_OSD_PROGRAM_BMP_CLUT

PSP_VPBE_IOCTL_OSD_PROGRAM_BMP_CLUT – This Ioctl is used to enable the background color of any graphics windows. User can set the background color using these ioctl.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);

- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel
  - **cmd** – PSP_VPBE_IOCTL_OSD_PROGRAM_BMP_CLUT
  - **cmdArg** – Pointer to PSP_VPBEBMPClutParams structure

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the blending factor is valid
  - IOM_EBADARGS – Error if args passed is NULL or args passed with value different than enum values.

- **DESCRIPTION**

This ioctl is used to enable background color look of any graphics windows. User can configure the background color dynamically using these ioctl. The structure used as cmdarg is described as below:

```
typedef struct _PSP_VPBE_BMPClutParams
{
   Uint8              BMPClut_val[PSP_VPBE_MAX_BMP_CLUT_ENTRIES];
   /**< BMP clut values                              */
   PSP_VPBERAMWRMode      mode;
   /**< RAM WR Mode                                  */
} PSP_VPBEBMPClutParams;
```

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.8. PSP_VPBE_IOCTL_START_VIDEO- 0

PSP_VPBE_IOCTL_START_VIDEO-0 – This Ioctl is used to start the video -0 window.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel
  - **cmd** – PSP_VPBE_IOCTL_START_VIDEO-0
  - **cmdArg** – NULL
- **RETURN VALUE**
  - IOM_COMPLETED – Success if the argument is valid
  - IOM_EBADARGS – Error if args passed is NULL
- **DESCRIPTION**
  This Ioctl is used to start Video -0 window. It is required to queue one buffer initially before using these Ioctl.
- **LIMITATIONS/CONSTRAINTS**
  None

3.2.9. PSP_VPBE_IOCTL_START_VIDEO- 1

PSP_VPBE_IOCTL_START_VIDEO-1 – This Ioctl is used to start the video -1 window.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of OSD Channel

  - **cmd** – PSP_VPBE_IOCTL_START_VIDEO-1
  - **cmdArg** – NULL

- **RETURN VALUE**
  Refer to section 3.2.8 for return value

- **DESCRIPTION**
  Refer to section 3.2.8 for description

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.10 PSP_VPBE_IOCTL_START_OSD-0

PSP_VPBE_IOCTL_START_OSD-0 – This Ioctl is used to start the osd - 0 window.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);

- **ARGUMENTS**
  - **gioChan** – Handle of OSD-0 Channel
  - **cmd** – PSP_VPBE_IOCTL_START_OSD-0
  - **cmdArg** – NULL

- **RETURN VALUE**
  Refer to section 3.2.8 for return value.

- **DESCRIPTION**
  Refer to section 3.2.8 for description

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.11 PSP_VPBE_IOCTL_START_OSD-1

PSP_VPBE_IOCTL_START_OSD-1 – This Ioctl is used to start the osd - 1 window.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);

- **ARGUMENTS**
  - **gioChan** – Handle of OSD-1 Channel
  - **cmd** – PSP_VPBE_IOCTL_START_OSD-1
  - **cmdArg** – NULL

- **RETURN VALUE**
  Refer to section 3.2.8 for return value

- **DESCRIPTION**
  Refer to section 3.2.8 for description
- **LIMITATIONS/CONSTRAINTS**
  None

3.2.12 PSP_VPBE_IOCTL_START_CURSOR

PSP_VPBE_IOCTL_START_CURSOR – This Ioctl is used to start the cursor window.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of Cursor window
  - **cmd** – PSP_VPBE_IOCTL_START_CURSOR.
  - **cmdArg** – NULL
- **RETURN VALUE**
  - IOM_COMPLETED – Success if the argument is valid
  - IOM_EBADARGS – Error if args passed is NULL
- **DESCRIPTION**
  This Ioctl is used to start Cursor window.
- **LIMITATIONS/CONSTRAINTS**
  None

3.2.13 PSP_VPBE_IOCTL_START_VENC

PSP_VPBE_IOCTL_START_VENC – This Ioctl is used to start the venc module.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of Venc module
  - **cmd** – PSP_VPBE_IOCTL_START_VENC
  - **cmdArg** – NULL
- **RETURN VALUE**
  - IOM_COMPLETED – Success if the argument is valid
  - IOM_EBADARGS – Error if args passed is NULL
- **DESCRIPTION**
  This Ioctl is used to start venc module.

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.14 PSP_VPBE_IOCTL_STOP_VIDEO- 0

PSP_VPFE_IOCTL_STOP_VIDEO-0 – This Ioctl is used to stop the video -0 window.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of video -0 channel
  - **cmd** – PSP_VPBE_IOCTL_STOP_VIDEO-0
  - **cmdArg** – NULL
- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL
- **DESCRIPTION**
  This IOCTL stops the Video -0 window.
- **LIMITATION/CONSTRAINTS**
  There can be many buffers which are in the pending state when you stop the engine. These pending buffers do not have any valid information. By dequeue call, application can get those pending buffers but its application's responsibility to make sure that these buffers are not used for any further processing as these buffers are not having valid video information.

3.2.15 PSP_VPBE_IOCTL_STOP_VIDEO- 1

PSP_VPFE_IOCTL_STOP_VIDEO-1 – This Ioctl is used to stop the video -1 window

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of video -1 channel
  - **cmd** – PSP_VPBE_IOCTL_STOP_VIDEO-1
  - **cmdArg** – NULL
- **RETURN VALUE**
  Refer to section 3.2.14 for return values.
- **DESCRIPTION**
  Refer to section 3.2.14 for description

- **LIMITATIONS/CONSTRAINTS**
  Refer to section 3.2.14 for limitation

3.2.16 PSP_VPBE_IOCTL_STOP_OSD- 0

PSP_VPFE_IOCTL_STOP_OSD -0 – This Ioctl is used to stop the osd -0 window

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of video -0 channel
  - **cmd** – PSP_VPBE_IOCTL_STOP_OSD -0
  - **cmdArg** – NULL
- **RETURN VALUE**
  Refer to section 3.2.14 for return values
- **DESCRIPTION**
  Refer to section 3.2.14 for description
- **LIMITATIONS/CONSTRAINTS**
  Refer to section 3.2.14 for limitation

3.2.17 PSP_VPBE_IOCTL_STOP_OSD- 1

PSP_VPFE_IOCTL_STOP_OSD -1 – This Ioctl is used to stop the osd -1 window

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of osd -1 channel
  - **cmd** – PSP_VPBE_IOCTL_STOP_OSD -1
  - **cmdArg** – NULL
- **RETURN VALUE**
  Refer to section 3.2.14 for return values
- **DESCRIPTION**
  Refer to section 3.2.14 for description
- **LIMITATIONS/CONSTRAINTS**
  Refer to section 3.2.14 for limitation

3.2.18 PSP_VPBE_IOCTL_STOP_CURSOR

PSP_VPBE_IOCTL_STOP_CURSOR – This Ioctl is used to stop the Cursor window

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of Cursor channel
  - **cmd** – PSP_VPBE_IOCTL_STOP_CURSOR
  - **cmdArg** – NULL
- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL
- **DESCRIPTION**

  This Ioctl is used to stop the Cursor window.

