Texas Instruments

VPFE Device Driver

Architecture Document

November 2007

Platform Software Group

Document Version 1.55

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

About This Document

This document discusses the TI device driver architecture for VPFE Device Driver for DM6437 SOCs. The target audience includes device driver developers from TI as well as consumers of the driver.

Although this document makes explicit reference to a typical DSP/BIOS serial class device driver as an example, the general aspects of a device driver design presented here, are equally applicable to other class of devices and OS.

Trademarks

Related Documents

- VPSS Driver Documentation
- SPRU977
- DSP/BIOS Driver Guide

Notations

For normal document information, Helvetica Font is used.

`Courier New Font is used in places where the function and its prototype are mentioned.`

Terms and Abbreviations

Term	Description
API	Application Programmer's Interface
CSL	TI Chip Support Library – primitive h/w abstraction
DDC	TI terminology for portion of device driver that is abstracted of any given OS
DM6437	TI's digital multi-media processor with C64+ core
IOM	TI terminology for portion of device driver that is specific to target OS. This constitutes "adaptation" of the generic DDC to identified target OS.
IP	Intellectual Property
ISR	Interrupt Service Routine
LLC	Low level controller
OS	Operating System
PALOS	Platform abstraction layer for operating system
SOC	System on chip
VPBE	Video processing back end
VPFE	Video processing front end
VPSS	Video processing sub-system

Revision History

Date	Author	Comments	Version
September 7, 2006	Maulik Desai	Created the document	1.0
November 30, 2006	Maulik Desai	Modified the document for the release 0.3.0	1.1
December 1, 2006	Maulik Desai	Client's review comments closed	1.2
February 3, 2007	Maulik Desai	CCS version modified	1.3

June 22, 2007	Anuj Aggarwal	BIOS version modified	1.4
November 29, 2007	Sivaraj R	PSP merge package changes - directory structure changes, FVID_allocBuffer and FVID_freeBuffer functions are implemented as GIO control commands	1.5

Table of Contents

List Of Figures

1 System Context

The VPFE device driver architecture presented in this document is situated in the context of DM6437 SOCs targeted for various Video Applications.

1.1 Hardware

The VPFE module a part of VPSS module used in DM6437 is in-built in SOC core. Figure 1 below shows an overview of the VPFE module.

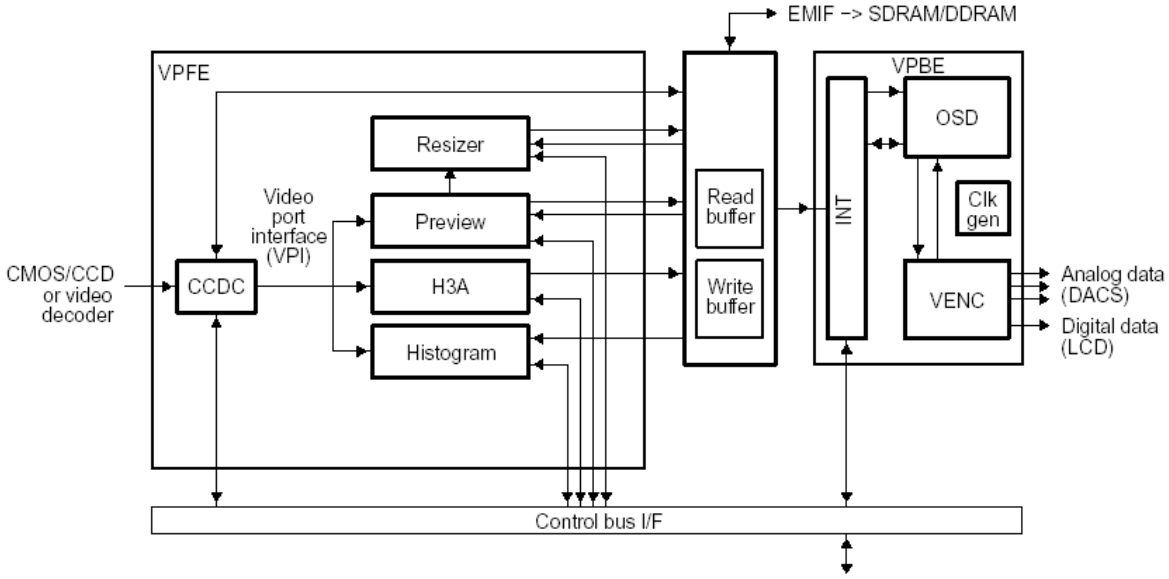

Figure 1 VPSS Block Diagram

1.2 Software

The document provides an overall understanding of the TI VPFE device driver architecture with GIO architecture as a reference framework in the context of DSP/BIOS.

1.2.1 Operating Environment and dependencies

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

1.2.2 System Architechture

The block diagram below shows the overall system architecture.

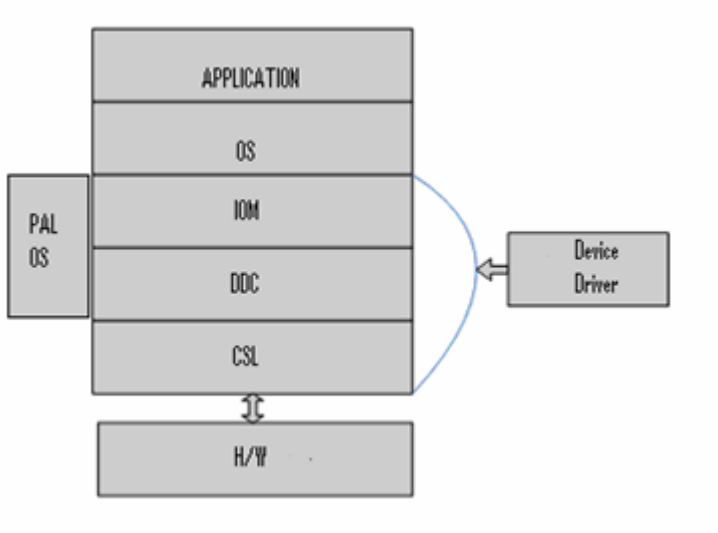

Figure 2 System Architecture

The Application would invoke the driver routines through the GIO Calls APIs. This layer is called as adaptation layers. Device drivers are accessed by the applications for performing I/O through the BIOS through the above-mentioned layers.

IOM is the component that exposes the driver Core to the OS. DDC is the driver core, which actually performs the device specific operations. LLC is the layer that has a direct access to the hardware. IOM, DDC and LLC are discussed in detail in the later sections. The rest of the document elaborates on the architecture of the Device driver by TI.

1.3 Design Philosophy

Please refer section 1.3 of DM6437_BIOS_PSP_User_Guide.pdf for Design Philosophy.

2 VPFE Driver Software Architecture

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

2.1 Static View

2.1.1 Functional Decomposition

The device driver is partitioned into distinct sub-components, consistent with the roles and responsibilities already discussed in section 1.3. In the following sub-sections, each of these functional sub-components of the device driver is further elaborated.

Please refer section 1.2.1 of DM6437_BIOS_PSP_User_Guide.pdf for Diagrammatic explanation.

The central portion (IOM-DDC-LLC) shown constitutes the mainline VPFE driver component. The surrounding module PAL-OS constitute the supporting system components that facilitates the interfaces between the OS and the above mentioned driver components. These modules do not specifically deal with VPFE but assist the driver by providing OS abstraction.

2.1.2 H/W Device Specific Layer (LLC)

Please refer section 1.2.2 of DM6437_BIOS_PSP_User_Guide.pdf for Hardware Layer explanation.

2.1.3 Device Driver Core functionality (DDC)

Please refer section 1.2.3 of DM6437_BIOS_PSP_User_Guide.pdf for DDC Layer explanation.

