Texas Instruments

DSP/BIOS VPFE Device Driver

User's Manual

June 2008 *Platform Software Group*

Document Version 1.14

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

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

Read This First

About This Manual

The API reference guide serves as a software programmer's handbook for working with the VPFE 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
September 21, 2006	Maulik Desai	Added Data Structure and Enumerations	1.1
November 30, 2006	Maulik Desai	Modified the document for release 0.3.0	1.2
December 16, 2006	Maulik Desai	Modified the document to close review comments	1.3
January 2, 2007	Maulik Desai	Modified the documents to close review comments	1.4
January 16, 2007	Maulik Desai	Bios version modified	1.5
February 3, 2007	Maulik Desai	CCS version modified	1.6
April 25, 2007	Maulik Desai	FVID layer modifications	1.7
May 5, 2007	Maulik Desai	IOCTL Description Added	1.8
May 15, 2007	Maulik Desai	FVID layer modifications	1.9
June 22, 2007	Anuj Aggarwal	Bios version modified	1.10
June 29,2007	Amit Chatterjee	Modified Revision Version	1.11
July 1, 2007	Maulik Desai	MT9001 IOCTL added	1.12
November 30, 2007	Sivaraj R	PSP merge package changes - directory structure changes, FVID_allocBuffer and FVID_freeBuffer functions are implemented as GIO control commands	1.13
January 24, 2008	Sivaraj R	Added TCI file driver initialization illustration and added dependent libraries for building video application	1.14
May 21, 2008	Chandan Nath	Updated for adding compiler switches in build options	1.15

TABLE OF CONTENTS

Preface 3
Abbreviations 3
Revision History 4
TABLE OF CONTENTS 5
List Of Figures 7
CHAPTER 1 8
CHAPTER 1 8
INTRODUCTION 8
1.1 H/W S/W Support 8
1.2 Driver Components 8
1.3 Default Driver Configuration 9
1.4 Driver Capabilities 10
1.5 System Requirements 10
CHAPTER 2 11
INSTALLATION GUIDE 11
2.1 Component Folder 11
2.2 Build 12
2.3 Build Options 12
CHAPTER 3 14
DSP/BIOS VPFE 14
3.1 Functions 14
3.1.1. GIO_create/FVID_create 14
3.1.2. GIO_delete/FVID_delete 16
3.1.3. GIO_control/FVID_control 16
3.1.4. GIO_submit 18
3.2 Control Commands 23
3.2.1. PSP_VPFE_IOCTL_CCDC_VALIDATE_BUFFER 23
3.2.2. PSP_VPFE_IOCTL_CCDC_BLACK_ADJUST 23
3.2.3. PSP_VPFE_IOCTL_CCDC_FPC 24
3.2.4. PSP_VPFE_IOCTL_START_CCDC 25
3.2.5. PSP_VPFE_IOCTL_STOP_CCDC 26
3.2.6. PSP_VPFE_IOCTL_FIELD_INTERLACED 26
3.2.7. PSP_VPFE_IOCTL_FIELD_SEPARATE 27
3.2.7. PSP_VPFE_IOCTL_CONFIG_VDINT 28
3.3. Data Structures Configuration defines 29
3.4 Enumerations 32
CHAPTER 4 34
PORTING GUIDE 34
4.1 Porting Description 34
CHAPTER 5 36
HARDWARE DEPENDENCY 36
5.1 TVP5146 Video Decoder 36
5.1.1. TVP5146 Interface details 37
5.2 MT9001 Image Sensor 39
5.2.1. MT9001 Sensor Interface details 40
CHAPTER 6 44
EXAMPLE APPLICATIONS 44

6.1 Writing Applications for VPFE 44
6.1.1. File Inclusion 44
6.1.2. Driver Initialization 45
6.1.3. Dependent Projects/Libraries 45
6.1.4. Pragma directives used in the Applications 45
6.1.5. Memory Allocation 45
6.1.6. Buffer Management 47
6.2 The VPFE YUV Sample Application 49
6.2.1. Introduction 49
6.2.2. Building the Application 49
6.2.3. Loading the Application 49
6.2.4. Configuration Parameters 49
6.3 The VPFE RAW Sample Application 50
6.3.1. Introduction 50
6.3.2. Building the Application 51
6.3.3. Loading the Application 51
6.3.4. Configuration Parameters 51
6.4 The VPFE PREVIEWER on-the-fly Sample Application 54
6.4.1. Introduction 54
6.4.2. Building the Application 54
6.4.3. Loading the Application 54
6.4.4. Configuration Parameters 54
Appendix A 56

List Of Figures

Figure 1 Device Driver Functional Decomposition 9
Figure 2 VPFE Driver Directory Structure 11
Figure 3 Driver Architecture 34
Figure 4 Driver Portability 35
Figure 5 TVP5146 Decoder Function compatible with VPFE driver 36
Figure 6 MT9001 Sensor Function compatible with VPFE driver 39

CHAPTER 1

INTRODUCTION

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

1.1 H/W S/W Support

This VPFE 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:

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

VPFE DDC –OS Independent part of VPFE Driver Core

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

System components:

PALOS – DSP/BIOS Abstraction

CSLR– Non-Functional h/w abstraction.

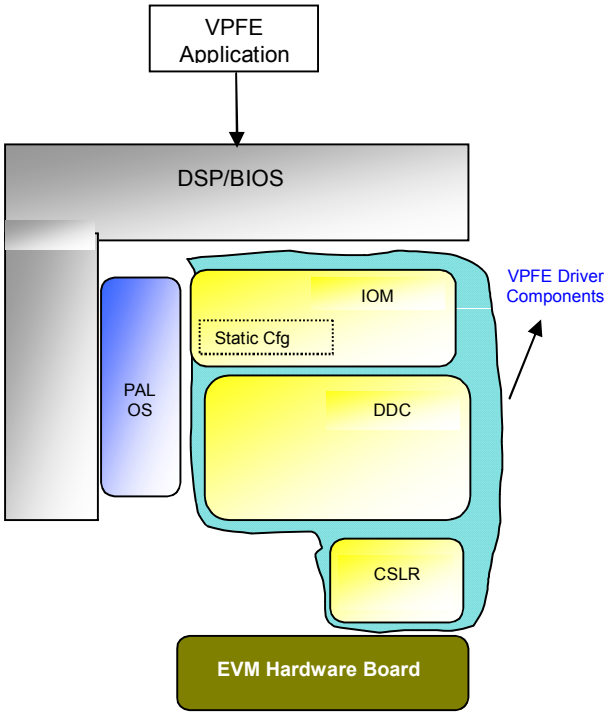


Figure 1 Device Driver Functional Decomposition

1.3 Default Driver Configuration

VPFE 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 CCDC and other Front End modules.
- Driver is SYNCHRONOUS and operates in INTERRUPT mode only.
- Supports flipping of multiple frame buffers for seamless capture from CCDC.
- Easy to maintain & re-target to new platforms.
- Supports Multiple Instances.

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 VPFE driver the following directory structure is found in the driver's directory.

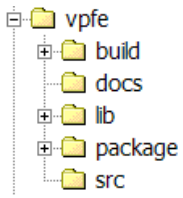


Figure 2 VPFE Driver Directory Structure

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

- **build:** This folder contains vpfe driver library project file. The generated driver library shall be included in the application where VPFE 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 vpfe 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/vpfe*" folder and the driver library is provided in */drivers/vpfe/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.
- **VPFE_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

- **PSP_VIDEO_PATH_ENABLE**

  Enable Video Path for H3A, HISTOGRAM and PREVIEWER (on-the-fly) else directly send the data to the resizer (VPEN is reset)

CHAPTER 3

DSP/BIOS VPFE

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

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

This function is called by the application to create the various VPFE channel. VPFE driver supports CCDC module

It populates static settings in driver object; formally creates/registers driver entry points with DSP/BIOS. This call registers interrupt for CCDC.

