VPORT BIOS Device Driver

VPORT Architecture/Design Document

Revision History

Document Version	Author(s)	Date	Comments
0.1	Grishma Parikh	April 26, 2007	Initial Draft
0.2	Grishma Parikh	June 14, 2007	Updated for FVID NORMAL Mode description
0.3	Grishma Parikh	August 6, 2007	Updated to add detailed flow charts for mdcontrolchan() and mdsubmitchan()
0.4	Grishma Parikh	September 14, 2007	Updated for review comments
0.5	Sivaraj R	October 22, 2007	Changed XDC and BIOS versions in system requirements
0.6	Sivaraj R	November 27, 2007	Removed system environment from this document and made reference to system release notes

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.

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

Table of Contents

List Of Figures

1 System Context

The purpose of this document is to explain the device driver design for Video Port used in DM648 SoC using DSP/BIOS operating system running on DSP 64+ joule.

Note: The usage of structure names and field names used throughout this design document is only for indicative purpose. These names shall not necessarily be matched with the names used in source code.

1.1 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
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
EDC	External Device Control
VPORT	Video Port
EDMA	Enhanced Direct Memory Access
OS	Operating System

1.2 Related Documents

1.	SPRU616	DSP/BIOS Driver Developer's Guide
2.	DM648_PSP_VideoportDriver_SRD.pdf	Video Driver Requirement Document
3.	spruem1_Video_Port.pdf	Video Port Datasheet

1.3 Hardware

The VPORT driver architecture presented in this document is situated in the context of DM648 SoC targeted for various video applications. The driver design is in the context of DSP/BIOS running on DSP 64x+ joule core. The below figure shows DM648 Architecture shall be used for Video application.

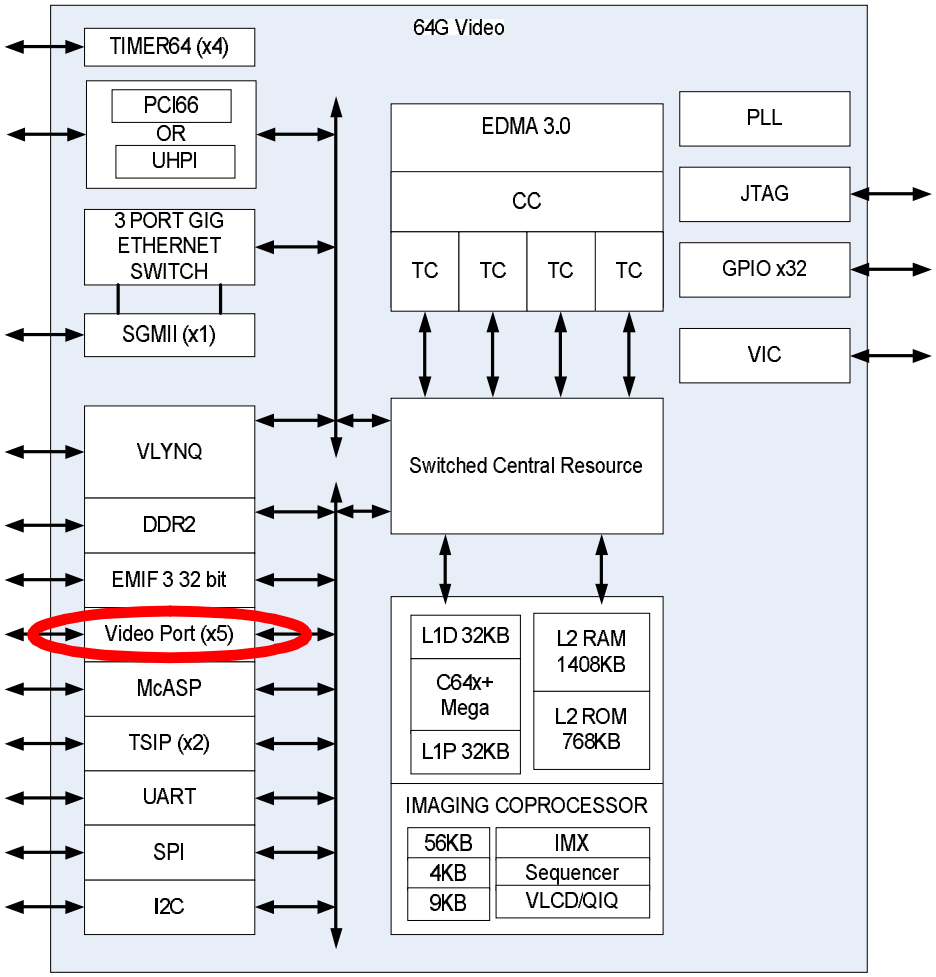

Figure 1 DM648 block diagram for Video

The VPORT module used in DM648 SoC core has the following blocks:

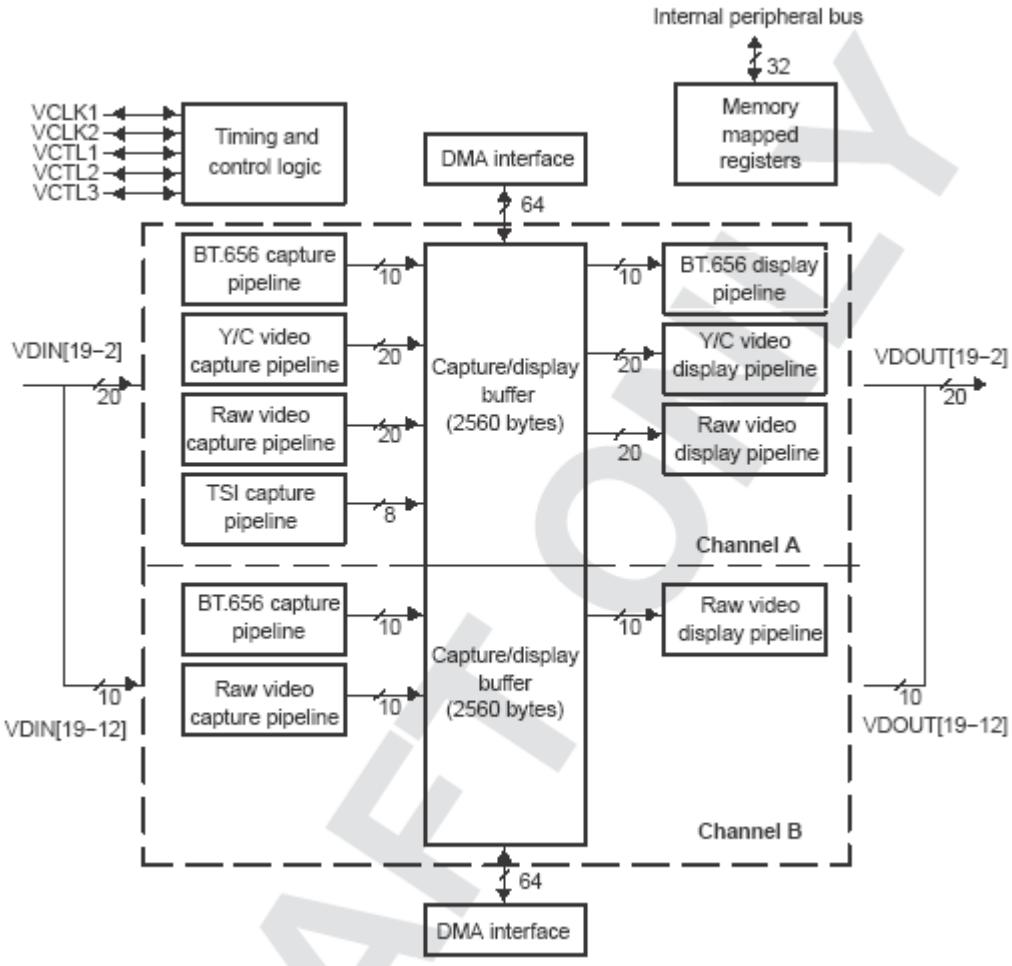

Figure 2 Video Port Block Diagram

1.4 Software

The VPORT mini-driver discussed here is targeted at the DM648 device, running DSP/BIOS on the 64x+ DSP. More details can be found in the later part of this section.

1.4.1 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.

1.4.2 System Architechture

The block diagram below shows the overall system architecture.

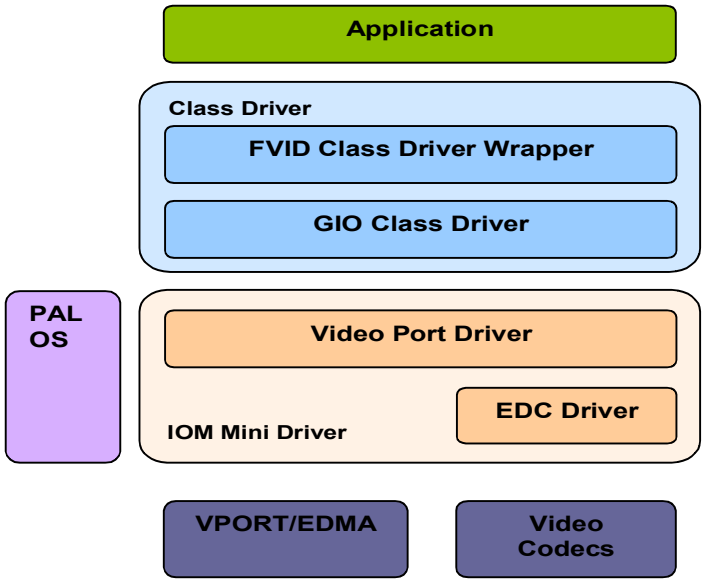

Figure 3 System Architecture

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

IOM is the component that performs the device specific operations. IOM Mini Driver directly controls Video Port and indirectly controls External devices like encoders and decoders through EDC driver.

Figure 5 shows the overall DSP/BIOS device driver architecture. For more information about the IOM device driver model, see the DSP/BIOS Device Driver Developer's Guide (SPRU616). The rest of the document elaborates on the architecture of the Device driver by TI.

1.5 Component Interfaces

In the following subsections, the interfaces implemented by each of the sub-component are specified. Refer to VPORT device driver User Guide documentation for complete details on APIs.

1.5.1 IOM Interface and Generic Layer

The IOM constitutes the Device Driver Manifest to Application. The user may not look into 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 *mdBindDev ()* populates static settings in driver object, creates the necessary interrupt handler, attaches the Driver Core interfaces. All these operations in effect, constitute the "loading" of VPORT Driver implementation. The VPORT Driver contains separate Capture and Display drivers at IOM layer. Capture and Display drivers interface is same to class driver. The IOM mini-driver implements the following API interfaces to the class driver.