  Note: Queue operations are not performed for these channels as they are performed for other channels (Video-0, Video-1, etc).
- **LIMITATIONS/CONSTRAINTS**

  None

3.2.19 PSP_VPBE_IOCTL_STOP_VENC

PSP_VPBE_IOCTL_STOP_VENC – This Ioctl is used to stop the Venc window

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of Venc channel
  - **cmd** – PSP_VPBE_IOCTL_STOP_VENC
  - **cmdArg** – NULL
- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL
- **DESCRIPTION**

  This Ioctl is used to stop the Venc window.

  Note: Queue operations are not performed for these channels as they are performed for other channels (Video-0, Video-1, etc).

- **LIMITATIONS/CONSTRAINTS**
  None

3.3 Data Structures Configuration defines

The file **psp_vpbe.h** has the **PSP_VPBEChannelParams** data structure that is passed at the time of opening the VPBE individual Channels from the Application .The params are explained below:

1) Table Configuration Data Structure

Parameter	Description
Id	Channel Id **Note:** Each VPBE modules are treated as individual channels
Params	Pass the Structure of the Channel i.e;PSP_VPBEOsdConfigParams or PSP_VPBEVencConfigParams, etc

The file **psp_vpbe.h** has the **PSP_VPBEOsdConfigParams** data structure passed as a parameter of VPBEChannelParams structure mentioned above if configuring any OSD windows. The params are explained below:

2) Table Configuration Data Structure

Parameter	Description
ffMode	FILED/FRAME Mode **Note:** Here field mode refers to Progressive and Frame mode refers to Interlaced mode.
bitsPerPixel	Bits Per Pixel
colorFormat	ColorFormat i.e.; YCbCr or RGB888
pitch	Pitch or offset of the Frame.
leftMargin	Left Margin of the Osd window frame
topMargin	Top margin of the Osd window frame
width	Width of the Osd window frame
height	Height of the Osd window frame
segId	segId to allocate memory
Blending	Blending Factor **NOTE:** Applicable to OSD -0 and OSD-1 windows only
Transparency	Transparency enable /disable **NOTE:** Applicable to OSD -0 and OSD-1 windows only
TransparencyColor	Transparency Color to be set **NOTE:** Applicable to OSD -0 and OSD-1 windows only
hExpansion	Square Pixel Expansion in Horizontal direction

vExpansion	Square Pixel Expansion in Vertical direction
hScaling	Scaling in Horizontal Direction
vScaling	Scaling in Vertical Direction
appCallback	Application Callback Function
CLUTSource	ClutSource for OSD-0 and OSD-1 plane

Note: Select FIELD mode when VENC module is configured in PROGRESSIVE mode and select FRAME mode when VENC module is configured in INTERLACED mode.

The file **psp_vpbe.h** has the **PSP_VPBECursorConfigParams** data structure that is parameter of VPBEChannelParams structure mentioned above if configuring Cursor Window. The params are explained below:

3) Table Configuration Data Structure

Parameter	**Description**
leftMargin	Left Margin of Cursor window
topMargin	Top Margin of Cursor window
Width	Width of Cursor window
Height	Height of Cursor window
CursorCLUTS	Clut source of Cursor

The file **psp_vpbe.h** has the **PSP_VPBEVencConfigParams** data structure that is parameter of VPBEChannelParams structure mentioned above if configuring VENC. The params are explained below:

4) Table Configuration Data Structure

Parameter	**Description**
displayStandard	Display Standard used for Analog Display i.e; NTSC or PAL
extVidEncFxn	External Video Encoder Function. **Note:** Pass the external Video Encoder function only when display standard is configured for digital output

The file **psp_vpss.h** has the **FVID_Frame** data structure that is passed to **VPBE_mdSubmitChan** function of the driver. The params are explained below:

5) Table Configuration Data Structure

Parameter	**Description**

pitch	Pitch or Offset of the Frame
lines	Number of lines of the Frame.
frame.frameBufferPtr	FrameBuffer allocated by Application. **Note: Application allocates memory for the Frame buffer and passes it to driver.**
Bpp	Bits Per Pixel

Note: The maximum number of buffers that can be queued in the driver is upto 20.

3.4 Enumerations

This section lists the enumerations available in the PSP module.

enum IOCTL Commands

Defines	**Description**
PSP_VPBE_IOCTL_OSD_BOUNDARY	To change boundary parameters of OSD Windows
PSP_VPBE_IOCTL_OSD_BLENDING	To Change Blending Factor at runtime **NOTE:**Applicable to OSD-0 and OSD-1 window only
PSP_VPBE_IOCTL_OSD_TRANSPARENCY	To change the Transparency Color at runtime **NOTE:**Applicable to OSD-0 and OSD-1 window only
PSP_VPBE_IOCTL_OSD_ZOOM	To Zoom any OSD windows runtime
PSP_VPBE_IOCTL_OSD_PROGRAM_MAIN_CLUT	To program the MAIN CLUT
PSP_VPBE_IOCTL_OSD_PROGRAM_BMP_CLUT	To program the BMP CLUT
PSP_VPBE_IOCTL_OSD_VALIDATE_BUFFER	To validate Buffer and also get the Correct buffer size
PSP_VPBE_IOCTL_START_VIDEO_0	To Start theVideo -0 plane of OSD engine
PSP_VPBE_IOCTL_START_VIDEO_1	To Start theVideo -1 plane of OSD engine
PSP_VPBE_IOCTL_START_OSD_0	To Start the OSD-0 plane of OSD engine
PSP_VPBE_IOCTL_START_OSD_1	To Start the OSD-1 plane of OSD engine
PSP_VPBE_IOCTL_START_CURSOR	To Start the Cursor plane of OSD engine
PSP_VPBE_IOCTL_START_VENC	To Start the VENC Engine
PSP_VPBE_IOCTL_STOP_VIDEO_0	To Stop the Video -0 plane of OSD engine
PSP_VPBE_IOCTL_STOP_VIDEO_1	To Stop the Video -1 plane of OSD engine
PSP_VPBE_IOCTL_STOP_OSD_0	To Stop the Osd -0 plane of OSD engine
PSP_VPBE_IOCTL_STOP_OSD_1	To Stop the Osd -1 plane of OSD engine