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
attrs invalid parameters then VPFE driver shall return NULL and the corresponding channel shall not be created
[IN] pointer to GIO_attrs structure

Returns:

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

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

Example:

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

The FVID call in a way make the GIO call.

The example below shows creation of CCDC Channel for VPFE

FVID_Handle vpfeCcdcHandle
GIO_Attrs gioAttrs = GIO_ATTRS;

PSP_VPFEChannelParams  feinitParams;
static PSP_VPFECcdcConfigParams ccdcParams =
{
    FVID_VI_BT656_8BIT,          /* dataFlow     */
    FVID_FRAME_MODE,             /* ffMode       */
    480,                         /* height       */
    720,                         /* width        */
    (720 *2),                    /* pitch        */
    0,                           /* horzStartPix */
    0,                           /* vertStartPix */
    NULL,                        /* appCallback  */
    {
        TVP5146_Open,            /* extVD Fxn    */
        TVP5146_Close,
        TVP5146_Control,
    },
    0                            /*segId         */
};
feinitParams.id         = PSP_VPFE_CCDC;
feinitParams.params     = (PSP_VPFECcdcConfigParams*)&ccdcParams;
CcdcHandle              = FVID_create("/VPFE0",IOM_INOUT,NULL,
                                      &feinitParams,&gioAttrs);
if(NULL == CcdcHandle)
{
        printf("VPSS  :CCDC Create.......FAILED \r\n");
        return;
}
```

Name is passed as VPFE0 whose instance is declared in the configuration file

3.1.2. GIO_delete/FVID_delete

Int FVID_delete (FVID_Handle gioChan,

OR
Int GIO_delete)

This function deletes the driver channel.

Parameters:
gioChan [IN] FVID Handle for the channel

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

Example:

```
FVID_Handle  CcdcHandle;

FVID_delete (CcdcHandle);
```

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 VPFE 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                    CcdcHandle;
PSP_VPFEConfigParams     ccdcParams;
PSP_VPFECCDC_BlackAdj  blackAdj;

static Ptr InitBlackCompParameters(void)
{

   blackAdj.clampEn              = PSP_VPFE_BlkCompDisable;
   blackAdj.blackSampLen         = PSP_VPFE_BlkComp1Pixel;
   blackAdj.blackSampLine        = PSP_VPFE_BlkComp1Pixel;
   blackAdj.blackPixStart        = 0;
   blackAdj.blackGain            = 0;
   blackAdj.dcSub                = 20;
   blackAdj.colPtn               = 0;
   blackAdj.blkCmpR              = 0;
   blackAdj.blkCmpGR             = 0;
   blackAdj.blkCmpGB             = 0;
   blackAdj.blkCmpB              = 0;
   return ((Ptr)&blackAdj);
}
/** DC Clamping in Raw Capture */
if(IOM_COMPLETED!=FVID_control(CcdcHandle,PSP_VPFE_IOCTL_CCDC_BLACK_ADJUST,
                                          blackAdjl ))
 {
     printf ("VPSS  :Error in DCSUB  !\r\n");
 }
 else
 {
     printf("VPSS  :DC Clamp done Successfully !\r\n");
 }
```

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 VPFE 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 VPFE 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      CcdcHandle;
FVID_Frame       *CcdcallocFB = NULL;

/* Allocate memory to Frame Buffer         */
      FVID_allocBuffer (CcdcHandle, &CcdcallocFB);
/* Queue the Frame Buffer after allocation */
      FVID_queue (vpfeCcdcHandle, CcdcallocFB);
```

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 VPFE 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 VPFE 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            CcdcHandle;
FVID_Frame *allocBF = NULL;

status = FVID_dequeue (CcdcHandle, &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 VPFE 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            CcdcHandle;
FVID_Frame *CcdcallocFB = NULL;
FVID_Frame *FBAddr      = NULL;

/* Allocate memory to Frame Buffer        */
      FVID_allocBuffer (CcdcHandle, &CcdcallocFB);
/* Queue the Frame Buffer after allocation */
      FVID_queue (vpfeCcdcHandle, CcdcallocFB);
/* Allocate memory to Frame Buffer        */
   FVID_allocBuffer (CcdcHandle, & FBAddr);
/* Exchange the Frame Buffers             */
status = FVID_exchange (CcdcHandle, &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 VPFE 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 alloc the FrameBuffer using FVID_allocBuffer
function.

FVID_Handle            CcdcHandle;
FVID_Frame *FBAddr = NULL;

status = FVID_allocBuffer (CcdcHandle, & 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 CcdcHandle;
FVID_Frame *FBAddr = NULL;

status = FVID_allocBuffer (CcdcHandle, & FBAddr);

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

3.2 Control Commands

3.2.1. PSP_VPFE_IOCTL_CCDC_VALIDATE_BUFFER

PSP_VPFE_IOCTL_CCDC_VALIDATE_BUFFER – Validate the CCDC 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 CCDC module
  - **cmd** – PSP_VPFE_IOCTL_CCDC_VALIDATE_BUFFER
  - **cmdArg** – Pointer to PSP_VPFEValidateParams 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_VPFEValidateParams structure. An application can read the second element of PSP_VPFEValidateParams structure to know the expected value of framebuffer pitch.

  The structure is described as below:

```
typedef struct _PSP_VPFEValidateParams
{
   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_VPFEValidateParams;
```

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.2. PSP_VPFE_IOCTL_CCDC_BLACK_ADJUST

PSP_VPFE_IOCTL_CCDC_BLACK_ADJUST – Compensate and adjust the black level related parameters. It is used when the input to CCDC is from Image sensor i.e., raw input.

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

- **ARGUMENTS**
  - **gioChan** – Handle of CCDC module
  - **cmd** – PSP_VPFE_IOCTL_CCDC_BLACK_ADJUST
  - **cmdArg** – Pointer to PSP_VPFECCDC_BlackAdj Structure

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL

- **DESCRIPTION**
  The black level adjustment and compensation is done dynamically using this Ioctl. This Ioctl configures the related parameters for adjusting and compensating the black level. The Ioctl should be used only when input to CCDC is raw input.

  The structure is described as below:

```
typedef struct _PSP_VPFECCDC_BlackAdj
{
   PSP_VPFE_BlkComp              clampEn;
        /**<  Enable/disable black level clamping                          */
   PSP_VPFE_BlkComp_Pixlen       blackSampLen;
   /**<  Num Black Sample pixels per line to include in the calculation  */
   PSP_VPFE_BlkComp_Pixlen       blackSampLine;
   /**<  Num Black Sample lines to include in the calculation        */
   Uint16     blackPixStart;
   /**<  Start black pixel position                                        */
   Uint8      blackGain;
   /**<  Gain to be applied to optical black average                 */
   Uint16     dcSub;
   /**<  Constant DC value to subtract if clampEn is disabled        */
   Uint32     colPtn;
   /**<  CCDC color pattern                                          */
   Uint8      blkCmpR;
   /**<  Black level compensation for R color pixels                 */
   Uint8      blkCmpGR;
   /**<  Black level compensation for GR color pixels                */
   Uint8      blkCmpGB;
   /**<  Black level compensation for GB color pixels                */
   Uint8      blkCmpB;
   /**<  Black level compensation for B color pixels                 */
} PSP_VPFECCDC_BlackAdj;
```

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.3. PSP_VPFE_IOCTL_CCDC_FPC

PSP_VPFE_IOCTL_CCDC_FPC – This Ioctl is used for setting Fault pixel correction related parameters.

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

- **ARGUMENTS**
  - **gioChan** – Handle of CCDC module
  - **cmd** – PSP_VPFE_IOCTL_CCDC_FPC
  - **cmdArg** – Pointer to PSP_VPFE_CCDC_Fpc structure

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL.
- **DESCRIPTION**

  This IOCTL sets the Fault pixel correction related parameters. If the image sensor is faulty and giving some faulty pixel then by calling these IOCTL the fault pixel correction module is enabled which will correct the faulty pixel received from Image sensor. The IOCTL is used only when the input is from Image sensor i.e.; raw input.

  The structure is described as below:

```
typedef struct _PSP_VPFE_CCDC_Fpc
{
   PSP_VPFE_Fpc       fpcEnable;
  /**< CCDC Fault Pixel Correction enable/disable    */
   Uint8              fpcnum;
   /**< number of Fault Pixels to be corrected                  */
   Uint32             fpcAddr;
   /**< Address of the FPC Table                                */
   PSP_VPFE_Err       fpcreset;
   /**< use this parameter to reset the FPC error      */
} PSP_VPFE_CCDC_Fpc;
```

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.4. PSP_VPFE_IOCTL_START_CCDC

PSP_VPFE_IOCTL_CCDC_START_CCDC –This Ioctl is used to start the CCDC Engine.

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

- **ARGUMENTS**
  - **gioChan** – Handle of CCDC module
  - **cmd** – PSP_VPFE_IOCTL_CCDC_START_CCDC
  - **cmdArg** – NULL

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL.

- **DESCRIPTION**

  This IOCTL starts the CCDC engine. It is required to queue one buffer initially before calling these IOCTL.

- **LIMITATIONS/CONSTRAINTS**
  None

3.2.5. PSP_VPFE_IOCTL_STOP_CCDC

PSP_VPFE_IOCTL_CCDC_FPC – This Ioctl is used to stop the CCDC Engine.

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

- **ARGUMENTS**
  - **gioChan** – Handle of CCDC module
  - **cmd** – PSP_VPFE_IOCTL_STOP_CCDC
  - **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 CCDC Engine.

- **LIMITATIONS/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 buts 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.

3.2.6. PSP_VPFE_IOCTL_FIELD_INTERLACED

PSP_VPFE_IOCTL_FIELD_INTERLACED – This Ioctl is used to get the odd and even field interleaved

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

- **ARGUMENTS**
  - **gioChan** – Handle of CCDC module
  - **cmd** – PSP_VPFE_IOCTL_FIELD_INTERLACED
  - **cmdArg** – NULL

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL.

- **DESCRIPTION**
  This IOCTL will give the odd and even field data interleaved one after other.

- **LIMITATIONS/CONSTRAINTS**
  NONE

3.2.7. PSP_VPFE_IOCTL_FIELD_SEPARATE

PSP_VPFE_IOCTL_FIELD_SEPARATE – This Ioctl is used to get the odd and even field separate.

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of CCDC module
  - **cmd** – PSP_VPFE_IOCTL_FIELD_SEPARATE
  - **cmdArg** – NULL

- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL.

- **DESCRIPTION**
  This IOCTL will get the odd and even field data separate. The odd field data will be on the top half of the frame and even field data will be on bottom half.

- **LIMITATIONS/CONSTRAINTS**
  NONE

3.2.7. PSP_VPFE_IOCTL_CONFIG_VDINT

PSP_VPFE_IOCTL_CONFIG_VDINT – This Ioctl is used to configure the VSYNC interrupt

- **SYNOPSIS**
  - Int FVID_control (FVID_Handle gioChan, Int cmd, Ptr cmdArg);
- **ARGUMENTS**
  - **gioChan** – Handle of CCDC module
  - **cmd** – PSP_VPFE_IOCTL_CONFIG_VDINT
  - **cmdArg** – Number of lines
- **RETURN VALUE**
  - IOM_COMPLETED – Success if the args are passed properly.
  - IOM_EBADARGS – if args passed is NULL.
- **DESCRIPTION**
  This IOCTL is used to configure the VSYNC interrupt. Configure the VSYNC interrupt at specific number of lines.
- **LIMITATIONS/CONSTRAINTS**
  NONE

3.3. Data Structures Configuration defines

The file **psp_vpfe.h** has the **PSP_VPFEChannelParams** data structure that is passed at the time of opening the CCDC Channel from the Application .The params are explained below:

1) Table Configuration Data Structure – PSP_VPFEChannelParams

Parameter	Description
Id	Channel Id Note: Each VPFE modules are treated as individual channels
Params	Pass the Structure of the Channel .i.e;PSP_VPFECcdcConfigParams

The file **psp_vpfe.h** has the **PSP_VPFECcdcConfigParams** data structure that is parameter of VPFEChannelParams structure mentioned above. The params are explained below:

2) Table Configuration Data Structure – PSP_VPFECcdcConfigParams

Parameter	Description
inpmod	Specify the input to CCDC –Sensor(RAW) or Video Decoder(YCbCr)
ffMode	FILED/FRAME Mode **Note:** Here field mode refers to Progressive and Frame mode refers to Interlaced mode.
height	Height of the Frame i.e; vertical lines of the Frame
width	Width of the Frame i.e; Number of pixel in Horizontal direction
Pitch	Pitch or offset of the Frame.
horzStartPix	Horizontal start pixel
vertStartPix	Start of Vertical line
appCallback	Application callback Function
extVDFxn	External Video Decoder Interface(TVP5146 or Image Sensors)
segId	SegId is passed to allocate memory
rawParams	Configure the Raw Config Params. Note: Used when Input is from Sensor (RAW).

The file **psp_vpfe.h** has the **PSP_VPFECCDCRawParams** data structure that is parameter of **PSP_VPFECcdcConfigParams** structure mentioned above. The params are explained below:

Parameter	Description
dataSize	Specify the size of Raw Data.
pack8	Data stored in pack 8 or 16 bits per pixel
dataPol	Data Polarity Normal or 1's Complement
VDSyncPol	VD Sync Polarity
HDSyncPol	HD Sync Polarity
HDVDMaster	Master or Slave mode for HD and VD signals
HDSyncWidth	Width of HD Sync
VDSyncWidth	Width of VD Sync
numPxlPerLine	Number of Pixelclock periods in one line
numLinPerFld	Number of lines per field
ALawEnable	Enable/Disable A-Law
ALaw_Width	A-Law gamma bit selection

The file **psp_vpfe.h** has the **PSP_VPFECCDC_BlackAdj** data structure is used to configure the black adjustment for Raw Input runtime. The params are explained below:

Parameter	Description
clampEn	Enabl/Disable Black level Clamping
blackSampLen	Num Black Sample pixels per line
blackSampLine	Num Black Sample lines
blackPixStart	Start black pixel position
blackGain	Gain to be applied to optical black average
dcSub	Constant DC value to subtract if clampEn is disabled
colPtn	CCDC color pattern
blkCmpR	Black level compensation for R color pixels
blkCmpGR	Black level compensation for GR color pixels
blkCmpGB	Black level compensation for GB color pixels
blkCmpB	Black level compensation for B color pixels

The file **psp_vpfe.h** has the **PSP_VPFE_CCDC_Fpc** data structure is used to configure the black adjustment for Raw Input runtime. The params are explained below:

Parameter	Description
fpcEnable	CCDC Fault Pixel Correction enable/disable
fpcnum	Number of Fault Pixels to be corrected