2.1.4 OS Specific Device Driver Adaptation (IOM)

Please refer section 1.2.4 of DM6437_BIOS_PSP_User_Guide.pdf for IOM Layer explanation.

2.1.5 Platform Abstraction Layer for OS services (PALOS)

Please refer section 1.2.5 of DM6437_BIOS_PSP_User_Guide.pdf for IOM Layer explanation.

2.1.6 Component Interfaces

In the following subsections, the interfaces implemented by each of the sub-component are specified. Refer to VPFE Device Driver API reference documentation for complete details on APIs.

- **IOM Interface**

  The IOM constitutes the Device Driver manifest to Application. This adapts the Driver Core (DDC) to DSP/BIOS.

  The user of device driver will only need bother about the IOM interface, especially the upper-edge services exposed to the Application/OS. All other interfaces discussed later in this document are more of interest to people developing/maintaining the device driver.

  The IOM can be modified to re-target driver and/or customize to specific Apps framework by doctoring the upper-edge services.

  The VPFE_mdBindDev () populates static settings in driver object creates the necessary interrupt handler, attaches the Driver Core interfaces.

- **DDC Interface**

  DDC forms the heart of Device driver. It models driver state machine and implements the data movement. The Device DDC is inherently Asynchronous and contains the crux of the ISR functionality – VPFEIsr (). However, the ISR would be registered and dispatched thru' DDA.

  The VPFE DDC maintains two frame buffers queues with a defined size as MAX_FRAME_BUFFERS. One of these queues (ACTIVE) queue will be used to queue the frame buffers which the application requests to 'Queue'. The ACTIVE queue queues the frame buffers that will be used by the driver for capture of a frame. The other (FREE) queue will be used to queue the frame buffers that are no longer needed by the driver, are free and can be 'De-Queued' by the application.

  The VPFE DDC implements a method to create itself wherein it exchanges the interfaces contract with the DDA above. Once the VPFE driver instance is created, opens handle for each VPFE modules through DDC_VPFEOpenHandle.Once the device has been opened and application wants to capture the frame it will issue a "QUEUE" command and that buffer gets queued in the ACITVE buffer queue using DDC_VPFEAddQueue. In the ISR context thru DDC_VPFEIsr the driver checks for any new frame buffer in the Active queue and if there is any then driver puts the last displayed frame buffer in the FREE queue. It deletes entry of frame buffer from the ACITVE queue using DDC_VPFEDelQueue.

Upon completion of use, driver can be gracefully closed and removed from system by following sequence of calls DDC_VPFEClose () and DDC_VPFEDelete ().

- **LLC Layer Interface**

  The view graph below depicts the services implemented by LLC layer is leveraged in DDC implementation. It should be noted that LLC never calls DDC functions. It's always the other way round.

  LLC does NOT allocate memory dynamically and do NOT use OS services.

- **PALOS Interface**

  PALOS abstracts DSP/BIOS services to the Driver. PALOS functions are state-less on their own and run in context of calling thread. However, when underlying OS function is called, OS may switch context as appropriate, to realize the requested function.

  VPFE driver uses Protect service abstraction. The DDC is always the caller & PALOS the called one, the PALOS will never call a DDC function.

  The VPFE driver release will include an implementation of necessary OS abstraction services for DSP/BIOS environment, if driver is ported to a different platform or integrated with a custom framework, these functions would need to be ported so as to reuse the DDC as-is.

  For further details on PALOS services, refer to API Reference documentation.

- **Frame Buffer Memory Management**

  Application allocates the Frame buffer and has the entire control over these. Driver shall not be allocating the Frame buffers. Driver however shall be validating the frame buffers passed by the application prior to queuing. This is due to some hardware restrictions on the VPFE module.

2.2 Dynamic View

2.2.1 The Execution Threads

The VPFE device driver works only in the Interrupt mode. The VPFE device driver operation involves following execution threads:

2.2.2 Driver Creation

The following section elaborates on the detailed information for the call sequence.

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

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

The mdBindDev () performs register overlaying of the device driver. It registers the interrupt handler of the driver. It attaches the DDC functions for use later during actual initialization of each device instance.