PSP_VPBE_IOCTL_STOP_CURSOR	To Stop the Cursor plane of OSD engine
PSP_VPBE_IOCTL_STOP_VENC	To Stop the VENC Engine

enum Channel Id

Defines	**Description**
PSP_VPBE_INVALID_ID	Invalid Id
PSP_VPBE_VIDEO_0	Video-0 window Id
PSP_VPBE_VIDEO_1	Video-1 window Id
PSP_VPBE_OSD_0	OSD-0 window Id
PSP_VPBE_OSD_1	OSD-1 window Id
PSP_VPBE_CURSOR	Cursor Id
PSP_VPBE_VENC	VENC Id
PSP_VPBE_MAX_INTERFACES_SUPPORTED	Maximum number of Id supported

enum Scaling Factor

Defines	**Description**
PSP_VPBE_ZOOM_IDENTITY	Scale OSD Windows by 1X
PSP_VPBE_ZOOM_2X	Scale OSD Windows by 2X
PSP_VPBE_ZOOM_4X	Scale OSD Windows by 4X

enum Expansion Factor

Defines	**Description**
PSP_VPBE_EXP_IDENTITY	Square Pixel Expansion on any OSD Windows by 1
PSP_VPBE_EXP_NINE_BY_EIGHT	Square Pixel Expansion on any OSD Windows in Horizontal Direction by 9/8
PSP_VPBE_EXP_SIX_BY_FIVE	Square Pixel Expansion on any OSD Windows in Vertical Direction by 6/5

enum Color Format

Defines	**Description**
PSP_VPBE_YCbCr422/FVID_YCbCr422_INTERLEAVED	YCbCr422 format **NOTE:** Applicable to Video Windows only
PSP_VPBE_YCrCb422/FVID_YCrCb422_INTERLEAVED	YCrCb422 format **NOTE:** Applicable to Video Windows only
PSP_VPBE_RGB_888 / FVID_RGB_888_INTERLEAVED	RGB888 format

	NOTE: Applicable to Video Windows only
PSP_VPBE_RGB565 / FVID_RGB565_INTERLEAVED	RGB565 format **NOTE:** Applicable to OSD-0 and OSD-1 windows only
PSP_VPBE_DVD_MODE/FVID_DVD_MODE	DVD mode format **NOTE:** Applicable to OSD-0 and OSD-1 windows only
PSP_VPBE_CLUT_INDEXED/FVID_CLUT_INDEXED	CLUT Indexed **NOTE:**Applicable to OSD-0 and OSD-1 windows only
PSP_VPBE_ATTRIBUTE/FVID_ATTRIBUTE	Attribute Format **NOTE:**Applicable to OSD-1 windows only
PSP_VPBE_INVALID/FVID_COLORFORMAT_INVALID	Invalid Color Format

enum Display Timings

Defines	**Description**
PSP_VPBE_AUTO_MODE	Auto Mode
PSP_VPBE_DISPLAY_NTSC_INTERLACED_COMPOSITE	NTSC Composite in Interlaced
PSP_VPBE_DISPLAY_NTSC_INTERLACED_S_VIDEO	NTSC S-VIDEO in Interlaced
PSP_VPBE_DISPLAY_NTSC_INTERLACED_COMPONENT	NTSC Component in Interlaced
PSP_VPBE_DISPLAY_NTSC_PROGRESSIVE_COMPONENT	NTSC Component in Progressive
PSP_VPBE_DISPLAY_PAL_INTERLACED_COMPOSITE	PAL Composite in Interlaced
PSP_VPBE_DISPLAY_PAL_INTERLACED_S_VIDEO	PAL S-Video in Interlaced
PSP_VPBE_DISPLAY_PAL_INTERLACED_COMPONENT	PAL Component in Interlaced
PSP_VPBE_DISPLAY_PAL_PROGRESSIVE_COMPONENT	PAL Component in Progressive
PSP_VPBE_DISPLAY_YCC16_DIGITAL_480P	Digital YCC16 480P

1. For COMPOSITE mode output, connect cable to any of one DAC of DM6437 and observe the output on display screen.
2. For COMPONENT mode output, connect cable DAC B->Y, DAC C->Pb and DAC D ->Pr and observe the output on display screen.
3. For S_VIDEO mode output, connect S-video cable to s-video connector and observe the output on display screen.

NOTE: The hardware minimum clips the output by 0x18h in horizontal direction and 0x1h lines in vertical direction. The display screen clips as it output in VGA (640*480).

To adjust the VGA output in center, driver clips 0x7A in horizontal direction and 0x12h lines in vertical direction for NTSC (720*480) input. Driver clips 0x84h in horizontal direction and 0x16h lines in vertical direction for PAL (720*576) input.

CHAPTER 4

PORTING GUIDE

This section describes porting of VPBE driver on different TI platforms.

4.1 Porting Description

The figure below shows VPBE device driver architecture and changes those are required at the driver layers while porting VPBE device driver to any other Platform.