S.No	IOM Interfaces	Description
1	mdBindDev ()	Allocates and configures the Video Port specified by devid.
2	mdCreateChan ()	Creates a communication channel in specified mode to communicate data between the application and VPORT device instance. It also configures VPORT channel and allocates required resources.
3	mdDeleteChan ()	Frees a channel and all its associated resources
4	mdControlChan ()	Implements the IOCTLS for VPORT IOM mini driver.
5	mdSubmitChan ()	Submits/retrieves an I/O packet to a channel for processing.

1.5.2 CSLR Interface

The CSL register interface (CSLR) provides register level implementations. CSLR is used by the IOM mini driver to configure VPORT registers. CSLR is implemented as a header file that has CSLR macros and register overlay structure.

1.6 Design Philosophy

This device driver is written in conformance to the DSP/BIOS IOM device driver model and handles communication to and from the VPORT hardware.

1.6.1 The Port and Channel Concept

The IOM model provides the concept of the *Port* and *Channel* for the realization of the device and its communication path as a part of the driver implementation. The VPORT driver provides simultaneous capture or display operations on five video ports. This is supported by providing two Capture channels and one Display channel per port in order to perform IO operations.

The *Port Object* maintains the state of the VPORT device or an instance. The *port* can also be called as *instance* or *device* and the names can be used interchangeably. DM648 SoC contains five instances of VPORT, and the driver for this needs to maintain five port objects. The following figure shows the generic port-channel-hardware mapping for VPORT Capture driver.

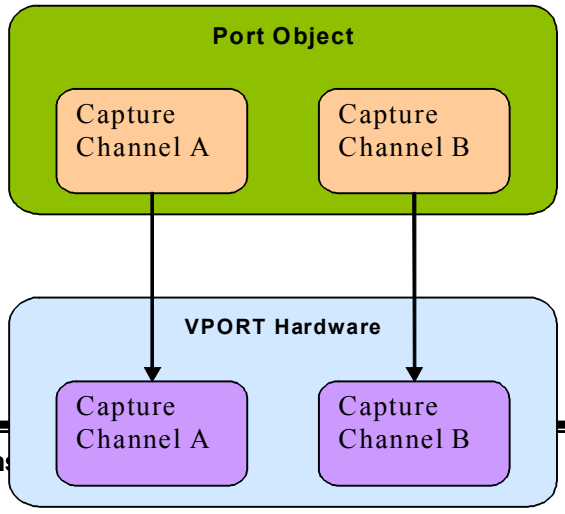

Figure 4 Port and Capture Channel Object

The following figure shows the generic port-channel-hardware mapping for VPORT Display driver

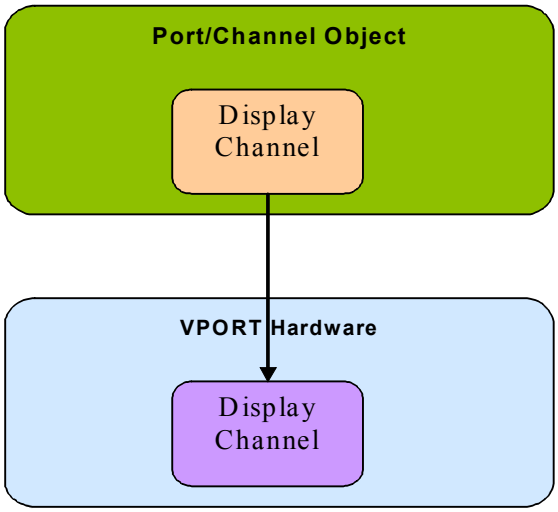

Figure 5 Port and Display Channel Object

1.6.2 Design Constrains

VPORT mini-driver imposes the following constraint(s).

- VPORT driver doesn't support synchronization to another video port.

2 VPORT Driver Software Architecture

This section details the data structures used in the VPORT mini-driver and the interface it presents to the GIO layer. A diagrammatic representation of the mini driver functions is presented and then the usage scenario is discussed in some more details.

Following this, we'll discuss the deployed driver or the dynamic view of the driver where the driver operational scenarios are presented.

2.1 Static View

2.1.1 Functional Decomposition

The driver is designed keeping a device, also called port and channel concept in mind. The instance of VPORT is treated as a device, which each can have a single display and dual capture channels for DM648 SoC.

The Capture driver uses two internal data structures, port object and channel objects. Display driver uses one channel object to maintain its state during execution. The VPORT peripheral needs the port instance to maintain its state. The channel object holds the IOM channel state during execution. These are explained in greater detail in the following *Data Structures* sub-section. The following figure shows the static view of DM648 VPORT Capture driver.

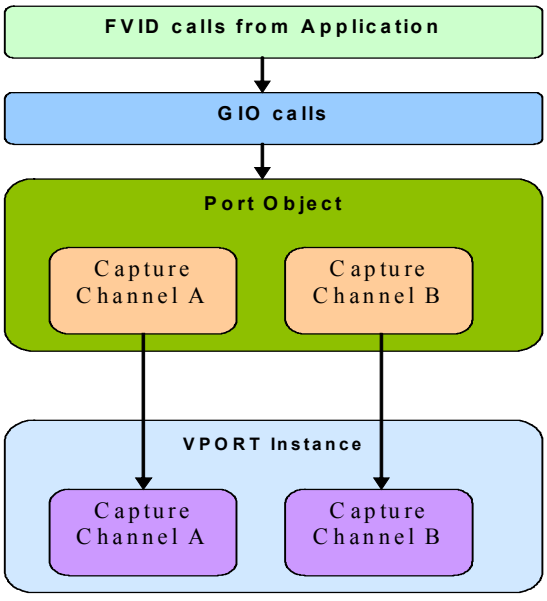

Figure 6 VPORT Capture driver static view

The following figure shows the static view of DM648 VPORT Display driver.