fpcAddr	Address of the FPC Table
fpcreset	Parameter to reset the FPC error

3.4 Enumerations

This section lists the enumerations available in the PSP module.

enum PSP_VPFEIoctlCommand

Defines	Description
PSP_VPFE_IOCTL_CCDC_VALIDATE_BUFFER	To validate the FrameBuffer and also to get the Correct Frame Buffer size
PSP_VPFE_IOCTL_CCDC_BLACK_ADJUST	For black adjustment when the Input is Raw
PSP_VPFE_IOCTL_CCDC_FPC	For Fault Pixel Correction when the Input is Raw
PSP_VPFE_IOCTL_START_CCDC	To Start CCDC Engine
PSP_VPFE_IOCTL_STOP_CCDC	To Stop CCDC Engine

enum PSP_VPFE_Id

Defines	Description
PSP_VPFE_INVALID_ID	Invalid Id
PSP_VPFE_CCDC	CCDC Channel Id
PSP_VPFE_MAX_INTERFACES_SUPPORTED	Maximum number of Id supported

enum FVID_videoInterface

Defines	Description
FVID_VI_BT656_8BIT	Input is from Video Decoder (YCbCr)
FVID_VI_RAW_10BIT_CS	Input is from MT9T001 sensor

enum PSP_VPFEPack

Defines	Description
PSP_VPFE_PACK8_16BITS_PIXEL	Pack raw data in to 16bits
PSP_VPFE_PACK8_8BITS_PIXEL	Pack raw data into 8 bits

enum PSP_VPFE_Datapol

Defines	Description
PSP_VPFE_DataPol_Normal	Data Polarity :Normal
PSP_VPFE_DataPol_OnesComplement	Data Polarity : One's Complement

enum PSP_VPFE_SyncPolarity

Defines	Description
PSP_VPFE_SyncPol_Positive	Sync Polarity : Positive
PSP_VPFE_SyncPol_Negative	Sync Polarity : Negative

enum PSP_VPFE_SyncDirection

Defines	Description
PSP_VPFE_SyncDir_Input	Sync Direction :Input
PSP_VPFE_SyncDir_Output	Sync Direction :Output

enum PSP_VPFE_BlkComp

Defines	Description
PSP_VPFE_BlkCompEnable	Black Compensation Enable
PSP_VPFE_BlkCompDisable	Black Compensation disable

enum PSP_VPFE_ALaw

Defines	Description
PSP_VPFE_ALaw_Disable	A-Law Disable
PSP_VPFE_ALaw_Enable	A-Law Enable

enum PSP_VPFE_Fpc

Defines	Description
PSP_VPFE_Fpc_Disable	FPC Disable
PSP_VPFE_Fpc_Enable	FPC Enable

enum PSP_VPFE_Err

Defines	Description
PSP_VPFE_Err_Noreset	Do not reset the FPC error
PSP_VPFE_Err_reset	Reset the FPC error

CHAPTER 4

PORTING GUIDE

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

4.1 Porting Description

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

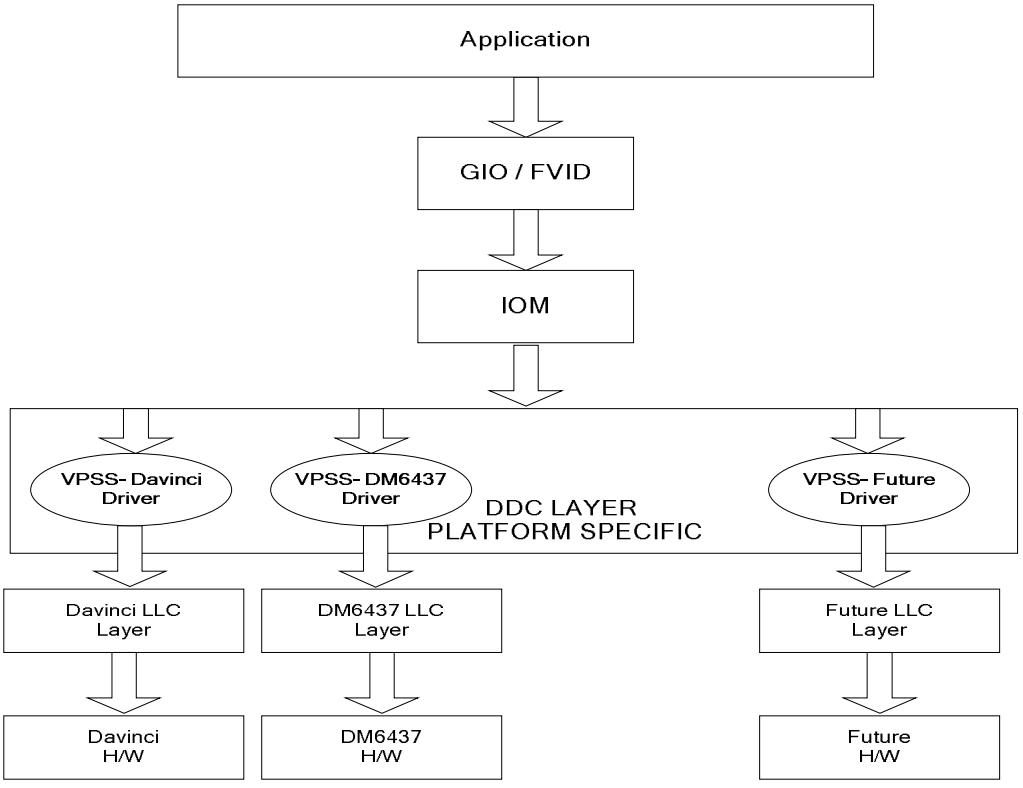


Figure 3 Driver Architecture

There will not be any change required in the GIO Layer, IOM Layer and DDA Layer while porting VPFE 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 Frontend (VPFE) and Backend (VPBE) as whole.

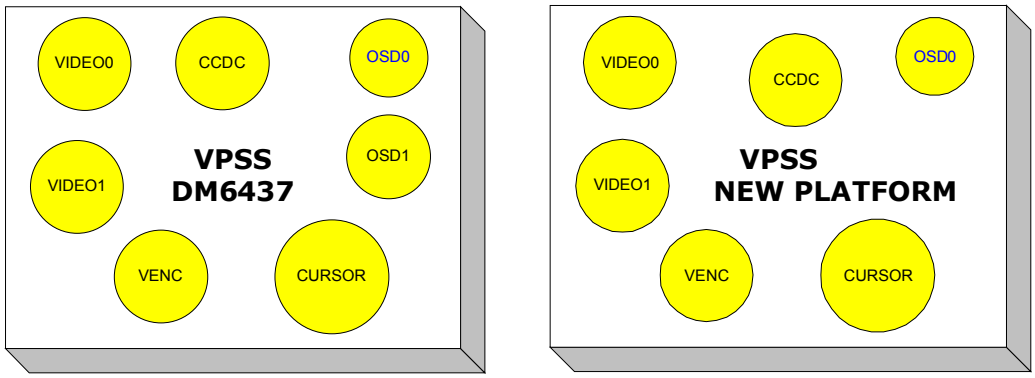


Figure 4 Driver Portability

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 hardware components that are not built inside VPSS module and VPSS has dependency on such peripherals. (For eg: The DM6437 VPSS module doesn't have in buit video decoder).For these scenarios the driver should have informations about certain parameters that it external peripheral needs.

5.1 TVP5146 Video Decoder

The TVP5146 is a single chip external digital video decoder that digitizes and decodes all popular baseband analog video formats into digital component video. The TVP5146 decoder supports the analog-to-digital (A/D) conversion of component RGB and YPbPr signals, as well as the A/D conversion and decoding of NTSC, PAL and SECAM composite and S-video into component YCbCr. (For details on the device specifications see references above). Interfacing the VPSS module front-end with TVP5146 device is required since TVP5146 provides necessary interfaces to the VPSS front-end.

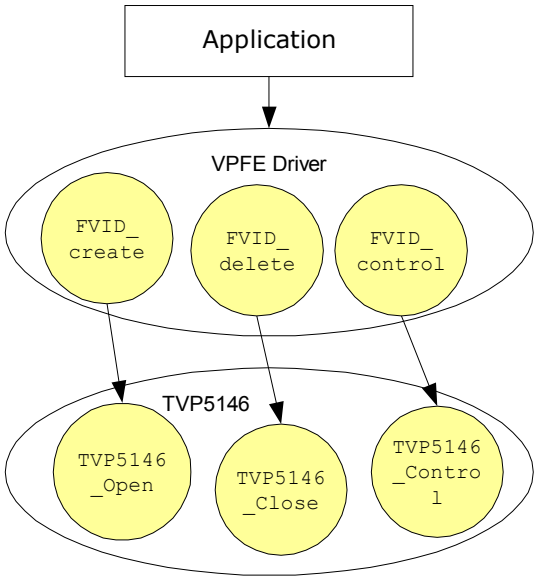


Figure 5 TVP5146 Decoder Function compatible with VPFE driver

TVP5146 video decoder is an independent interface which is called from the VPFE driver.As shown in above when the VPFE driver will call FVID_create it will call TVP5146_Open function which will initialize the TVP5146 and intialize I2C driver for serial communication. To configure the TVP5146, application has to pass the PSP_VPSS_EXT_VIDEO_DECODER_CONFIG Ioctl command by calling FVID_control. This will indirectly call TVP5146_Control function. Once the VPFE driver deletes the channel, it will delete the TVP5146 instance and close the I2C driver as well.

5.1.1. TVP5146 Interface details

The TVP5146 interfaces with the VPFE driver using 3 pointers to the functions as below:

- PSP_VPFE_TVP5146_Open
- PSP_VPFE_TVP5146_Close
- PSP_VPFE_TVP5146_Control

	Function	Function Description
1	PSP_VPFE_TVP5146_Open	❑ To initialize external tvp5146 video decoder that is used by application
2	PSP_VPFE _TVP5146_Close	❑ To do the final house-keeping before the decoder is closed
4	PSP_VPFE _TVP5146_Control	❑ IOCTL to change tvp5146 video decoder parameters runtime.

TVP5146 video decoder peripheral registers are configured using I2C driver. TVP5146 video decoder will act as Slave device.I2C will communicate with TVP5146 using the TVP5146 video decoder slave address (0x5D).Refer to TVP5146 specs for more detail.

PSP_VPFE_TVP5146_Open function is used to initialize the I2C driver. It configures the I2C for further register read and write of TVP5146 video decoder. During open call application plugs-in the decoder functions in the function pointers provided in the front-end (external decoder) object.

During the close call the external decoder close function is called through the function pointer PSP_VPFE_TVP5146_close.

During the IOCTL calls the external decoder control function is called through the function pointer PSP_VPFE_TVP5146_Control.

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

#	Prototype
1	`EVD_Handle (*Open)()`
2	`Int (*Close)(Ptr handle)`

#	Prototype
3	`Int (*Control)(Ptr handle,Uint32 Cmd,Ptr CmdArg)`

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

IOCTL Command	VALUE	PARAMS
PSP_VPSS_EXT_VIDEO_DECODER_CONFIG	0x01	TVP5146_ConfigParams
PSP_VPSS_EXT_VIDEO_DECODER_STATUS	0x02	TVP5146_StatusParams
PSP_VPSS_EXT_VIDEO_DECODER_CONTROL	0x03	TVP5146_ControlParams

To configure the TVP5146 video decoder application needs to create one object of TVP5146_ConfigParams structure.

Application needs to call VPFE Ioctl with PSP_VPSS_EXT_VIDEO_DECODER_CONFIG Ioctl cmd to configure the TVP5146 external video decoder and pass object of TVP5146_ConfigParams structure as cmdArg in the IOCTL.

Below is TVP5146_ConfigParams structure description:-