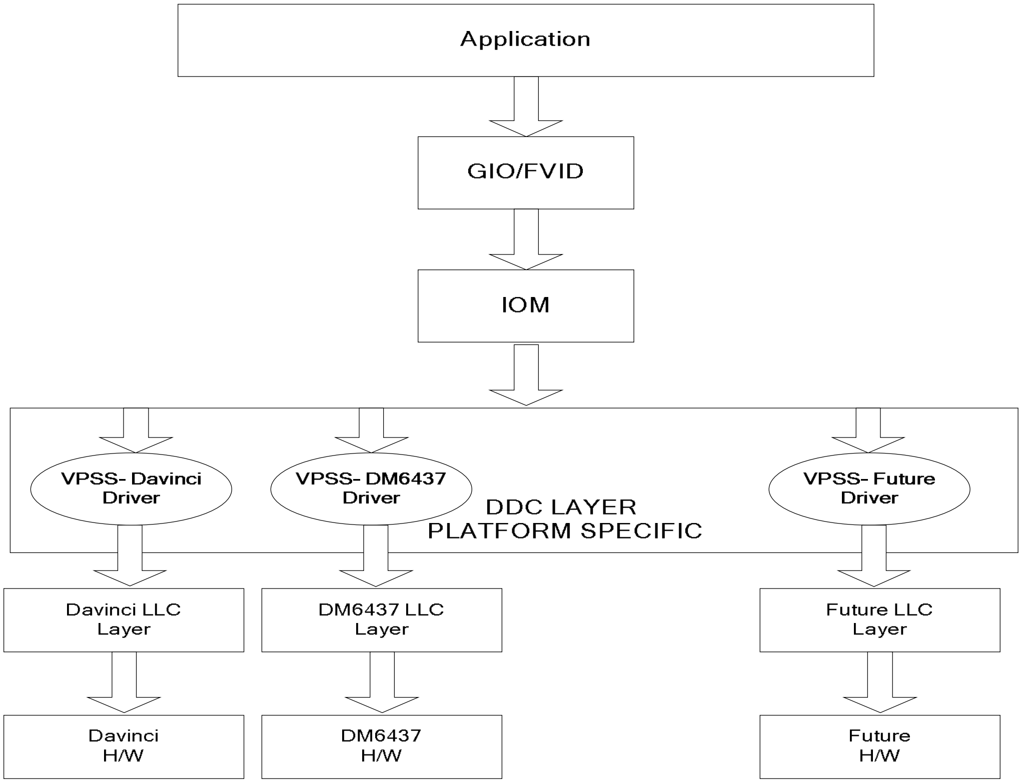


Figure 3 Driver Architecture

There will not be any change required in the GIO Layer, IOM Layer and DDA Layer while porting VPBE device driver on any TI platform. This Layer will be used as-is.

DDC Layer

The DDC layer is the core driver. This layer will exclusively contain the functionality related to a particular platform. At the DDC layer VPSS driver has divided CCDC/Video/OSD/Cursor and VENC into sub-components known as planes. VPBE and VPFE driver considers all this planes as separate hardware itself. Each of these planes has its own object which will contain the elements as per its functionality. These are the part of main parent object which will have all the parameters that are related to Video Front-end (VPFE) and Backend (VPBE) as whole.

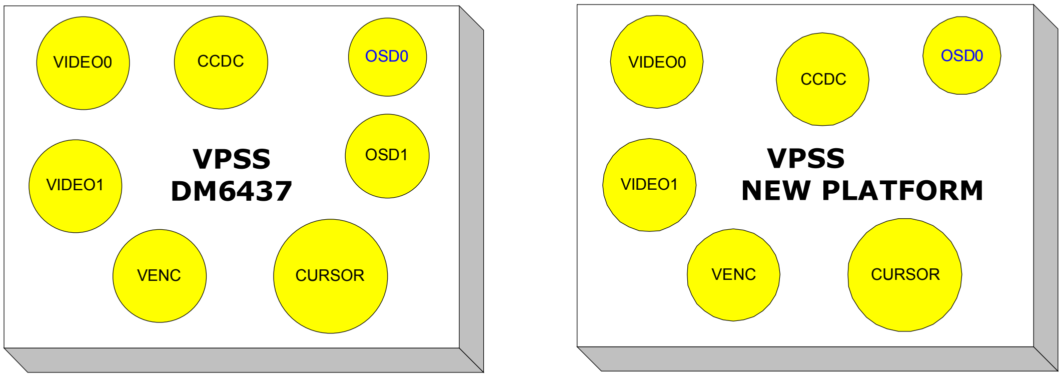


Figure 4 Driver Portability

As shown in above figure, DM6437 has OSD1 plane which is not present in the VPSS hardware on new Platform. While porting DM6437 VPBE driver on new platform you need to remove the OSD-1 plane sub-component object from the current DM6437 driver. Similarly in future, if there is any new sub-component then its corresponding object needs to be added in the driver object at this layer.

Hence in future platforms if there is a change in a way to perform a task the driver shall plug-in platform specific core for the new platform.

LLC Layer

This layer provides the abstraction to the Driver core on different platforms. This layer is specific to a specific platform. Mainly this layer should be having register overlaying, macro definitions. If not register overlaying is used then this layer will have low-level APIs to communicate with the hardware.
This layer should be having an as-is map of the peripheral device registers in the processor's memory map. Peripheral device registers map may differ from one platform to other. This change needs to be incorporated while porting driver code from once platform to another platform at LLC layer.

CHAPTER 5

HARDWARE DEPENDENCY

This section describes the hardware components that are not built inside the VPSS module and VPSS has dependency on such peripherals. For these scenarios the driver should have information's about certain parameters that it external peripheral needs

5.1 THS8200 Video Encoder

THS8200 is a complete video back-end D/A solution for any system requiring the conversion of digital components video signals into the analog domain.THS8200 can accept a variety of digital input formats, in both 4:4:4 and 4:2:2 format, over a 3 ·10-bit, 2 ·10-bit or 1 ·10-bit interface. The device synchronizes to incoming video data either through dedicated Hsync/Vsync inputs or through extraction of the sync information from embedded sync (SAV/EAV) codes inside the video stream. Alternatively, when configured for generating PC graphics output, THS8200 also provides a master timing mode in which it requests video data from an external (memory) source.

NOTE: THS8200 encoder is used when the VENC (VPBE module) is configured to give the Digital YCC16 480P output and the user wish to convert the digital data into analog domain. The THS8200 chip is connected on Sandwich board which comes along with EVMDm6437 board.

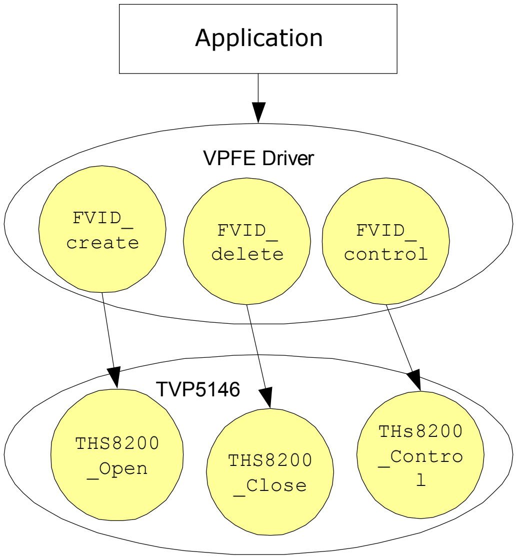


Figure 5 THS8200 Encoder Function compatible with VPBE driver

THS8200 video encoder is an independent interface which is called from the VPBE driver when VENC is configured in Digital YCC16 mode. As shown in above when the VPBE driver will call FVID_create to open VENC channel, it will call THS8200_Open function which will initialize the THS8200 and intialize I2C driver for serial communication. To configure the THS8200, application has to pass the PSP_VPSS_EXT_VIDEO_ENCODER_CONFIG Ioctl command by calling FVID_control. This will indirectly call THS8200_Control function. Once the VPBE driver deletes the channel, it will delete the THS8200 instance and close the I2C driver as well.