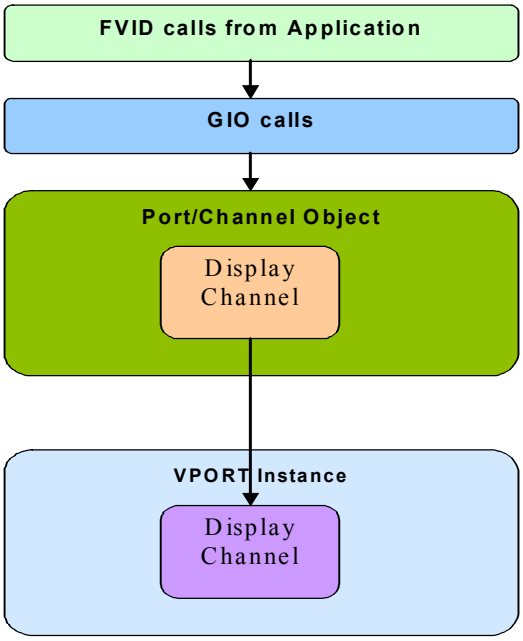

Figure 7 VPORT Display driver static view

2.1.2 Data Structures

The mini-driver employs the PortObj and ChannelObj structures to maintain state of the port and channel respectively.

In addition, the driver has two other structures defined – the device params and channel params. The device params structure is used to pass on data to initialize the driver during DSP-BIOS initialization. The channel params structure is used to specify required characteristics while creating a channel.

The following sections provide major data structures maintained by IOM interface. For more details about IOM data structures and their usage can be found in the API reference guide.

2.1.2.1 The Port Object(IOM) Capture Driver

S.No	Structure Elements (PortObj)	Description
1	*status*	Port status register, contains information on whether port is opened, configured, etc
2	*vportBase*	Points to base address of Video Port
3	*chanObj []*	Holds two channel objects for capture channels only

2.1.2.2 The Channel Object(IOM)

S.No	Structure Elements (_VPORT_ChanObj)	Description
1	*status*	Channel status. Channel opened, configured, allocated, etc...
2	*portNum*	Port number for which channel is created
3	*chanNum*	Channel number to distinguish from one of the two capture channel
4	*vportRegs*	Points to Video Port base address
5	*edmaChanNum []*	Array to store edma channel number
6	*edmaAddr []*	Array to store VPORT FIFO address for EDMA transfer
7	*evtQueue*	EDMA Transfer Controller to use for the channel
8	*hEdmaChan []*	To store EDMA channel handle
9	*edmaLinkChanNum*	To store EDMA link channel number

10	*vIntMask*	To store Video Port interrupt mask value
11	*viops []*	Object type of FVID_Frame to store image frame buffers in LEGACY Mode
12	*qIn*	Queue element for queue in
13	*qOut*	Queue element for queue out
14	*qPullDown*	Queue element
15	*hEdma*	To store EDMA driver handle
16	*tcc []*	To store EDMA tcc number
17	*mergeFlds*	Flag to identify fields are merged or separated
18	*interlaced*	Flag to identify fields are interlaced or progressive
19	*cbFxn*	IOM callback function
20	*cbArg*	IOM callback Argument
21	*vIntFxn*	Callback function for VPORT interrupt
22	*queEmpty*	To reflect queue status
23	*scale*	Flag to indicate scaling enabled or disabled
24	*resmpl*	Flag to indicate resampling enabled or disabled
25	*yPitch*	Luminance pitch
26	*cPitch*	Chrominance pitch
27	*numLines*	Number of lines in one frame
28	*numPixels*	Number of pixels per line
29	*numLinesFld1*	Number of lines in one field
30	*numFrms*	Number of frame buffers used by driver
31	*numEvents*	Number of EDMA events

32	*numEventsFld1*	Number of EDMA events for one field
33	*yThrld*	Luminance FIFO threshold value
34	*cThrld*	Chrominance FIFO threshold value
35	*curViop*	Pointer to current video input/output packet
36	*nextViop*	Pointer to next video input/output packet
37	*mrViop*	Pointer to most recent video input/output packet
38	*mode*	Mode of operation of channel
39	*edcFxns*	Function pointer table of EDC (encoder or decoder)
40	*edcHandle*	Handle of EDC device
41	*packetIOM*	Pointer to IOM packet
42	*vIntCbArg*	Argument of VPORT interrupt callback function
43	*segId*	Memory segment ID in which buffers to be allocated
44	*bufSz*	Size of buffer
45	*lastLineNum*	Number of recent line captured or displayed
44	*nextEDMARlds*	Next EDMA reload parameter
45	*numTotalLinesFld1*	Number of total lines in one field
46	*autoSyncEnable*	Flag to indicate auto sync enabled or disabled
47	*asyncModeEnable*	Flag to indicate asynchronous enabled or disabled
48	*pullDownMode*	To select pull down mode for

		display
49	*asyncFrameCt*	Async frame counter
50	*asyncCallBackFxn*	Asynchronous mode callback function
51	*frmSz*	Frame line size for vertical interrupt
52	*asyncCbArg*	Argument for asynchronous transfer callback function
53	*startFlag*	Start flag for asynchronous transfer mode
54	*driverMode*	Driver operation mode 0 = normal mode, 1 = legacy mode
55	*numQueBuf*	Number of queued buffer in normal mode
56	*alignment*	Frame buffer alignment
57	*fldOp*	Flag for field operation or frame operation
58	*vportChanRegs*	Points to respective channel base of Vport
59	*lineIntInfo*	Line interrupt parameters structure