```
typedef struct  _PSP_VPFE_TVP5146_ConfigParams
{
        Bool                    enable656Sync;
        TVP5146_Format          format;
        TVP5146_Mode            mode;
} PSP_VPFE_TVP5146_ConfigParams;
```

Status parameters of TVP5146 video decoder like fieldRate, lostlock, hlock, vlock etc are elements of PSP_VPFE_TVP5146_StatusParams structure.

To get the status of above mentioned status params runtime, call VPFE IOCTL with PSP_VPSS_EXT_VIDEO_DECODER_STATUS cmd.Pass Object of PSP_VPFE_TVP5146_StatusParams as cmdArg.

Below is PSP_VPFE_TVP5146_StatusParams structure description:-

```
typedef struct _PSP_VPFE_TVP5146_StatusParams
{
        Uint8       fieldRate;
        Uint8       lostLock;
        Uint8       colSubCarrier_lock;
        Uint8       vLock;
        Uint8       hLock;
} PSP_VPFE_TVP5146_StatusParams;
```

Control parameters of TVP5146 video decoder like autogain, contrast, brightness, hue, etc are elements of TVP5146_ControlParams structure.

```
typedef struct _PSP_VPFE_TVP5146_ControlParams
{
        Bool    autoGain;
        Uint8   brightness;
        Uint8   contrast;
        Uint8   saturation;
        Uint8   hue;
} PSP_VPFE_TVP5146_ControlParams;
```

Refer to pspdrivers\inc\ psp_tvp5146_extVidDecoder.h header file for more detail.

5.2 MT9001 Image Sensor

The MT9001 Image sensor is a QXGA-format ½-inchCMOS active-pixel digital image sensor with an active imaging pixel array of 2048H x 1536V.It incorporates sophisticated camera functions on-chip such as windowing; column and row skip mode and snapshot mode. It is a programmable simple two serial wire interface.

The image sensor can be operated in its default mode or programmed by the user for frame size, exposure, gain setting, and other parameters. The default mode outputs a QXGA image at 12 frames per second. An on-chip analog-to-digital converter (ADC) provides 10bits per pixel.

The MT9T001 produces extraordinarily clear, sharp digital pictures, and its ability to capture both continuous video and single frames makes it the perfect choice for a wide range of consumer and industrial applications, including digital still cameras, digital video cameras, and PC cameras.

MT9001 image sensor is an independent interface with the VPFE driver.MT9001 image sensor will be configured, through IOCTL of VPFE driver.

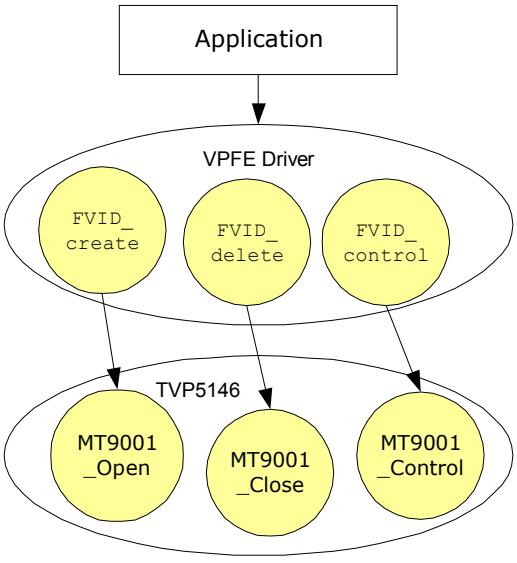


Figure 6 MT9001 Sensor Function compatible with VPFE driver

MT9901 image sensor is an independent interface which is called from the VPFE driver.As shown in above when the VPFE driver will call FVID_create it will call MT9001_Open function which will initialize the MT9001 chip and intialize I2C driverto serial communication. To configure the MT9001, application has to pass the PSP_VPSS_EXT_VIDEO_DECODER_CONFIG Ioctl command by calling FVID_control. This will indirectly call MT9001_Control function. Once the VPFE driver deletes the channel, it will delete the MT9001 instance and close the I2C driver as well.

5.2.1. MT9001 Sensor Interface details

The MT9001 image sensor interfaces with the VPFE driver using 3 pointers to the functions as below:

- MT9001_Open
- MT9001_Close
- MT9001_Control

	Function	Function Description
1	MT9001_Open	❑ To initialize external MT9001 image sensor that is used by application
2	MT9001_Close	❑ To do the final house-keeping before the sensor is closed
4	MT9001_Control	❑ IOCTL to change MT9001 image sensor parameters runtime.

MT9001 image sensor peripheral registers are configured using I2C driver. MT9001 image sensor will act as Slave device.I2C will communicate with MT9001 using the MT9001 video decoder slave address (0x5D).Refer to MT9001 specs for more detail. Before communicating with MT9001, VPFE driver also configures PCF8574A I/O expander with the slave address 0x70.

MT9001_Open function is used to initialize the I2C driver. It configures the I2C for further register read and write of MT9001 image sensor. During open call application plugs-in the sensor functions using the function pointers provided in the front-end (external decoder) object. It also configures PCF8574A I/O expander for MT9001 image sensor.

During the close call the external decoder close function is called through the function pointer MT9001_Close.

During the IOCTL calls the external decoder control function is called through the function pointer MT9001_Control.

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

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

VPFE driver will communicate to image sensor using the IOCTL commands as shown in below figure. Hence external decoder should provide the support to these functions that are required by the drivers as mandatory.

IOCTL Command	VALUE	PARAMS
PSP_VPSS_EXT_VIDEO_DECODER_CONFIG	0x01	MT9001_FormatParams
PSP_VPSS_EXT_VIDEO_DECODER_CONTROL	0x03	MT9001_ControlGainParams
PSP_VPSS_EXT_VIDEO_DECODER_STD_FORMAT	0x04	MT9001_StandardFormat

Application needs to call VPFE Ioctl with PSP_VPSS_EXT_VIDEO_DECODER_CONFIG Ioctl cmd to configure the MT9001 image sensor and pass object of MT9001_FormatParams structure as cmdArg in the IOCTL.