5.1.1. THS8200 Interface details

The THS8200 interfaces with the VPBE driver using 3 pointers to the functions as below:

- THS8200_Open
- THS8200_Close
- THS8200_Control

	Function	Function Description
1	THS8200_Open	☐ To initialize external ths8200 video encoder that is used by application
2	THS8200_Close	☐ To do the final house-keeping before the encoder is closed
4	THS8200_Control	☐ IOCTL to change ths8200 video encoder parameters runtime.

THS8200 video encoder peripheral registers are configured using I2C driver. THS8200 video encoder will act as Slave device. I2C will communicate with Ths8200 using the THS8200 video encoder slave address (0x5D).Refer to THS8200 specs for more detail.

THS8200_Open function is used to initialize the I2C driver. It configures the I2C for further register read and write of THS8200 video encoder. During open call application plugs-in the encoder functions in the function pointers provided in the front-end (external encoder) object.

During the close call the external encoder close function is called through the function pointer THS8200_close.

During the IOCTL calls the external encoder control function is called through the function pointer THS8200_Control.

Table below gives the details about the function pointers where-in the external encoder plugs-in.

#	Prototype
1	`EVENC_Handle  (*Open)()`
2	`Int         (*Close)(Ptr handle)`
3	`Int         (*Control)(Ptr handle,Uint32 Cmd,Ptr CmdArg)`

VPBE driver will communicate to external video encoder using the IOCTL commands as shown in below figure. Hence external encoder should provide the support to these functions that are required by the drivers as mandatory.

IOCTL Command	VALUE	PARAMS
PSP_VPSS_EXT_VIDEO_ENCODER_CONFIG	0x01	THS8200_ConfParams
PSP_VPSS_EXT_VIDEO_ENCODER_RESET	0x02	None

To configure the THS8200 video encoder application needs to create one object of THS8200_ConfParams structure.

Application needs to call VPBE Ioctl with PSP_VPSS_EXT_VIDEO_ENCODER_CONFIG Ioctl cmd to configure the THS8200 external video encoder and pass object of THS8200_ConfParams structure as cmdArg in the IOCTL.

Below is THS8200_ConfParams structure description:-

```
typedef struct THS8200_ConfParams_t
{
   THS8200_Mode          mode;     /**< Indicates resolution for THS8200                   */

   THS8200_InputFormat    iFmt;    /**< Selection for digital input format for THS8200  */

   THS8200_AnalogFormat    aFmt; /**< indicates analog output format for THS8200   */

   Bool               enableSlaveMode ;/**< Master or Slave mode                          */
   Bool               enableBT656Sync ;/**< embedded or external sync for digital input */
} THS8200_ConfParams;
```

Refer to pspdrivers\inc\ THS8200_extVidEncoder.h header file for more detail.

CHAPTER 6

EXAMPLE APPLICATIONS

This section describes the example applications that are included in the package. These sample application can be run as is for quick demonstration, but the user will benefit most by using these samples as sample source code in developing new applications.

5.1 Writing Applications for VPBE

This section provides guidance to user for writing their own application for VPFE driver

5.1.1. File Inclusion

To write sample application user has to include following header files in the application:

1. **psp_vpbe.h**

   This file contains the interfaces, data types and symbolic definitions that are needed by the application to utilizes the services of VPBE device driver.
2. **fvid.h**

   This file contains FVID layer macros. These macros are wrapper macros form a wrapper above GIO.
3. **fvid_evmDM6437.h**
   This file is provided to support compatibility to support previous DM6437 Video Releases

5.1.2. Driver Initialization

To use the VPBE device driver, a device entry must be added and configured in the DSP/BIOS configuration tool.

To have VPBE device driver included in the application, corresponding TCI file have to be included in BIOS TCF i.e. "*dm6437_vpbe0.tci*" must be included in BIOS TCF file of the application. This file can be found in video sample directory.

The VPBE driver initialization in BIOS TCF looks like the following:

```
bios.UDEV.create("VPBE0");
bios.UDEV.instance("VPBE0").fxnTable = prog.extern("VPBEMD_FXNS");
bios.UDEV.instance("VPBE0").fxnTableType = "IOM_Fxns";
```

Apart from the VPBE driver initialization, I2C driver should also be initialized in the BIOS TCF file. For details on how to initialize I2C driver, refer I2C driver user guide – *BIOS_I2C_Driver_UserGuide.pdf*.

5.1.3. Dependent Projects/Libraries

Following are the dependent libraries/projects to successfully build video application

- VPBE
- Video (for external encoders/decoders)
- I2C
- PAL_OS
- SoC specific PAL_SYS

5.1.4. Pragma directives used in the Applications

- DATA_ALIGN
  - Any buffer used for storing/retrieving data should be cache aligned at 128 bytes, since they write/read, to/from SDRAM/DDRAM.
  - The CCDC and OSD source and destination addresses should always be on 32-byte alignment.
  - DATA_ALIGN(4) is used in some places (for ex: params structures) in order to retain the 4-byte alignment even if padding switch is used in compiler options