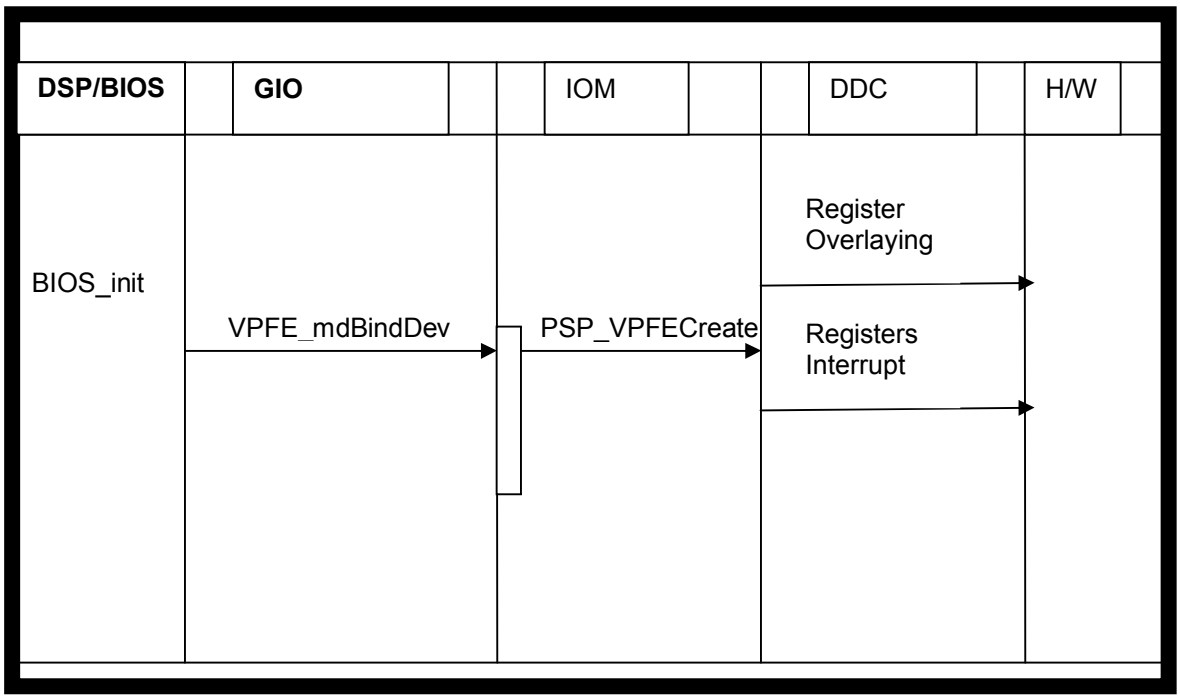

Figure 3 Driver Creation Overview

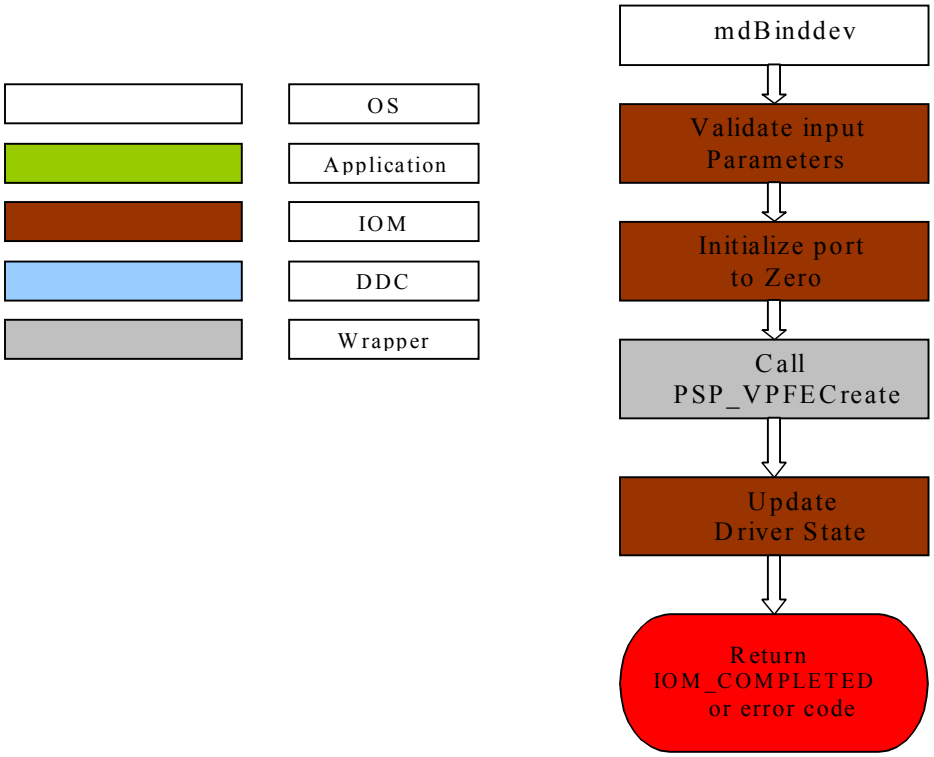

Figure 4 Driver Creation Detailed Flow Diagram – 1

C1	Function	mdBindDev()
	Function Prototype	`Int VPFE_mdBindDev (Ptr *devp, Int devid, Ptr devParams)`
	Input Parameters	devid – number of device instances devParams – would be the H/W configuration information pointer variable. devp – void pointer to be updated once instance is created
	Output Parameters	Int returning the IOM Status
	Description	This function would be implemented at the IOM layer. This function would create a DDC instance of the driver and initialize the driver as well.
	Preconditions	1. The Driver supports only one instance so devid should not be more than 1. 2. The driver should be opening for the first time at the OS initialization time.
	Design	Logic in steps. 1. Check the number of instance 2. set the initial global values to zero 3. call PSP_VPFECreate function 4. Update the State

2.2.3 Driver Open Channel

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

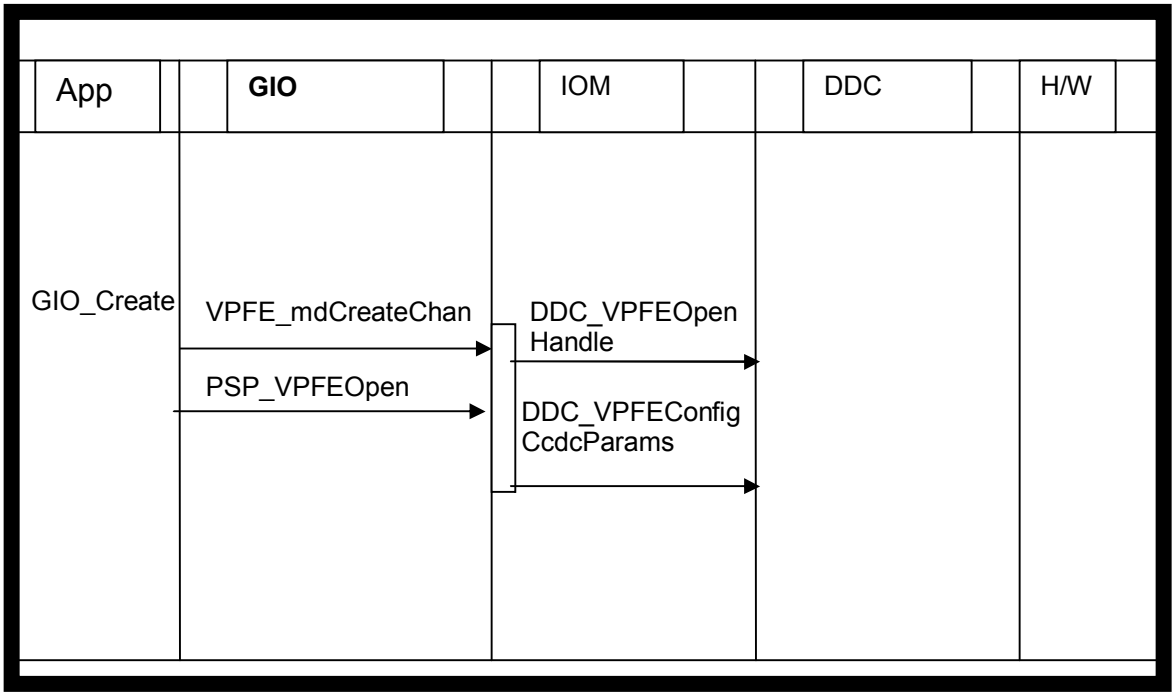

Figure 5 Channel Open Overview

VPFE_mdCreateChan

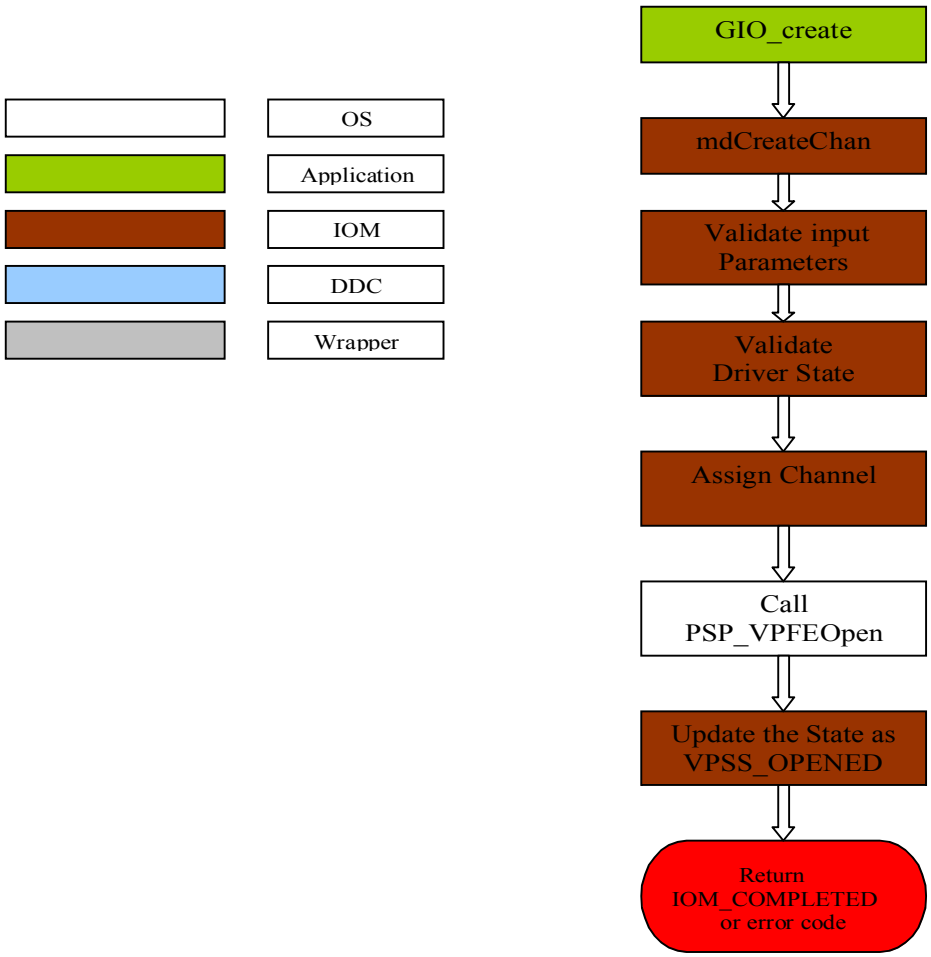

Figure 6 Driver Open Detailed Flow Diagram – 1

O1	Function	`mdCreateChan`
	Function Prototype	`Int VPFE_mdCreateChan (Ptr *chanp, Ptr devp, String name, Int mode, Ptr chanParams, IOM_TiomCallback cbFxn, Ptr cbArg)`
	Input Parameters	chanp – void pointer to be updated after the channel has been initialized devp – pointer to the port structure name – name of the driver mode – mode of the driver i.e. INPUT or OUTPUT chanParams – additional parameters needed to initialize the channel cbFxn – callback function to the GIO cbArg – callback arguments
	Output Parameters	Int according to the IOM errors
	Description	This function declares the driver is ready for any IO transactions.
	Preconditions	The DDC channel need to be initialized and it should be closed before opening
	Design	Logic in steps 1. Check for the valid mode of operation of the channel 2. Check that the channel is not in use. 3. Assign the channel. 4. Call PSP_VPFEOpen function. 5. Update the port state and put channel as in use.