2.1.2.3 The Device Params

The application passes a data structure VPORT_PortParams that is used to initialization function of the driver. The params are explained below:

S.No	Structure Elements (PortObj)	Description
1	*dualChanEnable*	Dual channel mode enable
2	*vc1Polarity*	Vport control pin 1 polarity
3	*vc2Polarity*	Vport control pin 2 polarity
4	*vc3Polarity*	Vport control pin 3 polarity
5	*edcTbl []*	function tables for EDC driver

modules

2.1.2.4 Capture Channel Params

The application passes a data structure VPORTCAP_Params at time of channel creation. The params are explained below:

S.No	Structure Elements (PortObj)	Description
1	*cmode*	Capture mode settings
2	*fldOp*	Field & frame operation(Please refer to vport.h for available values)
3	*scale*	Indicates whether to enable horizontal ½ scaling. The ½-scaling mode is used to reduce the horizontal resolution of captured luminance and chrominance data by a factor of two.
4	*resmpl*	Indicates whether to enable chrominance sub-sampling
5	*bpk10Bit*	10-bit bit-pack mode(Please refer to vportcap.h for available values)
6	*hCtRst*	Horizontal counter reset mode
7	*vCtRst*	Vertical counter reset mode
8	*fldDect*	Enable whether to use FID input or field detection logic based on the timing relation of hsync and vsync
9	*extCtl*	Enable external timing control
10	*fldInv*	Enable inversion of the detected fid

11	*fldXStrt1*	Field 1 X start
12	*fldYStrt1*	Field 1 Y start
13	*fldXStrt2*	Field 2 X start
14	*fldYStrt2*	Field 2 Y start
15	*fldXStop1*	Field 1 X stop
16	*fldYStop1*	Field 1 Y stop
17	*fldXStop2*	Field 2 X stop
18	*fldYStop2*	Field 2 Y stop
19	*thrld*	Video FIFO threshold
20	*numFrmBufs*	Number of frame buffers that the driver allocates. Minimum of 3 buffers are recommended for proper operation
21	*alignment*	Frame buffer alignment
22	*mergeFlds*	Indicates whether to store two fields in interleaved manner or to store them separately
23	*segId*	Memory segment ID, used by driver to allocate video frame buffer
24	*autoSyncEnable*	Boolean to enable auto sync
25	*hEdma*	EDMA3 driver handle

2.1.2.5 Display Channel Params

The application passes a data structure VPORTDIS_Params at time of channel creation. The params are explained below:

S.No	Structure Elements (PortObj)	Description
1	*dmode*	Display mode settings

2	*fldOp*	Field & frame operation(Please refer to vport.h for available values)
3	*scale*	Indicates whether to enable 2x scaling. The 2x-scaling mode is used to double the horizontal resolution of output luminance and chrominance data.
4	*resmpl*	Indicates whether to enable Chroma sub-sampling
5	*defValEn*	Default value enable
6	*bpk10Bit*	10-bit bit-pack mode(Please refer to vportdis.h for available values)
7	*vctl1Config*	VCTL1 pin selection(Please refer to vportdis.h for available values)
8	*vctl2Config*	VCTL2 pin selection(Please refer to vportdis.h for available values)
9	*vctl3Config*	VCTL3 pin selection(Please refer to vportdis.h for available values)
10	*extCtl*	Enable external timing control
11	*frmHSize*	Frame horizontal size
12	*frmVSize*	Frame vertical size
13	*imgHOffsetFld1*	Image horizontal offset field 1
14	*imgVOffsetFld1*	Image vertical offset field 1
15	*imgHSizeFld1*	Image line size field 1
16	*imgVSizeFld1*	Image total lines field 1
17	*imgHOffsetFld2*	Image horizontal offset field 2
18	*imgVOffsetFld2*	Image vertical offset field 2

19	*imgHSizeFld2*	Image line size field 2
20	*imgVSizeFld2*	Image total lines field 2
21	*hBlnkStart*	Horizontal blanking start value
22	*hBlnkStop*	Horizontal blanking stop value
23	*vBlnkXStartFld1*	Vertical blanking pixel start value field 1
24	*vBlnkYStartFld1*	Vertical blanking line start value field 1
25	*vBlnkXStopFld1*	Vertical blanking pixel stop value field 1
26	*vBlnkYStopFld1*	Vertical blanking line stop value field 1
27	*vBlnkXStartFld2*	Vertical blanking pixel start value field 2
28	*vBlnkYStartFld2*	Vertical blanking line start value field 2
29	*vBlnkXStopFld2*	Vertical blanking pixel stop value field 2
30	*vBlnkYStopFld2*	Vertical blanking line stop value field 2
31	*xStartFld1*	Pixel start field 1
32	*yStartFld1*	Line start field 1
33	*xStartFld2*	Pixel start field 2
34	*yStartFld2*	Line start field 2
35	*hSyncStart*	Horizontal sync start
36	*hSyncStop*	Horizontal sync stop
37	*vSyncXStartFld1*	Vertical sync pixel start field 1
38	*vSyncYStartFld1*	Vertical sync line start field 1
39	*vSyncXStopFld1*	Vertical sync pixel stop field