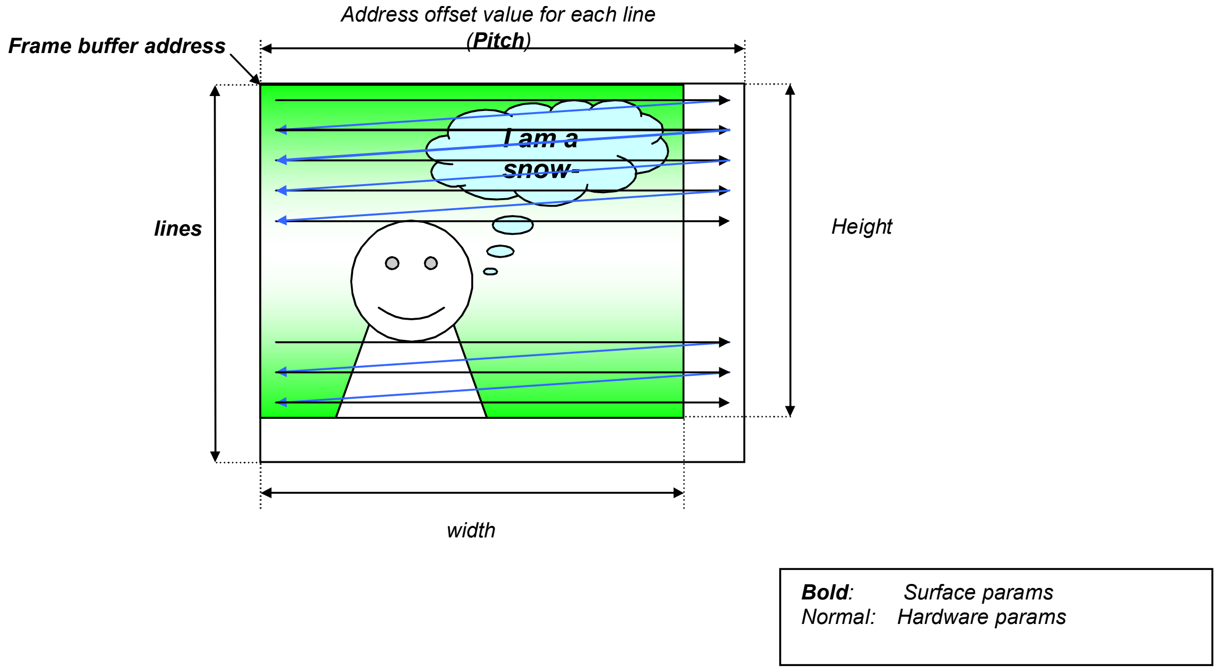
5.1.5. Memory Allocation

Memory allocation for video frame buffers has to be done by application. This frame buffers are queued in the drivers to facilitate the driver to capture. Configuration of surface params of frame like height, width and pitch has to be done by application. Buffer freeing should also be taken care by application.

The surface parameters of frame which are required for configuration are elements of FVID_Frame structure.

FVID_Frame Structure:

Parameter	Description
Pitch	Pitch or Offset of the Frame.
Lines	Number of lines of the Frame.
Frame.frameBufferPtr	FrameBuffer allocated by Application. **Note: Application allocates memory for the Frame buffer and passes it to driver.**
Bpp	Bits Per Pixel



FVID_allocBuffer function can be called from application to fill the FVID_Frame elements. This function fills the surface params along with allocating the memory to frame buffers. The information of height, width and pitch can be obtained from the handle that is passed as an argument to this function.

```
Refer to example code to alloc the FrameBuffer using FVID_allocBuffer
function.

FVID_Handle             vid0Handle;
FVID_Frame *FBAddr = NULL;

status = FVID_allocBuffer (vid0Handle, & FBAddr);
```

5.1.6. Buffer Management

Applications can capture the buffers by allocating the buffers and ‘queuing’ the buffers to facilitate the driver to capture. The driver captures the image on to the queued buffer and the filled buffer can be made use by the application by ‘de-queuing’ it.

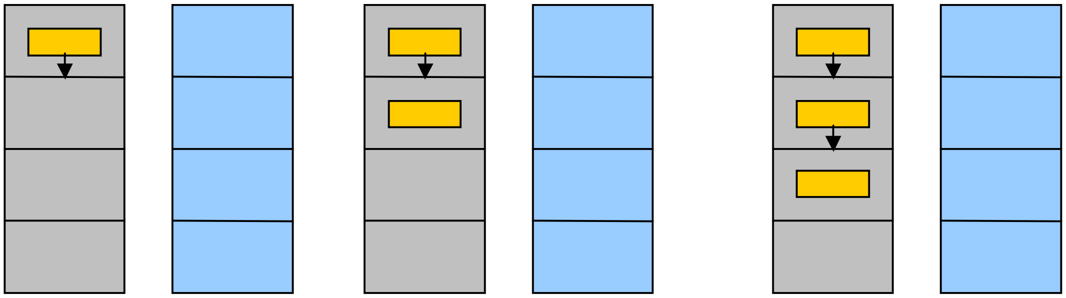


Figure. Operation of QUEUE call.

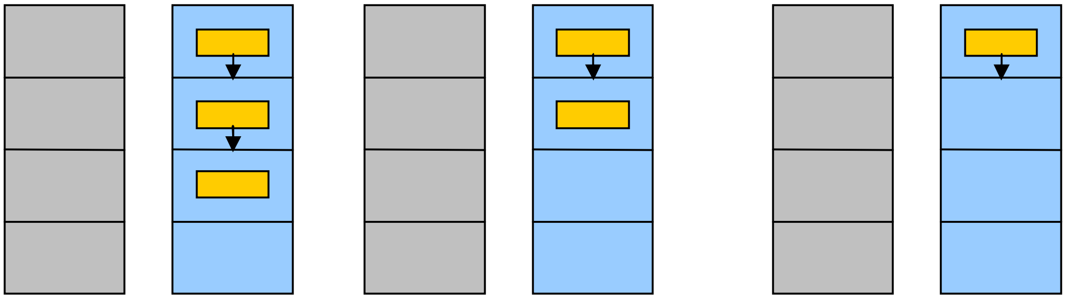


Figure: Operation of DE_QUEUE call.

▨ READY Queue

🟦 FREE Queue

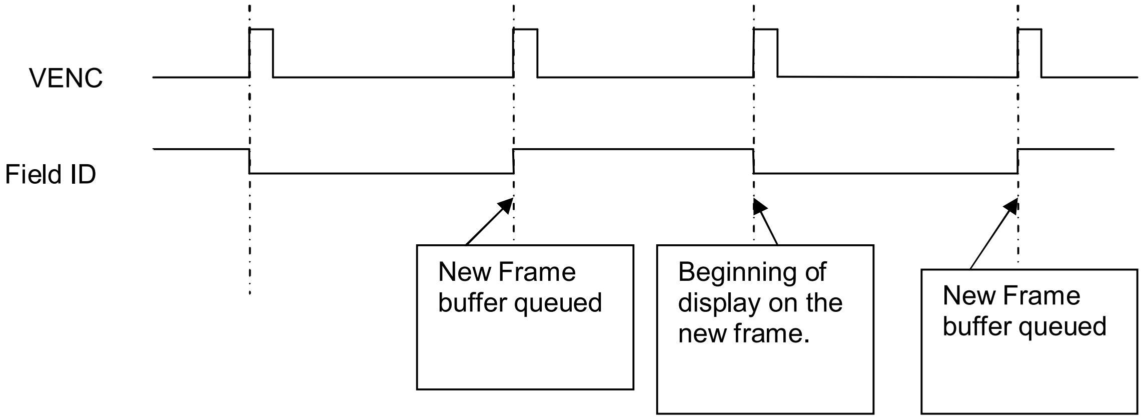
🟨 Buffers

The drivers maintain two circular queues called READY queue and a FREE queue for each channel of display and capture. The READY queue contains the buffer pointers of each of the buffers which are to be worked upon (passed to the driver using the QUEUE call), and the FREE buffer contains all the buffer pointers which have been worked upon. The buffers, when queued will be a part of the serial queue of READY buffers waiting to be worked on either by the display or for capture by their respective ISRs. Once worked upon, the driver checks if there are any more buffers in the queue to be worked on. If so, the worked buffer will move to a FREE queue, to be fetched by the application through DE_QUEUE. Once the driver comes to the last remaining buffer in the READY queue, the driver will loop over the same buffer to work upon till a new buffer is available. Since the driver cannot do away with this last remaining buffer, the DE_QUEUE call here will fail if issued

Application can call the FVID_exchange whose functionality is equivalent to serial calls of Queue and De-queue, in a single API call. The Exchange call will avoid the simultaneous call of Queue and Dequeue. Once In the exchange call, application passes the buffer pointers to queue them in READY Queue. Once buffer pointers are queued in READY queue, exchange call will wait for worked buffer which the driver will provide in the FREE queue once respective capture or display ISR is triggered.