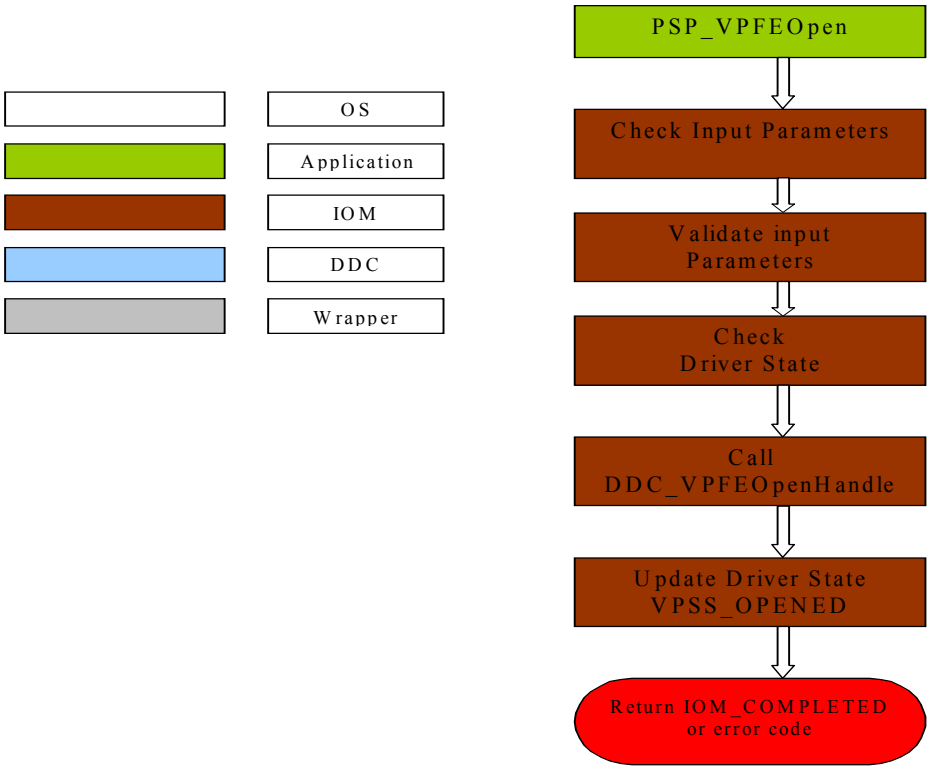

Figure 7 Driver Open Detailed Flow Diagram – 2

O1	Function	`PSP_VPFEOpen`
	Function Prototype	`PSP_Handle PSP_VPFEOpen (Uint32 instanceId, PSP_VPFEChannelParams * chanParams)`
	Input Parameters	InstanceId – Instance id of the Device ChanParams – ConfigParams of the Channel
	Output Parameters	Int according to the PSP errors
	Description	This function creates a channel to carry out the transaction.
	Preconditions	Driver must be in Created state
	Design	Logic in steps 1. Check the Input Parameters 2. Check the State of the Driver 6. Call DDC_VPFEOpenHandle function. 7. Update the state of the Channel as VPSS_OPENED

DDC_VPFEOpen Handle

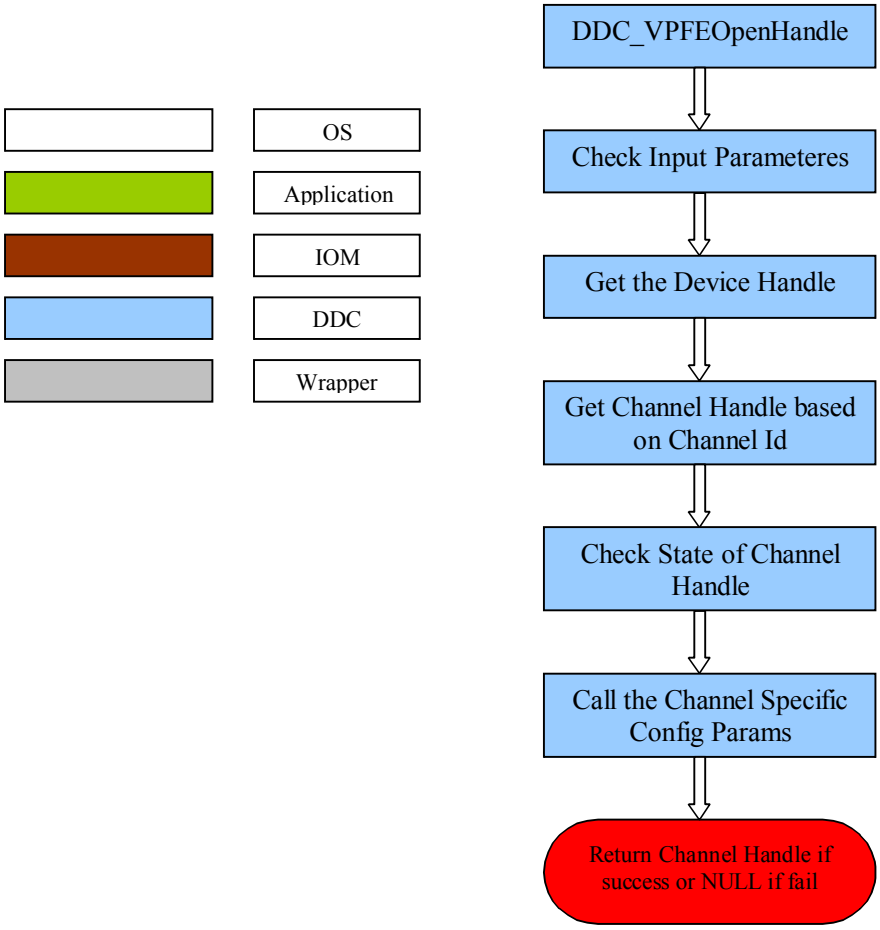

Figure 8 Driver Open Detailed Flow Diagram – 3

O1	Function	`DDC_VPFEOpenHandle`
	Function Prototype	`Ptr DDC_VPFEOpenHandle (PSP_VPFE_Id id, Ptr params)`
	Input Parameters	Id –Channel Id Params- ConfigParams of the Channel
	Output Parameters	Returns Channel Handle or NULL
	Description	This Function initialize the VPFE modules and returns Channel handle
	Preconditions	Driver must be in Created state
	Design	Logic in steps 1. Check the Input Parameters 2. Get the Device Handle 3. Get the Channel handle based on Id. 4. Check the State of Driver. 5. Call VPFE module specific config function based on Id. 6. Update the state asDDC_VPFE_OPENED 7. Return Channel Handle if success otherwise NULL.

2.2.4 IO Control

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

Following is the flow diagram for the above functionality.