		1
40	*vSyncYStopFld1*	Vertical sync line stop field 1
41	*vSyncXStartFld2*	Vertical sync pixel start field 2
42	*vSyncYStartFld2*	Vertical sync line start field 2
43	*vSyncXStopFld2*	Vertical sync pixel stop field 2
44	*vSyncYStopFld2*	Vertical sync line stop field 2
45	*yClipLow*	Luminance (Y) low clipping value
46	*yClipHigh*	Luminance (Y) high clipping value
47	*cClipLow*	Chrominance (CB/CR) low clipping value
48	*cClipHigh*	Chrominance (CB/CR) high clipping value
49	*yDefVal*	Luminance (Y) default value
50	*cbDefVal*	Chrominance (CB) default value
51	*crDefVal*	Chrominance (CR) default value
52	*rgbX*	RGB extract enable/disable select (1 enable, 0 disable)
53	*incPix*	Pixel increment for RAW mode only (0 else)
54	*thrld*	Video FIFO threshold
55	*numFrmBufs*	Number of frame buffers to be used in driver
56	*alignment*	Frame buffer alignment
57	*mergeFlds*	Separated or merged fields (0 separated fields, 1 merged fields)
58	*segId*	Memory segment ID, used by

		driver to allocate video frame buffer
59	*hEdma*	EDMA3 driver handle

2.2 Dynamic View

2.2.1 The Execution Threads

The device drivers typically implement Synchronous interface to the user. The VPORT device driver operation involves following execution threads:

BIOS thread: Function to load VPORT driver will be under BIOS OS initialization.

Application thread: Creation of channel, Control of channel, deletion of channel and processing of VPORT data will be under application thread. All Synchronous IO occur in the application thread of control, the calling thread may suspend for the requested transaction to complete.

2.2.2 Capture/Display using VPORT driver

With VPORT driver, the application can perform IO operation using one of below supported modes:

- LEGACY Mode

- NORMAL Mode

Below table provides list of APIs supported by VPORT driver:

API	Description
FVID_create	Allocate (optional) and initialize an FVID channel object
FVID_delete	De-allocate (optional) an FVID channel object
FVID_alloc	Get a pointer for allocated buffer from driver to application. Supported in LEGACY mode only
FVID_free	Relinquish a video buffer back to the driver. Supported in LEGACY mode only
FVID_control	Send a control command to the mini-driver
FVID_exchange	Exchange an application-owned buffer for a driver-owned buffer

FVID_dequeue	Get a pointer for allocated buffer from driver to application. Supported in NORMAL mode only
FVID_queue	Relinquish a video buffer back to the driver. Supported in NORMAL mode only
FVID_allocBuffer	Allocate a frame buffer using the driver's memory allocation routines. Supported in NORMAL mode only
FVID_freeBuffer	Free the buffer allocated via FVID_allocBuffer(). Supported in NORMAL mode only

In LEGACY mode, application can perform IO operation using FVID_exchange, FVID_alloc and FVID_free calls (corresponding GIO call is GIO_submit and corresponding IOM function is mdSubmitChan ()). This LEGACY Mode is compatible to DM642 VPORT driver API calls.

In NORMAL mode, application can perform IO operation using FVID_exchange, FVID_queue and FVID_dequeue calls (corresponding GIO call is GIO_submit and corresponding IOM function is mdSubmitChan ())

The buffers to be captured or displayed should be passed as argument in FVID_exchange, FVID_alloc or FVID_free calls. Driver will return the status of buffer exchanged for capture or display operation. e.g. IOM_COMPLETED for success.

2.2.3 Functional Decomposition

There are separate functions at IOM layer for Capture and Display driver that are described below.

2.2.3.1 Capture Driver

2.2.3.1.1 mdBindDev

During *mdBindDev*, the mini-driver has access only to pointers of device parameters. Memory for these structures is to be allocated outside the driver by the application.

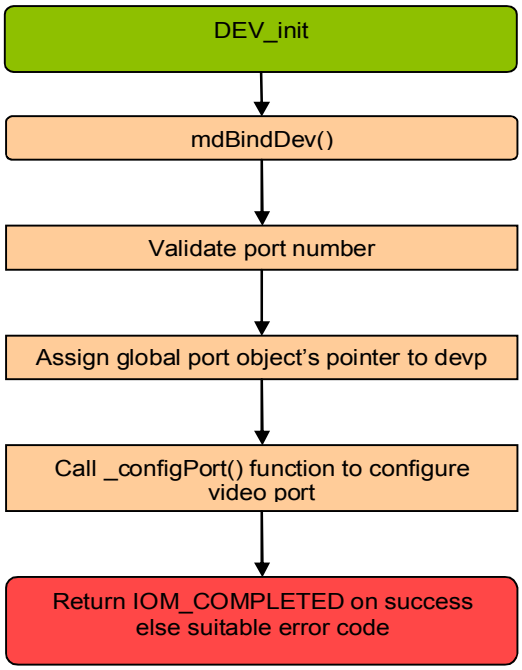

Figure 8 mdBindDev () flow diagram

2.2.3.1.2 mdCreateChan

When application calls FVID_create(), mdCreateChan() will get called internally. Application has to pass proper channel configuration parameters as an argument to FVID_create().