ISR OPERATION

The ISR for VPBE is hooked to VENC interrupt which occurs at every VSYNC. The ISR handles queuing and de-queuing of the frame. ISR also calls an application callback at each vsync. Queuing of frame buffer is done at every even field (second field) of every frame for interlaced modes so that the buffer does not capture fields of different frames. To achieve this, the queuing is done at ISR of even field so that register is updated at the start of the next vsync which is also the beginning of the next frame.



5.2 The VPBE Sample Application

5.2.1. Introduction

The sample application will display shrek image on display screen. The shrek image is displayed on Video - 0 window. User can observe the one thick line moving from up to down in between the shrek image. This will continue for 30 seconds.

5.2.2. Building the Application

The sample application project file is located in the <root>\packages\ti\sdo\pspdrivers\system\dm6437\bios\evmDM6437\video\sample\build\vpbe folder. The sample can be rebuilt directly from this project file using Code Composer studio..

5.2.3. Loading the Application

The sample application is loaded and executed via Code composed studio. It is good idea to reset the board before loading Code Composer. The application will print out the status messages and type of functionality the driver performs on the Console output (HyperTerminal).

Note:-It is required to connect display screen (TV) input to Video out connector (DAC B) on the board. The Video -0 Window is configured in order to display the captured image on the display screen.

Following are the configuration parameters used to configure Video- o window

```
static PSP_VPBEOsdConfigParams       vid0Params =
 {
       FVID_FRAME_MODE,                 /* ffmode        */
       FVID_BPP_BITS16,                 /* bitsPerPixel  */
       FVID_YCbCr422_INTERLEAVED,       /* colorFormat   */
       (720 *       (16/8u)),           /* pitch         */
       {
           720,                        /* width         */
           480,                        /* height        */
           0,                          /* leftMargin    */
           0,                          /* topMargin     */
         }
       PSP_VPBE_ZOOM_IDENTITY,         /* hScaling      */
       PSP_VPBE_ZOOM_IDENTITY,         /* vScaling      */
       PSP_VPBE_EXP_IDENTITY,          /* hExpansion    */
       PSP_VPBE_EXP_IDENTITY,          /* vExpansion    */
       NULL                            /* appCallback   */
   };
```

The Venc module is configured to output captured data on DAC and display it on Display screen.

```
static PSP_VPBEVencConfigParams vencParams =
{
       PSP_VPBE_DISPLAY_NTSC_INTERLACED_COMPOSITE /* Display Standard */
};
```

5.2.4. Usage of OSD Windows

There are 5 OSD windows that VPBE driver supports. These windows are video -0, video-1, OSD-0 ,OSD-1 and cursor window. The video-0 and video-1 windows supports YCbCr422,YCrCb422,RGB888,YCbCr444 modes.Video-0 and video-1 window can be configured in any of the supported modes. The OSD-0 and OSD-1 windows supports RGB565,CLUT_INDEXED.The OSD-1 window supports Attribute mode.

Refer to examples below which describes how the configure these windows

Configuration of Video-0 and Video-1 window

```
FVID_Handle          Vid0Handle;
GIO_Attrs gioAttrs = GIO_ATTRS;

PSP_VPBEChannelParams    beinitParams;
PSP_VPBEOsdConfigParams vid0Params;
static PSP_VPBEOsdConfigParams      vid0Params =
 {
     FVID_FRAME_MODE,               /* ffmode        */
     FVID_BPP_BITS16,               /* bitsPerPixel  */
     FVID_YCbCr422_INTERLEAVED,     /* colorFormat   */
     (720 *     (16/8u)),           /* pitch         */
     {
         720,                       /* width         */
         480,                       /* height        */
         0,                         /* leftMargin    */
         0,                         /* topMargin     */
       }
     PSP_VPBE_ZOOM_IDENTITY,        /* hScaling      */
     PSP_VPBE_ZOOM_IDENTITY,        /* vScaling      */
     PSP_VPBE_EXP_IDENTITY,         /* hExpansion    */
     PSP_VPBE_EXP_IDENTITY,         /* vExpansion    */
     NULL                           /* appCallback   */
}
/**
  * Create the Video-0 Channel
**/
beinitParams.id          = PSP_VPBE_VIDEO_0;
beinitParams.params      = (PSP_VPBEOsdConfigParams    *) &vid0Params;
Vid0Handle  = FVID_create ("/VPBE0", IOM_OUT, NULL,
                              & beinitParams, &gioattrs);
```

To configure the Video-1 window pass PSP_VPBE_VIDEO_1 as parameter id in PSP_VPBEChannelParams structure and rest all the configuration remain same as video-0 window configuration.

Configuration of OSD- 0 and OSD-1 window

```
FVID_Handle          Osd0Handle;
GIO_Attrs gioAttrs = GIO_ATTRS;

PSP_VPBEChannelParams    beinitParams;
PSP_VPBEOsdConfigParams  osd0Params;
static PSP_VPBEOsdConfigParams       osd0Params =
 {
        FVID_FRAME_MODE,                 /* ffmode        */
        FVID_BPP_BITS16,                 /* bitsPerPixel  */
        FVID_RGB565_INTERLEAVED,         /* colorFormat   */
        (720 *      (16/8u)),            /* pitch         */
        {
             720,                        /* width         */
             480,                        /* height        */
             0,                          /* leftMargin    */
             0,                          /* topMargin     */
           }
        PSP_VPBE_ZOOM_IDENTITY,          /* hScaling      */
        PSP_VPBE_ZOOM_IDENTITY,          /* vScaling      */
        PSP_VPBE_EXP_IDENTITY,           /* hExpansion    */
        PSP_VPBE_EXP_IDENTITY,           /* vExpansion    */
        NULL                             /* appCallback   */
}
/**
  * Create the OSD-0 Channel
**/
beinitParams.id     = PSP_VPBE_OSD_0;
beinitParams.params = (PSP_VPBEOsdConfigParams    *) &osd0Params;
Osd0Handle          = FVID_create (“/VPBE0”, IOM_OUT, NULL,
                                & beinitParams, &gioattrs);
```

To configure the OSD-1 window pass PSP_VPBE_OSD_0 as parameter id in PSP_VPBEChannelParams structure and rest all the configuration remain same as osd-0 window configuration.