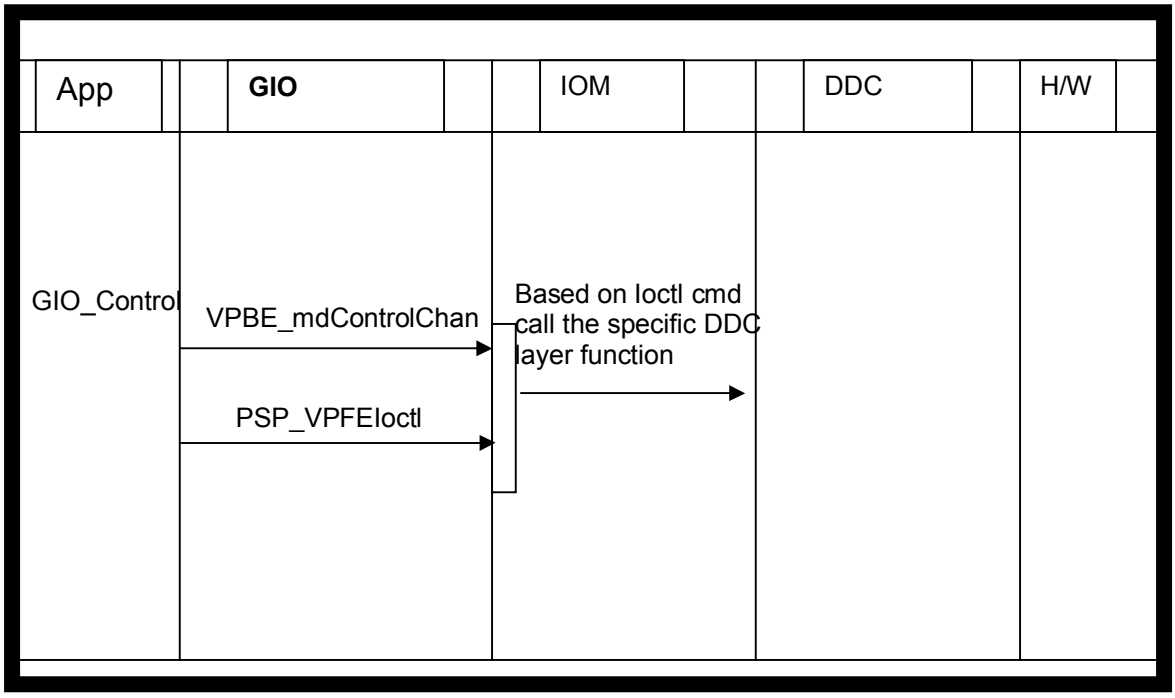

Figure 9 Driver IOCTL Overview

It should be observed that the user's IOCTL request completes in the context of calling thread i.e., application thread of control. Refer Appendix A1. for the corresponding IOCTL codes.

VPFE_mdControlChan

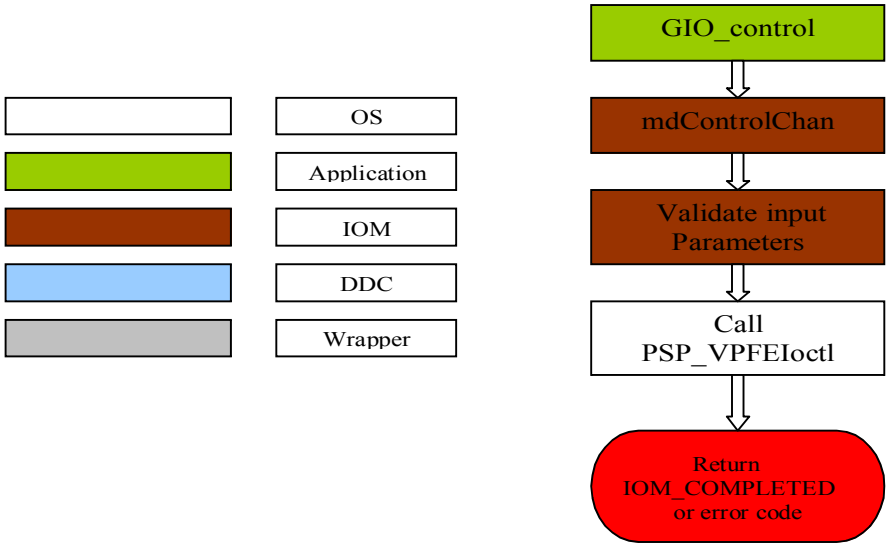

Figure 10 Driver Control Detailed Flow Diagram – 1

I1	Function	mdControlChan
	Function Prototype	`Int VPFE_mdControlChan(Ptr chanp, Uns cmd, Ptr arg)`
	Input Parameters	chanp – pointer to the channel object. cmd – control command to be executed arg – argument needed to execute the command
	Output Parameters	Int according to the IOM error codes
	Description	This function would perform various control actions runtime commands on the device driver. For example to validate the Frame buffer pass PSP_VPFE_IOCTL_CCDC_VALIDATE_BUFFER as cmd. For more information Refer to Appendix A.4
	Preconditions	The device driver state should be opened.
	Design	Logic in steps 1. Check for valid IOCTL command. 2. Call a function to Execute the IOCTL command For Example a. To Validate the Frame Buffer pass PSP_VPFE_IOCTL_CCDC_VALIDATE_BUFFER as cmd. Pass command argument the configParams structure.

2.2.4.1 IO Access

The Application will access VPFE driver IOM API VPFE_mdSubmitChan through interface functions from DSP/BIOS. These functions are registered on the GIO Layer during the driver initialization.

The DDC will maintain two frame buffers queues with a defined size of the buffers. Once the buffer is prepared, application will issue a “QUEUE” call to the driver and that buffer is queued in the Active Queue. At some point in time, when the H/W interrupt is asserted, the ISR checks for any new frame buffer in the ACTIVE buffer and if there is any driver puts the last captured buffer in the FREE queue and updates the address of the new queued frame buffer on the DMA Address.