```
typedef struct _MT9001_FormatParams
{
   Uint16 column_size;
   Uint16 row_size;
   Uint16 h_blank;
   Uint16 v_blank;
   Uint16 shutter_width;
   Uint16 row_addr_mode;
   Uint16 col_addr_mode;
   Uint16 black_level;
   Uint16  pixel_clock_ctrl;
   Uint16 row_start;
   Uint16 col_start;
}MT9001_FormatParams;
```

Control parameters of MT9001 image sensor like analog and digital gain of green, blue and red can be change through PSP_VPSS_EXT_VIDEO_DECODER_CONTROL Ioctl. The control params are present in MT9001_ControlGainParams structure.

```
typedef struct _MT9001_ControlGainParams
{
   Uint8   green1_analog_gain;
   Uint8   blue_analog_gain;
   Uint8   red_analog_gain;
   Uint8   green2_analog_gain;
   Uint8   green1_digital_gain;
   Uint8   blue_digital_gain;
   Uint8   red_digital_gain;
```

```
   Uint8   green2_digital_gain;
}MT9001_ControlGainParams;
```

Application needs to call VPFE Ioctl with PSP_VPSS_EXT_VIDEO_DECODER_STD_FORMAT Ioctl cmd to configure the MT9001 image sensor with specified standard format. Supported standard formats are specified by the enum MT9001_StandardFormat.

```
typedef enum _MT9001_STANDARD_FORMAT
{
   MT9T001_MODE_VGA_30FPS,
   MT9T001_MODE_VGA_60FPS,
   MT9T001_MODE_SVGA_30FPS,
   MT9T001_MODE_SVGA_60FPS,
   MT9T001_MODE_XGA_30FPS,
   MT9T001_MODE_480p_30FPS,
   MT9T001_MODE_480p_60FPS,
   MT9T001_MODE_576p_25FPS,
   MT9T001_MODE_576p_50FPS,
   MT9T001_MODE_720p_24FPS,
   MT9T001_MODE_720p_30FPS,
   MT9T001_MODE_1080p_18FPS
} MT9001_StandardFormat;
```

NOTE: It is required to invert the pixel clock of MT9T001 sensor . The image output will be of good quality only when pixel clock of MT9T001 is inverted.

Refer to pspdrivers\system\DM6437\bios\dm6437_evm\src\video\ MT9001_extImageSensor.h header file for more detail.

I2C Driver for External Chip

The i2c driver will be used by VPSS driver for communicating with external chips like TVP5146 and MT9001 image sensor. The various functions of I2C are implemented as interfaces are explained below.

Following are the functions list of I2C driver implemented as interface.

	Function	Function Description
1	i2c_init	❑ To open i2c driver. This is called from tvp5146VideoDecoderInit.
2	i2c_deInit	❑ To do the final house-keeping of I2C driver. This function is called from tvp5146VideoDecoderDeinit.
3	i2c_writeReg	❑ To write on any TVP5146 register through i2c.
4	i2c_readReg	❑ To read any TVP5146 register.

Refer to pspdrivers\system\DM6437\bios\dm6437_evm\src\video\src\psp_i2c_interface.c source 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.

6.1 Writing Applications for VPFE

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

6.1.1. File Inclusion

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

1. **psp_vpfe.h**

   This file contains the interfaces, data types and symbolic definitions that are needed by the application to utilizes the services of VPFE 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

4. **tvp5146_extVidDecoder.h**

   This file contains the interfaces, data types and symbolic definitions that are needed by the application to configure the TVP5146 video decoder. This header files needs to be added at the application only if the input to VPFE module is from video decoder.

5. **MT9001_extImageSensor.h**

   This file contains the interfaces, data types and symbolic definitions that are needed by the application to configure the MT9001 Image sensor. This header files needs to be added at the application only if the input to VPFE module is configured from Image Sensor.

6.1.2. Driver Initialization

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

To have VPFE device driver included in the application, corresponding TCI file have to be included in BIOS TCF i.e. "*dm6437_vpfe0.tci*" must be included in BIOS TCF file of the application. This file can be found in video sample directory.

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