Configuration of Cursor Window

```
FVID_Handle                    CursorHandle;
GIO_Attrs gioAttrs = GIO_ATTRS;
PSP_VPBEChannelParams    beinitParams;

static PSP_VPBECursorConfigParams   cursorParams =
 {
          90,                                          /* leftMargin      */
          90,                                          /* topMargin       */
          240,                                         /* width           */
          160,                                         /* height          */
          PSP_VPBE_CLUTSOURCE_RAM        /*  CLUT Source */
};

/**
  * Create CURSOR Channel
 **/
   beinitParams.id           = PSP_VPBE_CURSOR;
   beinitParams.params  = (PSP_VPBECursorConfigParams *)&cursorParams;
   CursorHandle            = FVID_create("/VPBE0",IOM_INOUT,NULL,
                                                       &beinitParams,&gioAttrs);
```

Appendix A

The example code of vpbe which copies the shrek image from the memory and continously display the image on display screen.One line will also be observed which will move from up to down.The display output mode selected is Interlaced composite.To run this sample application connect any of DAC of DM6437 to display screen.

```
#include <std.h>
#include <gio.h>
#include <mem.h>
#include <string.h>
#include <bcache.h>

#include "fvid.h"
#include "psputils.h"
#include "psp_vpbe.h"
#include "yuv_imageCbCr.h"

extern Int      DDR2;

/*Global Variable Defined */
static GIO_Handle   Vid0Handle;
static GIO_Handle   VencHandle;
static Ptr          vidRefFrameBuffer    = NULL;
static FVID_Frame *VidallocFB = NULL;

static PSP_VPBEOsdConfigParams  vid0Params =
{
    FVID_FRAME_MODE,                    /* ffmode       */
    FVID_BPP_BITS16,                    /* bitsPerPixel */
    FVID_YCbCr422_INTERLEAVED,          /* colorFormat  */
    (720 *  (16/8u)),                   /* pitch        */
    0,                                  /* leftMargin   */
    0,                                  /* topMargin    */
    720,                                /* width        */
    480,                                /* height       */
    0,                                  /* segId        */
    PSP_VPBE_ZOOM_IDENTITY,             /* hScaling     */
    PSP_VPBE_ZOOM_IDENTITY,             /* vScaling     */
    PSP_VPBE_EXP_IDENTITY,              /* hExpansion   */
    PSP_VPBE_EXP_IDENTITY,              /* vExpansion   */
    NULL                                /* appCallback  */

};
static PSP_VPBEVencConfigParams vencParams =
{
    PSP_VPBE_DISPLAY_NTSC_INTERLACED_COMPOSITE /* Display Standard */
};
```

```
/**
 * Insert a line in the Frame
**/
static void prepareVidFrame(Ptr fbAddr)
{
    Uint8 * vidiPtr = NULL;
    static Uint vidlineNo = 0;

    vidlineNo = vidlineNo % (475);
    vidiPtr = (Uint8 *)fbAddr;
    if(NULL!=vidiPtr)
    {
        memcpy(vidiPtr,vidRefFrameBuffer,(720*480*2));
        vidiPtr = vidiPtr + (1440 * vidlineNo);
        memset(vidiPtr,0xFFFFFFFF,(5*1440));
        vidlineNo++;
    }
}
/**
 * Main Function
**/
static void vpbe_main()
{
      PSP_VPBEChannelParams   beinitParams;
      GIO_Attrs               gioAttrs = GIO_ATTRS;
      Uint32 * iPtr = NULL;
      Uint32   NumOfIterations = 1000;
      Uint32   index              = 0;
      Uint32   i  =     0;

    VPSS_DBG ("VPSS: VPBE Sample Application Started \r\n");

    vidRefFrameBuffer   = MEM_alloc(DDR2, (720*480*2) , 32u);
      iPtr              = (Uint32 *)vidRefFrameBuffer;
      if(NULL!=iPtr)
      {
        /*Store Image into Memory */
         for(index = 0;index<((720*480*2)/(sizeof(Uint32))); index++)
         {
            *(iPtr + index) = yuv_imageCbCr_720x480 [index];
          }
       }
    /**
     * Create Video Channel
    **/
    beinitParams.id     = PSP_VPBE_VIDEO_0;
    beinitParams.params = (PSP_VPBEOsdConfigParams    *)&vid0Params;
    Vid0Handle=  FVID_create("/VPBE0",IOM_INOUT,NULL,
                              &beinitParams,&gioAttrs);
    if(NULL == Vid0Handle)
    {
        VPSS_DBG("VPSS  :VIDE0 -0 Create.......FAILED \r\n");
        VPSS_DBG("VPSS  :End of VPSS Loopback Application\r\n");
        return;
    }
    else
    {
        if(IOM_COMPLETED == FVID_allocBuffer(Vid0Handle,&VidallocFB))
        {
          memcpy(VidallocFB- >frame.frameBufferPtr,vidRefFrameBuffer,
                                                     (720*480*2));
          if(IOM_COMPLETED != FVID_queue(Vid0Handle,VidallocFB))
          {
            VPSS_DBG("VPSS  :Video 0 Queuing.......FAILED \r\n");
          }
        }
    }
    /**
     * Create Venc Channel
    **/
    beinitParams.id          = PSP_VPBE_VENC;
    beinitParams.params      = (PSP_VPBEVencConfigParams *)&vencParams;
    VencHandle               = FVID_create ("/VPBE0",IOM_INOUT,NULL,
                                           &beinitParams,&gioAttrs);
    if (NULL == VencHandle)
    {
        VPSS_DBG ("VPSS  :Venc Create ... FAILED!\r\n");
        VPSS_DBG ("VPSS  :End of VPSS Loopback Application\r\n");
        return;
    }
    for (i = 0; i < NumOfIterations; i++)
    {
      prepareVidFrame (VidallocFB->frame.frameBufferPtr);
      if (IOM_COMPLETED ==FVID_queue(Vid0Handle,VidallocFB))
       {
         if (IOM_COMPLETED != FVID_dequeue(Vid0Handle,&VidallocFB))
         {
            VPSS_DBG ("VPSS  :Dequeue Call.......FAILED \r\n");
          }
       }
    }
   /**
    *  Free Memory Buffers
    **/
    FVID_freeBuffer(Vid0Handle, VidallocFB);
    MEM_free (DDR2,vidRefFrameBuffer,(720*480*2));
    VPSS_DBG ("VPSS: VPBE Sample Application Ended \r\n");
}
```

```
/**
 * Function call
**/
void start_vpss_test()
{
      vpbe_main();
      return;
}
```