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

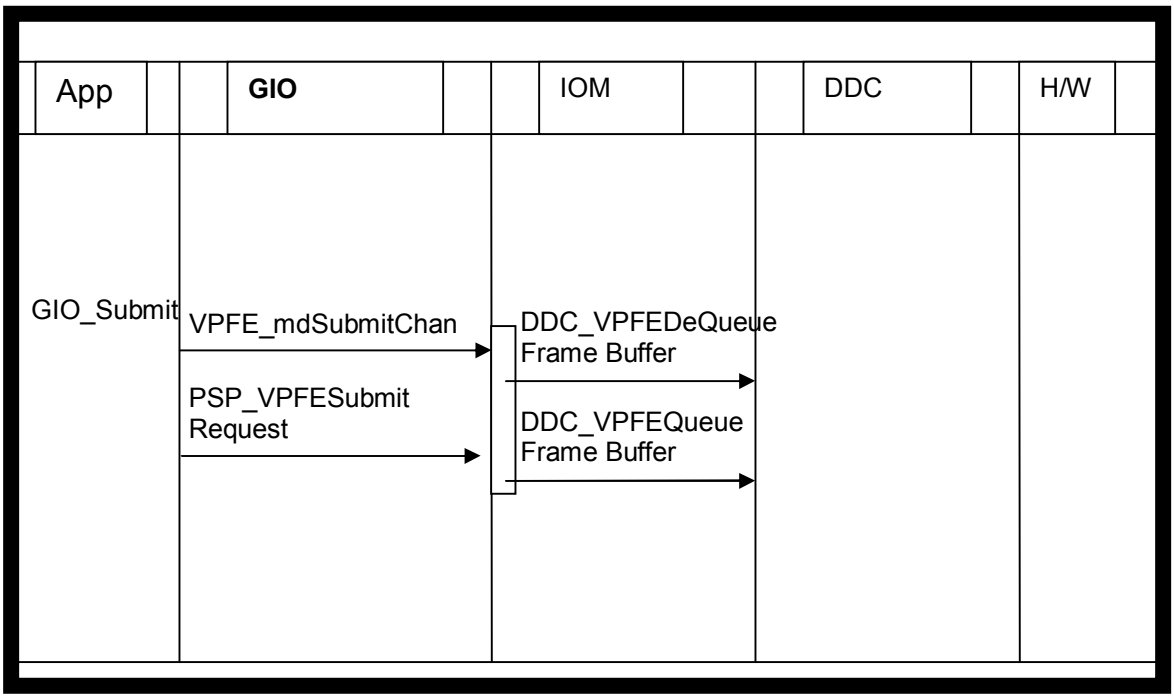

Figure 11 Driver Submit Overview

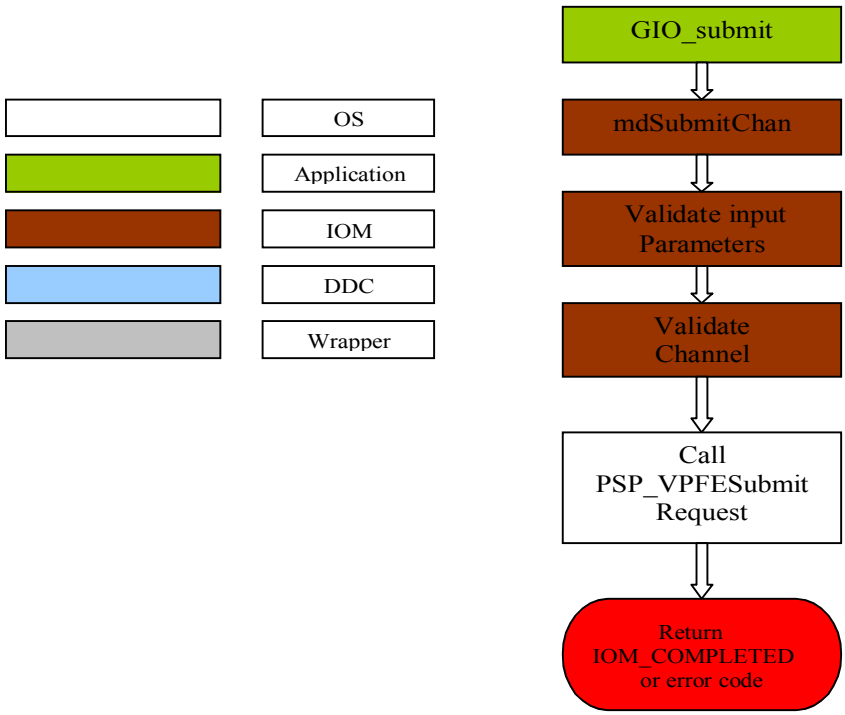

Figure 12 Driver Submit Detailed Flow Diagram – 1

R/W1	Function	mdSubmitChan
	Function Prototype	`static Int VPFE_mdSubmitChan(Ptr chanp, IOM_Packet *packet);`
	Input Parameters	chanp – pointer to chan object packet – the IOM Packet passed by the user
	Output Parameters	Int according to the IOM error codes
	Description	This function would result in the call to the various other APIs depending upon the command issued the packet.
	Preconditions	The driver should be opened
	Design	Logic in steps 1. Depending upon the command issued by User the control goes accordingly a. QUEUE Command When queue command is issued the driver will the Queue the Application frame buffer in the ACTIVE Queue. b. DE-QUEUE Command When the De-queue command is issued application will give the Frame Buffer address from the FREE Queue.

PSP_VPFESubmitRequest

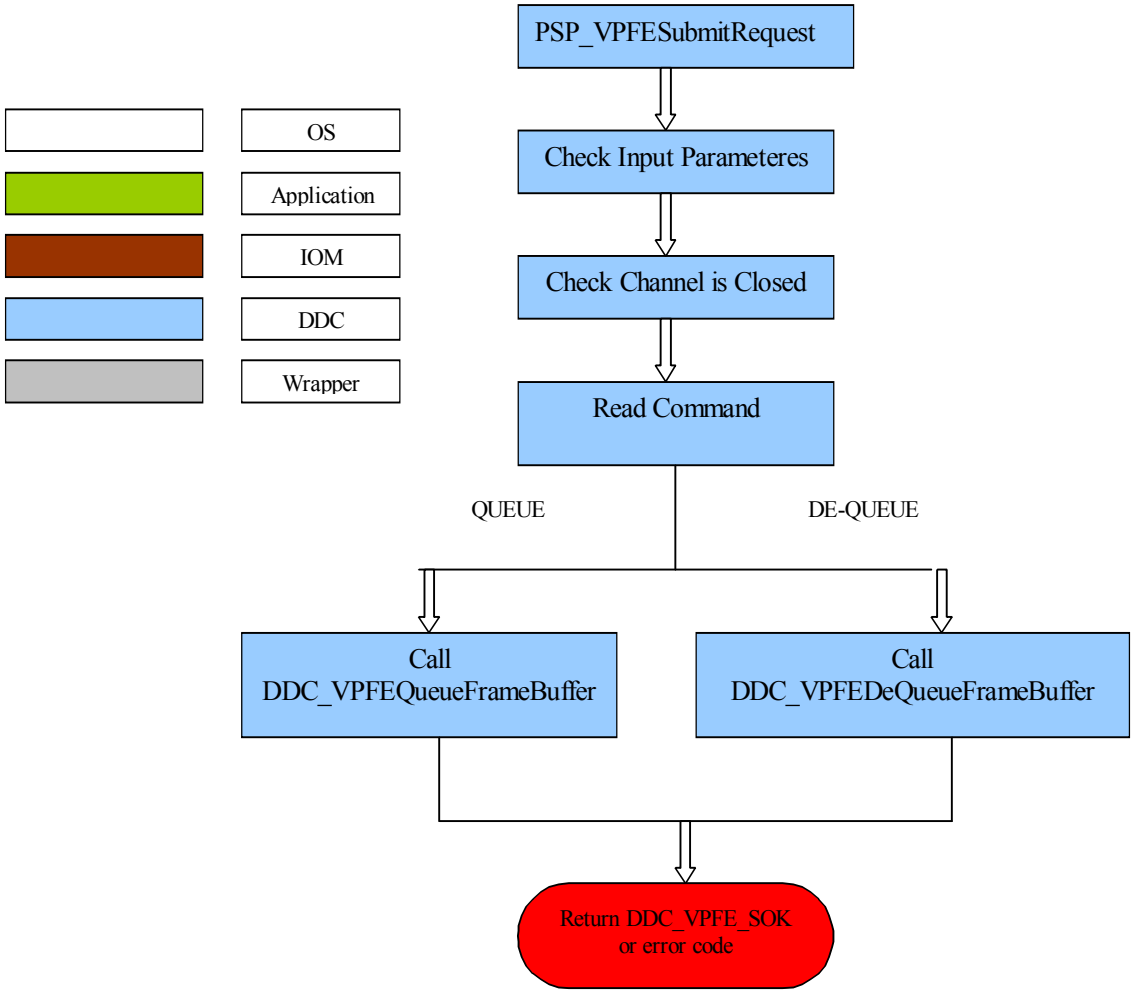

Figure 13 Driver Submit Detailed Flow Diagram – 2

O1	Function	`PSP_VPFESubmitRequest`
	Function Prototype	`PSP_Result PSP_VPFESubmitRequest (PSP_Handle handle, PSP_VPSSSubmitCommand cmd, Ptr cmdArg)`
	Input Parameters	Handle – Handle of the Channel Cmd – Submit command to be passed
	Output Parameters	Int according to the PSP errors
	Description	It acts as wrapper, which provides the information as per command.
	Preconditions	Driver must be in opened state and channel handle should not be NULL.
	Design	Logic in steps 1. Check the Input Parameters 2. Check Channel is Closed or not. 3. Read the submit command 4. For queue –call DDC_VPFEQueueFrameBuffer For Dequeue –call DDC_VPFEDeQueueFrameBuffer function

Interrupts

The Front end interrupt is triggered when the configured number of lines is captured by the FE into the Frame buffer specified in the FE capture DMA Address.