In Legacy mode this API is compatible to usage of DM642 FVID create API. The 'no. of buffers' ('numFrmBufs' member) in the channel parameters is the flag used to identify this mode. The value must be greater than or equal to 3 for the driver to work in this mode.

In this legacy mode, this call will allocate frame buffers (Number of Buffers provided as parameter), initialize EDMA channels and configure video port registers. These frame buffers will be used later for FVID exchange and other APIs.

In the normal mode (where the 'numFrmBufs' in the channel parameters is 0), the driver will not allocate frame buffers for FVID exchange and other APIs. Applications have to create buffers for this purpose. It is suggested that applications should use the APIs provided with driver for frame buffer allocation purpose.

As part of channel creation, VPORT driver will request 3 EDMA channels and 12 (3*4) EDMA parameter tables. Here, Y, Cb and Cr each data transfer will use 1 EDMA channel and 4 EDMA parameter tables.

After successful operation of create channel, handle is returned to the application. This handle is used afterwards to make calls for different channel operations.

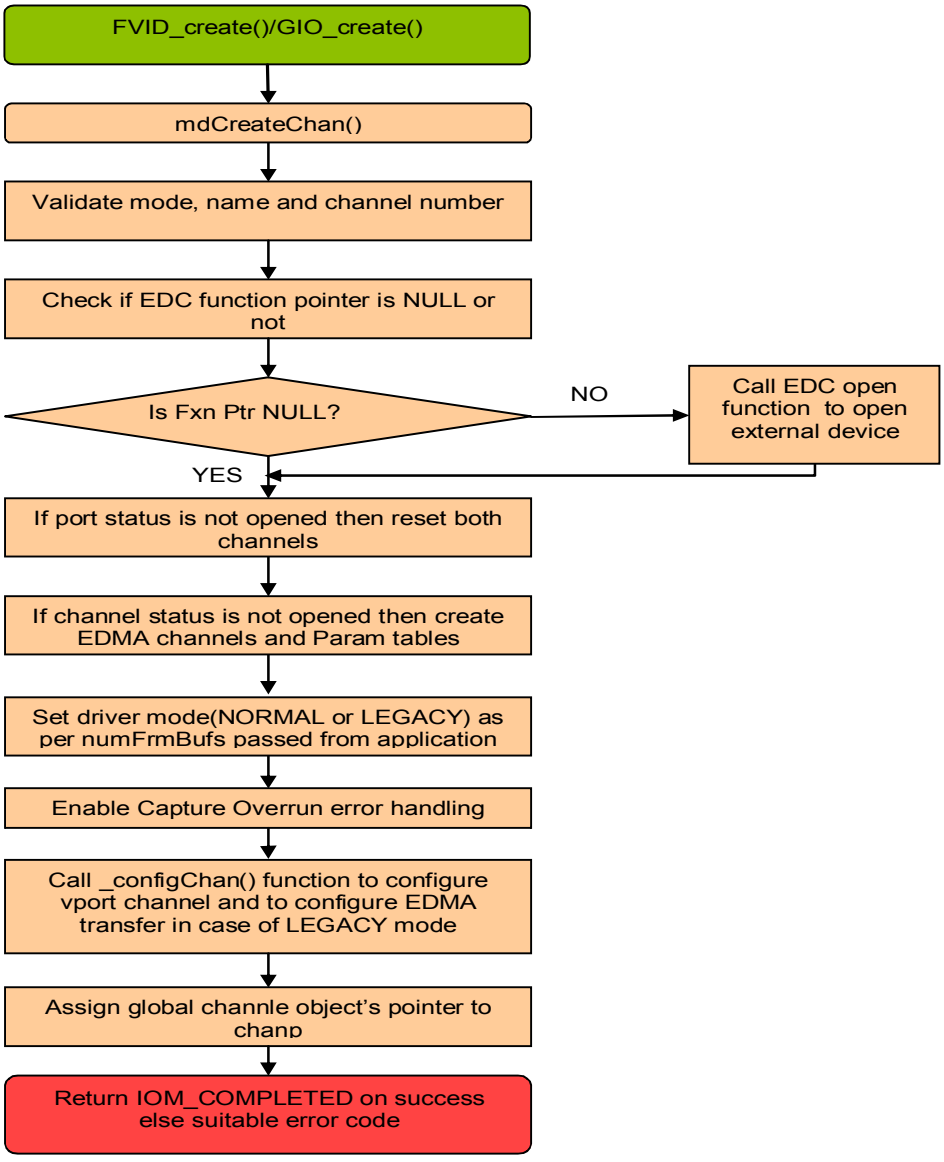

Figure 9 mdCreateChan () flow diagram

2.2.3.1.3 mdDeleteChan

mdDeleteChan will release all resources created for Capture channel and make channel status closed.