```
bios.UDEV.create("VPFE0");
bios.UDEV.instance("VPFE0").fxnTable = prog.extern("VPFEMD_FXNS");
bios.UDEV.instance("VPFE0").fxnTableType = "IOM_Fxns";
```

Apart from the VPFE 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*.

6.1.3. Dependent Projects/Libraries

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

- VPFE
- Previewer (Optional)
- Video (for external encoders/decoders)
- I2C
- PAL_OS
- SoC specific PAL_SYS

6.1.4. 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

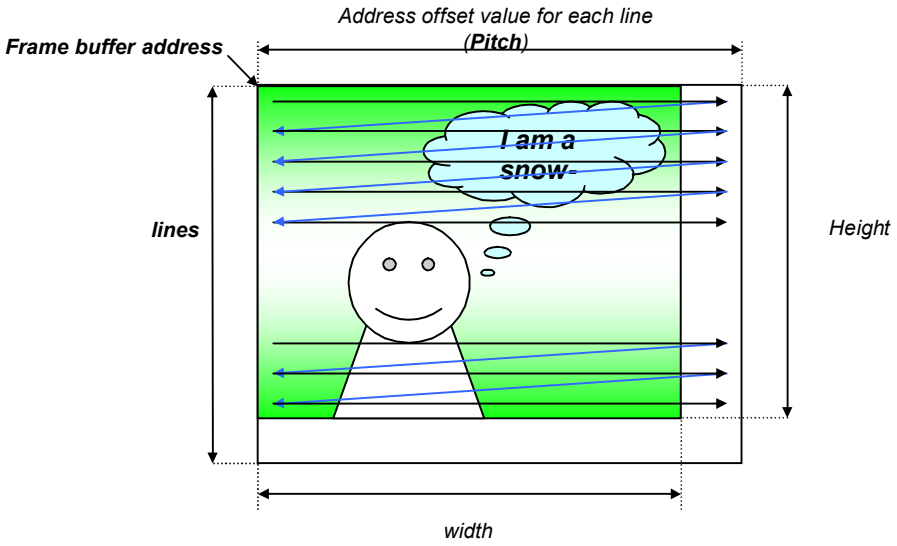
6.1.5. Memory Allocation

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



Bold: *Surface params*
Normal: *Hardware params*

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             CcdcHandle;
FVID_Frame *FBAddr = NULL;

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

6.1.6. 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.

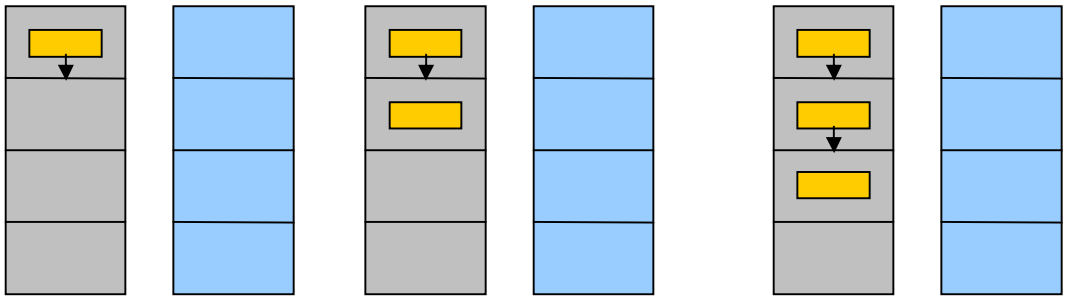


Figure. Operation of QUEUE call.

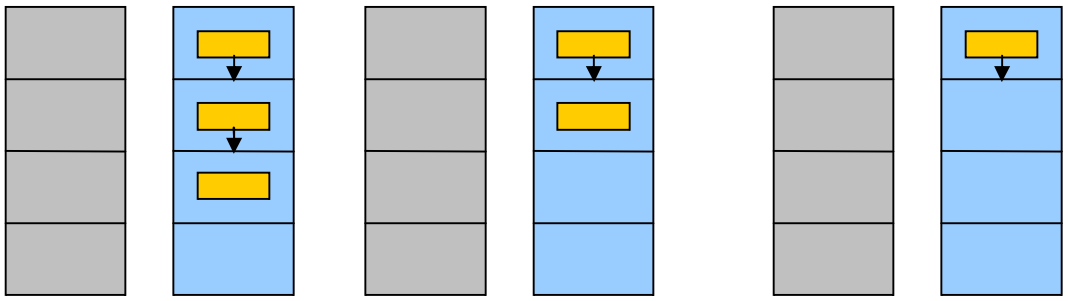


Figure: Operation of DE_QUEUE call.

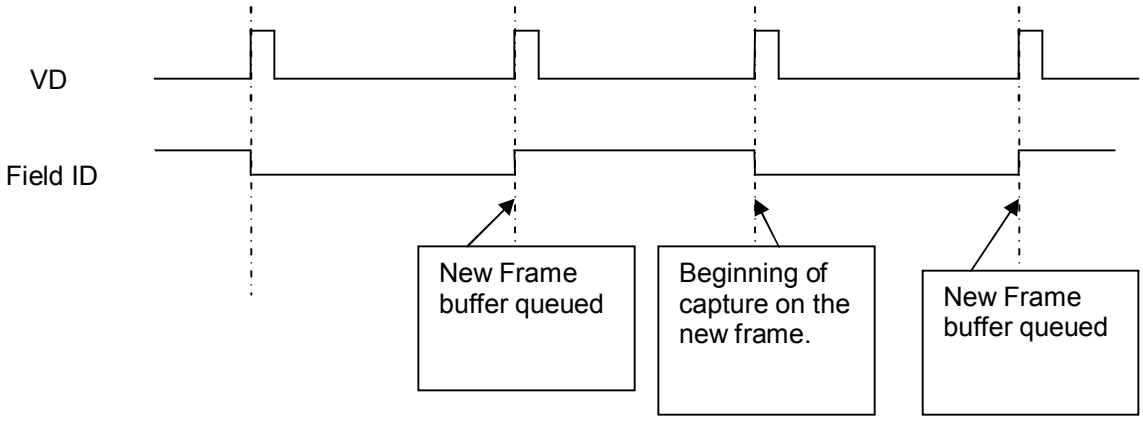
The drivers maintain two circular queues called READY queue and a FREE queue for each channel of display and capture. The READY queue contains the buffer pointers of each of the buffers which are to be worked upon (passed to the driver using the QUEUE call), and the FREE buffer contains all the buffer 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 Dequeue, in a single API call. The Exchange call will avoid the simultaneous call of Queue and Dequeue Application passes 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 with the pointer to the frame buffer that was previously queued in the device driver.

ISR OPERATION

The ISR for VPFE is hooked to VDINT0 interrupt which occurs at every VSYNC. The ISR handles queuing and dequeuing 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.



6.2 The VPFE YUV Sample Application

6.2.1. Introduction

The sample application configures TVP5146 video decoder chip as input to ccdc module. TVP5146 video decoder gives processed output in YUV format. Capture driver (VPFE) captures data in YUV format and gives it to display driver (VPBE) simultaneously in loop back.

6.2.2. Building the Application

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

6.2.3. 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 DVD player or Camera output to Video In connector on the board and display screen (TV) input to Video out connector (DAC B) on the board.

6.2.4. Configuration Parameters

This section describes about how the TVP5146 video decoder chip, VPFE driver and VPBE driver parameters configured for loop back application.

TVP5146 Configuration Parameters

TVP5146 video decoder should be configured when CCDC input is from TVP5156 .The default parameters for Configuring TVP5146 video decoder is described below:

```
static TVP5146_ConfigParams tvp5146Params =
{
        TRUE,                           /* enable656Sync */
        TVP5146_FORMAT_COMPOSITE,       /* format        */
        TVP5146_MODE_AUTO               /* mode          */
};
```

VPFE Driver Configuration Parameters

VPFE driver is configured to capture processed data (YUV) from TVP5146 video decoder. The default parameters for VPFE driver are described below:

```
static PSP_VPFECcdcConfigParams ccdcParams =
{
   FVID_VI_BT656_8BIT,          /*  dataFlow     */
   FVID_FRAME_MODE,             /*  ffMode       */
   480,                         /*  height       */
   720,                         /*  width        */
   (720 *2),                    /*  pitch        */
```

```
    0,                                  /*  horzStartPix     */
    0,                                  /*  vertStartPix     */
    NULL,                               /*  appCallback      */
    {
        PSP_VPFE_TVP5146_Open,          /* extVD Fxn         */
        PSP_VPFE_TVP5146_Close,
        PSP_VPFE_TVP5146_Control,
    },
    0                                   /* segId              */
};
```

VPBE Driver Configuration Parameters

The VPBE driver is used to display a captured image on display screen. The default parameters to configure the VPBE driver are described below:

```
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_PAL_INTERLACED_COMPOSITE /* Display Standard */
};
```

6.3 The VPFE RAW Sample Application

6.3.1. Introduction

The sample application takes input from Image sensor. Image sensor gives raw image as an output. Capture driver (VPFE) captures raw data and gives it to Black & white conversion logic. Black and White logic converts data in black & white. After converting data into black &White it is given to Display driver (VPBE).This application runs in loop back.

6.3.2. Building the Application

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

6.3.3. 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:-Connect Image Sensor card on ADON card provided along with the board. Then insert this add-on card setup on the DM6437 hardware board. Connect cable from DAC B output connector to TV screen to see the image captured.

6.3.4. Configuration Parameters

This section describes how the VPFE driver and VPBE driver parameters configured for raw loop back application to work.

MT9001 Image Sensor

The MT9001 Image sensor is not configured from Software. Image sensor has default values at power on which are taken as input to VPFE driver.

VPFE Driver Configuration Parameters

VPFE driver is configured to capture RAW data from MT9001 Image sensor. The VPFE driver is configured in default values for Raw Capture is described below:

```
static PSP_VPFECcdcConfigParams ccdcParams =
{
   FVID_VI_RAW_10BIT_CS,                  /*  dataFlow          */
   FVID_FIELD_MODE,                       /*   ffMode           */
   480,                                   /*  height            */
   720,                                   /*  width             */
   (720 *2),                              /*  pitch             */
    0,                                    /*  horzStartPix      */
     0,                                   /*  vertStartPix      */
    NULL,                                 /*  appCallback       */
    {
            NULL,                         /*  extVD Fxn         */
            NULL,
            NULL,
     },
     0,                                   /*segId               */
     {
         PSP_VPFE_BITS16,                 /* dataSize           */
         PSP_VPFE_PACK8_16BITS_PIXEL,     /* pack8              */
         PSP_VPFE_DataPol_Normal,         /* dataPol            */
         PSP_VPFE_SyncPol_Positive,       /* VDSyncPol          */
         PSP_VPFE_SyncPol_Positive,       /* HDSyncPol          */
         PSP_VPFE_SyncDir_Input,          /* HDVDMaster         */
         50,                              /* HDSyncWidth        */
         4,                               /* VDSyncWidth        */
         800,                             /* numPxlPerLine      */
         1000,                            /* numLinPerFld       */
         PSP_VPFE_ALaw_Disable,           /* ALawEnable         */
         PSP_VPFE_ALaw_bits15_6,          /* ALaw_Width         */
    }
};
```

VPBE Driver Configuration Parameters

The VPBE driver is used to display a captured image on display screen. Here OSD plane is also configured along with Video plane. The default parameters to configure the VPBE driver are described below:

```
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_VPBEOsdConfigParams      osd0Params =
{
        FVID_FRAME_MODE,                    /* ffmode        */
        FVID_BPP_BITS16,                    /* bitsPerPixel  */
        FVID_RGB565_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   */
        PSP_VPBE_BLEND7                     /* blending      */
};

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

6.4 The VPFE PREVIEWER on-the-fly Sample Application

6.4.1. Introduction

The sample application takes input from Image sensor. Image sensor gives raw image as an output. Capture driver (VPFE) captures raw data and gives it to Previewer module. Previewer converts the raw data (Bayer Pattern) in to processed data (YUV). Processed data is given to Display driver (VPBE).This application runs in loop back.

Constraints: The previewer driver must be opened before the VPFE driver. Previewer should be configured as CCDC input for on-the-fly mode. VPFE driver checks and decide the path based on Previewer input mode. If the previewer input is CCDC then VPFE driver is configured in on-the-fly mode.

6.4.2. Building the Application

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

Note: To enable A-Law Feature in Ccdc and Inverse A-Law in Previewer ensure that FPGA available on MT9T001 sensor card in programmed for 10-bit data. If FPGA is not programmed for 10-bit then faded image will be observed.

6.4.3. 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:-Connect Image Sensor card on ADON card provided along with the board. Then insert this adon card setup on the DM6437 hardware board. Connect cable from DAC B output connector to TV screen to see the image captured.

6.4.4. Configuration Parameters

This section describes how the VPFE, Previewer and VPBE driver parameters configured for raw loop back application to work.

PREVIEWER Driver Configuration Parameters

To configure the Previewer in on-the-fly mode pass the CCDC input at time of FVID_create. Once the Previewer configures the input as CCDC, Previewer will configure its default parameters using set params Ioctl.

```
Refer to example code to open the Previwer driver for on-the-fly
functionality.
GIO_Handle                    PrevHandle;
PSP_previewerChannelCreateMode  prevChannel;
/* Create Previewer Channel */
prevChannel.chanSource.source = PSP_PREVIEWER_CHANNEL_CCDC;
prevChannel.segId = 0;
PrevHandle = GIO_create("/previewer", IOM_INOUT, NULL,
                        &prevChannel,&gioAttrs);
```

```
if (NULL == PrevHandle)
{
  printf("VPSS  :Previewer Create........Failed !\r\n");
  return;
 }
```

Appendix A

The example code of loopback which capture the Image from DVD player and output the captured image to display driver is described.The input to VPFE driver is Processed data (YUV) from video decoder and output is displayed on display Screen (TV).There is one header file yuv_imageCbCr.h provided in the package which stores array of values which represents the shrek image.

```
#include <std.h>
#include <gio.h>
#include <log.h>

#include "fvid.h"
#include "psp_vpfe.h"
#include "psp_vpbe.h"
#include "psp_tvp5146_extVidDecoder.h"
#include "psputils.h"

#define VPSS_DBG            PSP_DEBUG
#define NO_OF_BUFFERS       (4u)
/*Global Variable Defined */
static FVID_Frame *CcdcallocFB[NO_OF_BUFFERS]={NULL};
static FVID_Frame *VidallocFB[NO_OF_BUFFERS] ={NULL};

static FVID_Handle   CcdcHandle;
static FVID_Handle   Vid0Handle;
static FVID_Handle   VencHandle;

static PSP_VPFE_TVP5146_ConfigParams tvp5146Params =
{
    TRUE,                                 /* enable656Sync */
    PSP_VPFE_TVP5146_FORMAT_COMPOSITE, /* format        */
    PSP_VPFE_TVP5146_MODE_AUTO         /* mode          */
};

static PSP_VPFECcdcConfigParams ccdcParams =
{
    FVID_VI_BT656_8BIT,                 /* dataFlow     */
    FVID_FRAME_MODE,                    /* ffMode       */
    480,                                /* height       */
    720,                                /* width        */
    (720 *2),                           /* pitch        */
    0,                                  /* horzStartPix */
    0,                                  /* vertStartPix */
    NULL,                               /* appCallback  */
    {
        PSP_VPFE_TVP5146_Open,         /* extVD Fxn    */
        PSP_VPFE_TVP5146_Close,
        PSP_VPFE_TVP5146_Control,
```

```
    },
    0                                   /*segId         */
};
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 */
};

static void vpss_main()
{
    PSP_VPBEChannelParams       beinitParams;
    PSP_VPFEChannelParams       feinitParams;
    GIO_Attrs                   gioAttrs     = GIO_ATTRS;
    FVID_Frame       *FBAddr      = NULL;
    Uint32  i = 0;
    Uint32 NumOfIterations  = 2000;

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

    /**
     * Create Ccdc Channel
    **/
    feinitParams.id        = PSP_VPFE_CCDC;
    feinitParams.params    = (PSP_VPFECcdcConfigParams*) &ccdcParams;
    CcdcHandle             = FVID_create ("/VPFE0",IOM_INOUT,NULL,
                                          &feinitParams,&gioAttrs);
    if (NULL == CcdcHandle)
    {
        VPSS_DBG ("VPSS  :CCDC Create.......FAILED \r\n");
        VPSS_DBG ("VPSS  :End of VPSS Loopback Application\r\n");
        return;
    }
```

```
   /**
    * Configure the TVP5146 Video Decoder
   **/
if (IOM_COMPLETED != FVID_control(CcdcHandle,
                                  VPFE_ExtVD_BASE+
                                  PSP_VPSS_EXT_VIDEO_DECODER_CONFIG,
                                  &tvp5146Params))
{
   VPSS_DBG ("VPSS  :Error in Configuring Video Decoder  \r\n");
 }
 else
 {
    if (IOM_COMPLETED == FVID_allocBuffer(CcdcHandle,&CcdcallocFB[0]))
    {
      if (IOM_COMPLETED != FVID_queue(CcdcHandle,CcdcallocFB[0]))
      {
        VPSS_DBG ("VPSS  :CCdC Queuing.......FAILED \r\n");
      }
    }
  }
   /**
    * 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))
       {
           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;
    }
    VPSS_DBG("VPSS  :VPSS Loopback Started \r\n");
    FVID_allocBuffer(CcdcHandle, &CcdcallocFB[1]);
    FBAddr    = CcdcallocFB[1];
    for(i = 0; i < NumOfIterations; i++)
    {
        if(IOM_COMPLETED != FVID_exchange(CcdcHandle,&FBAddr))
        {
                VPSS_DBG("VPSS  :CCDC Exchange.......FAILED \r\n");
        }
        if(IOM_COMPLETED != FVID_exchange(Vid0Handle,&FBAddr))
        {
            VPSS_DBG("VPSS  :Video -0 Exchange.......FAILED \r\n");
        }
    }
    /**
     *  Free Memory Buffers
    **/
    for(i=0;i<NO_OF_BUFFERS;i++)
    {
        FVID_freeBuffer(CcdcHandle,CcdcallocFB[i]);
    }
    FVID_freeBuffer (Vid0Handle,VidallocFB);

    /**
     *  Delete Channels
    **/
    FVID_delete(CcdcHandle);
    FVID_delete(Vid0Handle);

    VPSS_DBG("VPSS: Loopback Application Ended \r\n");
}

void start_vpss_test()
{
    vpss_main();
}
```