2.2.5 Driver Delete Chan

The application invokes the mdDeleteDev () function to close the channel of the VPFE device.

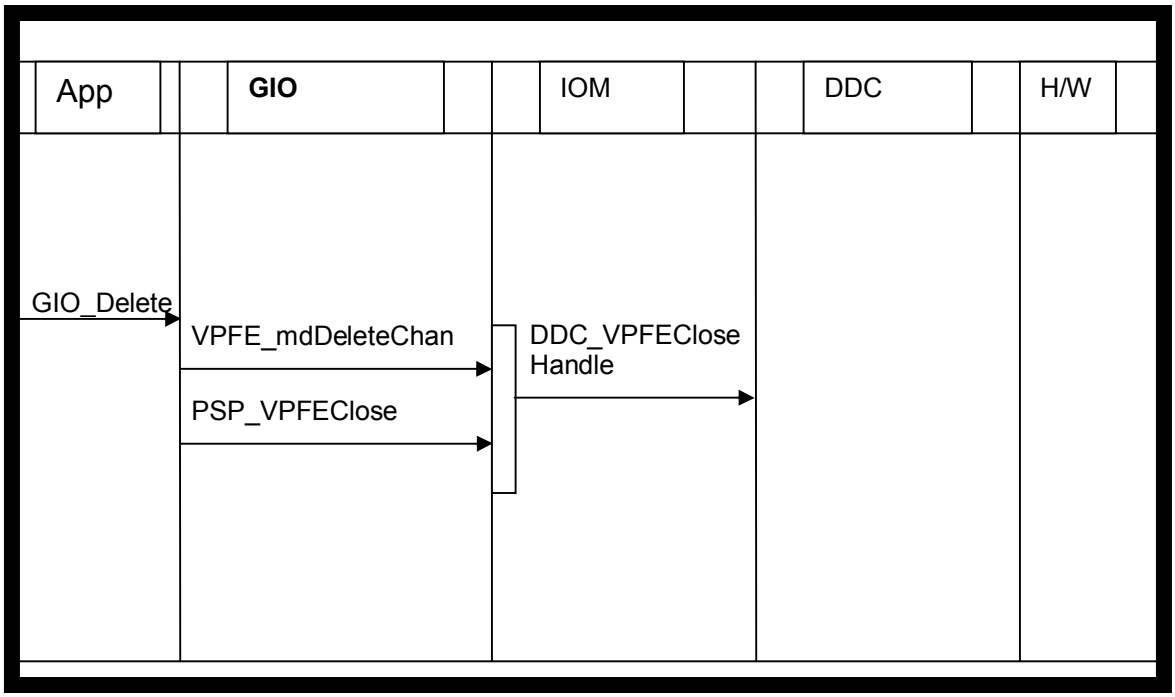

Figure 14 Driver Close Chan Overview

Once the channel is closed it has no life. The user will have to bring the driver back to life by creating the driver through `mdCreateChan ()`.

VPFE_mdDeleteChan

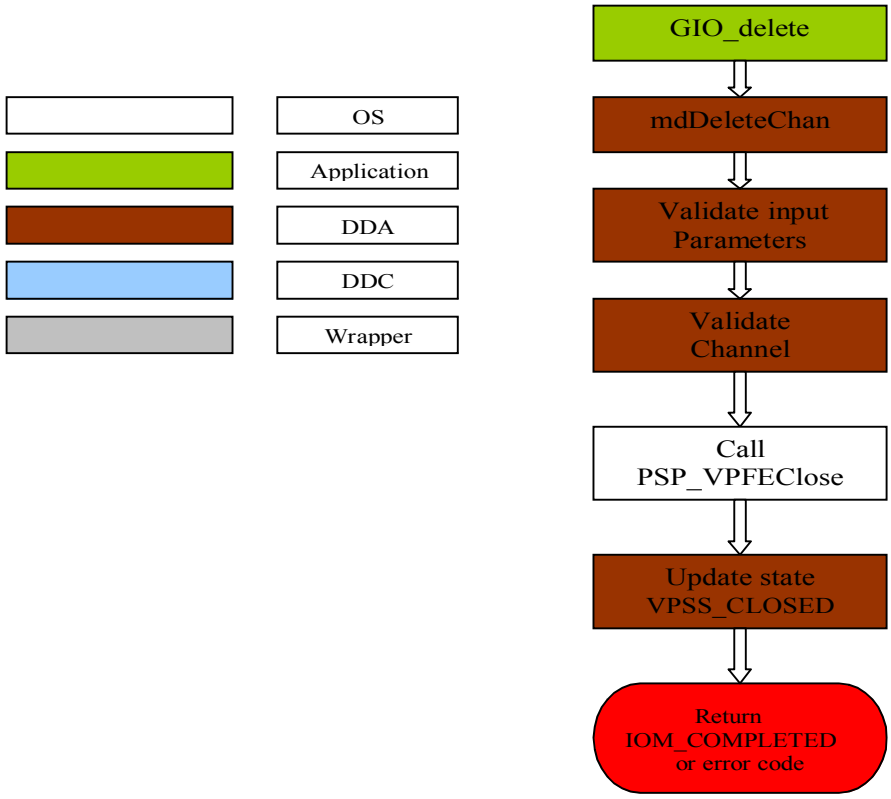

Figure 15 Driver Closed Detailed Flow Diagram – 1

CL1	Function	mdDeleteChan
	Function Prototype	`Int VPFE_mdDeleteChan (Ptr chanp)`
	Input Parameters	Chanp – pointer to chan object
	Output Parameters	Int according to the IOM error codes
	Description	This function would close current session of IO by calling the DDC Layer `_DDC_VPFEClosehandle ( )` function.
	Preconditions	The driver should be opened
	Design	Logic in steps 1. Check the State of the Channel. 2. Close the specific channel and update state as closed.

PSP_VPFEClose

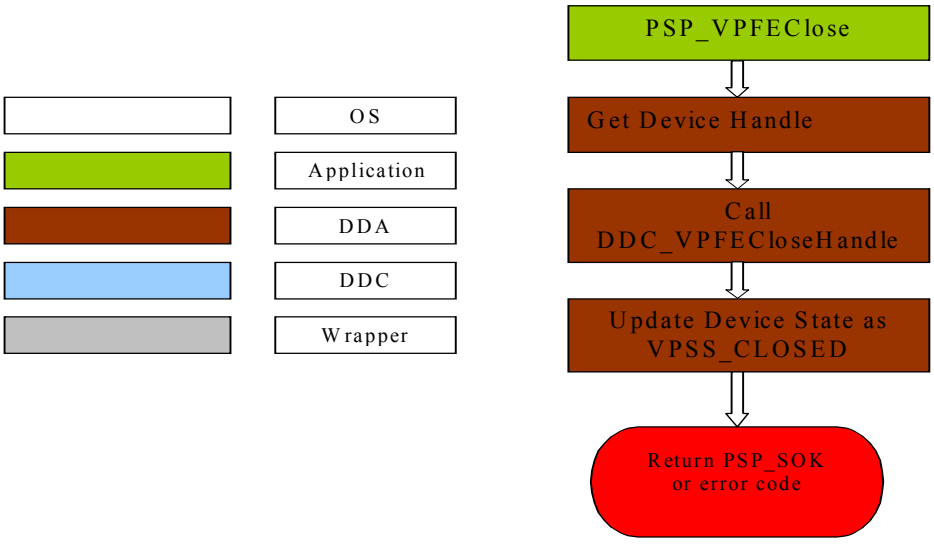

Figure 16 Driver Closed Detailed Flow Diagram – 2

DDC_VPFECloseHandle

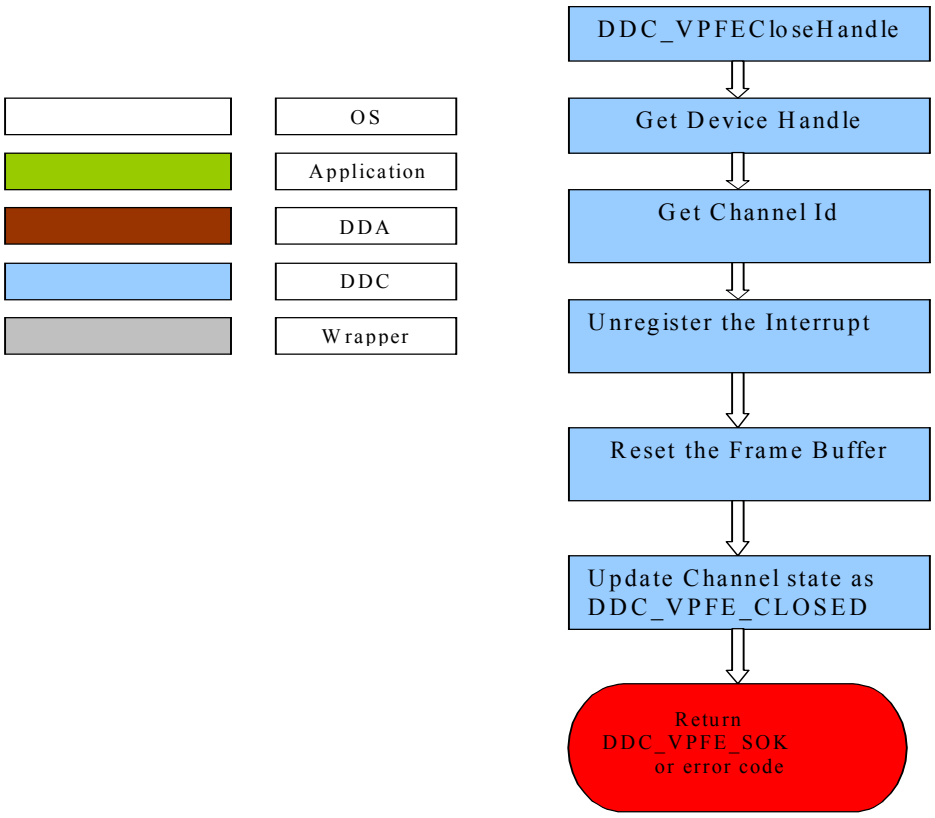

Figure 17 Driver Closed Detailed Flow Diagram – 3

2.2.6 Driver Teardown

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

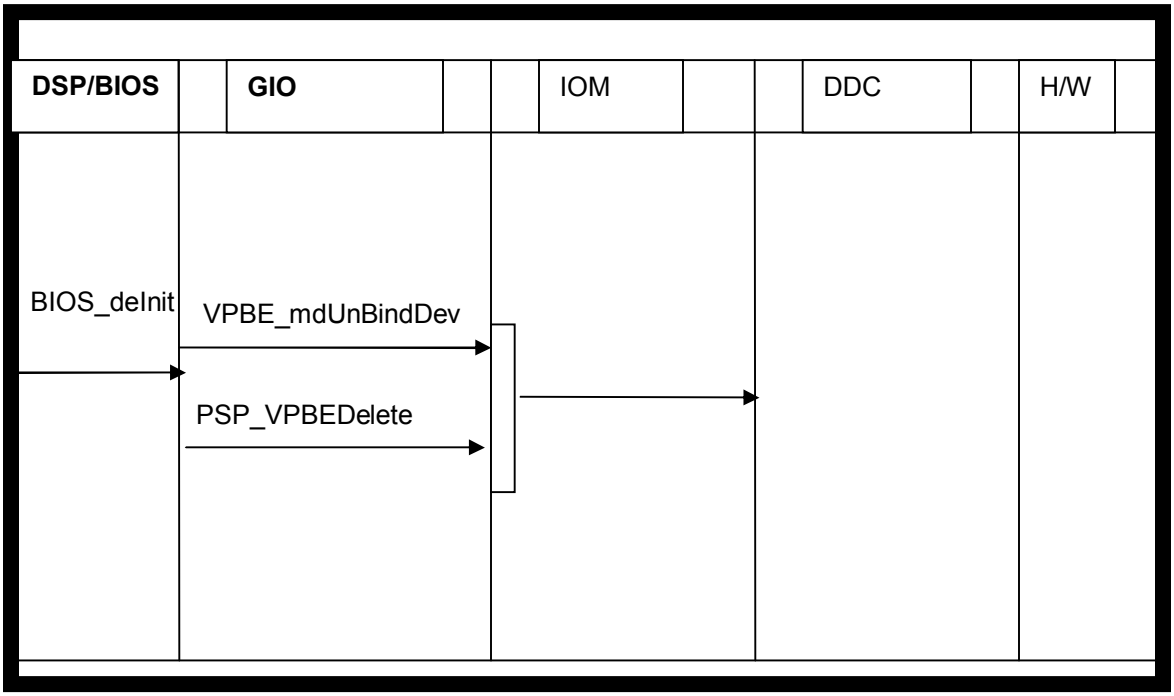

Figure 18 Driver Close Overview

VPFE_mdBindDev

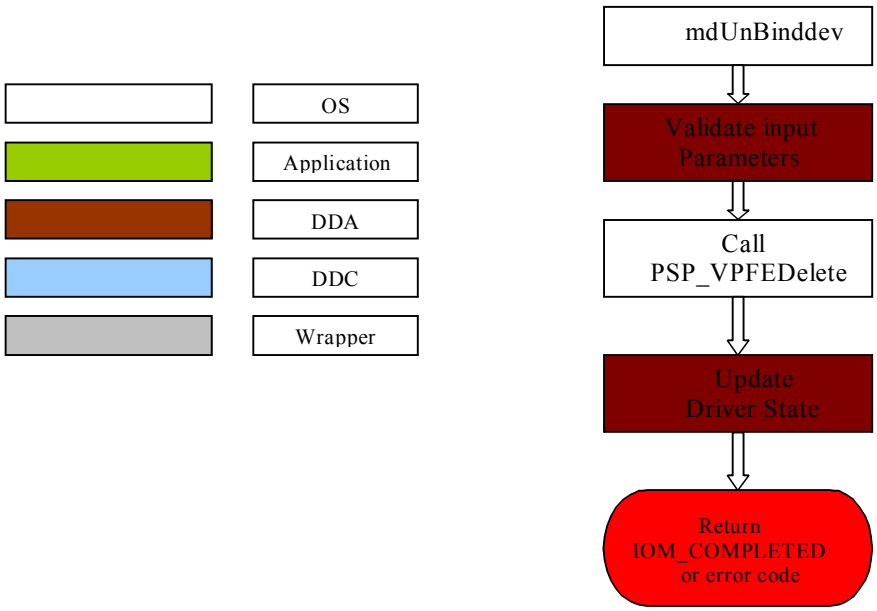

Figure 19 Driver UnBind Detailed Flow Diagram – 1

D1	Function	mdUnbindDev
	Function Prototype	`Int VPFE_mdUnBindDev(Ptr devp)`
	Input Parameters	Devp – Handle to VPFE device
	Output Parameters	Int according to the IOM error code
	Description	This function would remove or unload the the driver instance.
	Preconditions	The DDC has to be created and initialized.
	Design	Logic in steps. 1. Call the PSP_VPFEDelete function. 2. Check the state of driver. 3. Update the state as VPSS_DELETED. 4. Make the driver object as NULL.