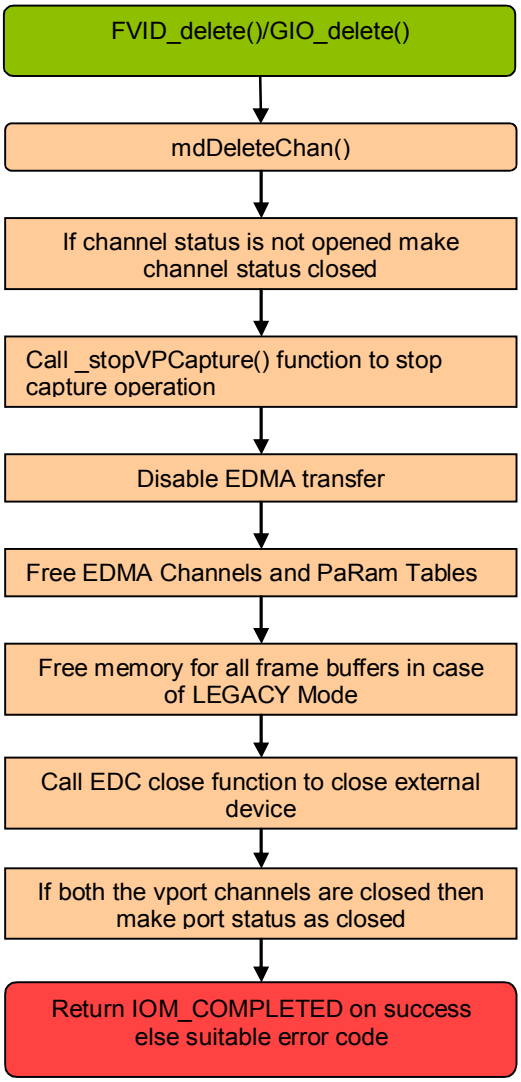

Figure 10 mdDeleteChan () flow diagram

2.2.3.1.4 mdControlChan

mdControlChan is used to call different ioctls supported by driver. External device configuration is also done by calling mdControlChan with EDC specific control commands.

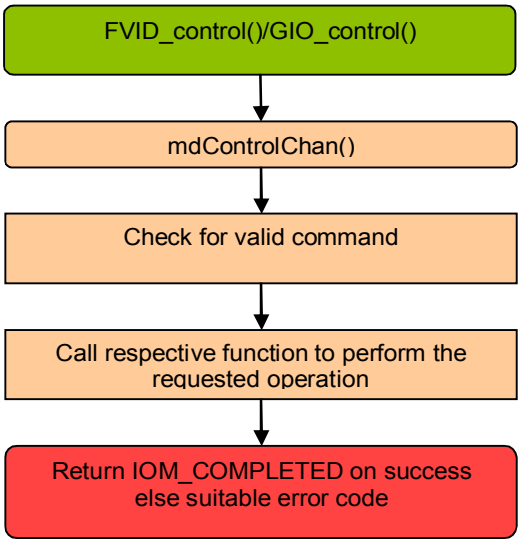

Figure 11 mdControlChan () flow diagram

mdControlChan is supporting following ioctls:

1) VPORT_CMD_START
2) VPORT_CMD_STOP
3) VPORT_CMD_SET_VINTCB
4) VPORT_CMD_COVR_RECOVER
5) VPORT_CMD_CONFIG_CHAN
6) VPORT_CMD_GET_NUM_IORQST_PENDING
7) VPORT_CMD_GET_PARAMS
8) FVID_ALLOC_BUFFER
9) FVID_FREE_BUFFER
10) VPORTCAP_CMD_SET_LINE_INT
11) VPORTCAP_CMD_GET_NUMLINES_CAPTURED

The detailed flow diagrams of above ioctls are as follow:

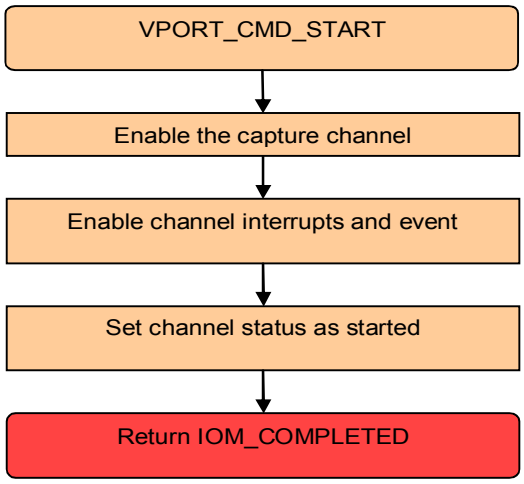

Figure 12 VPORT_CMD_START flow diagram

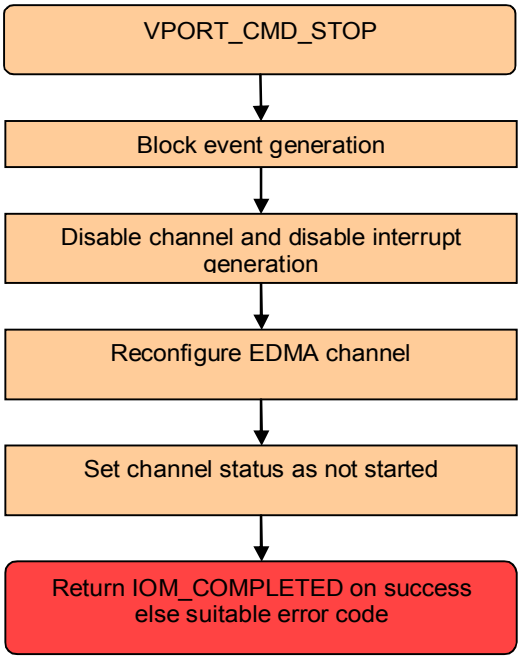

Figure 13 VPORT_CMD_STOP flow diagram

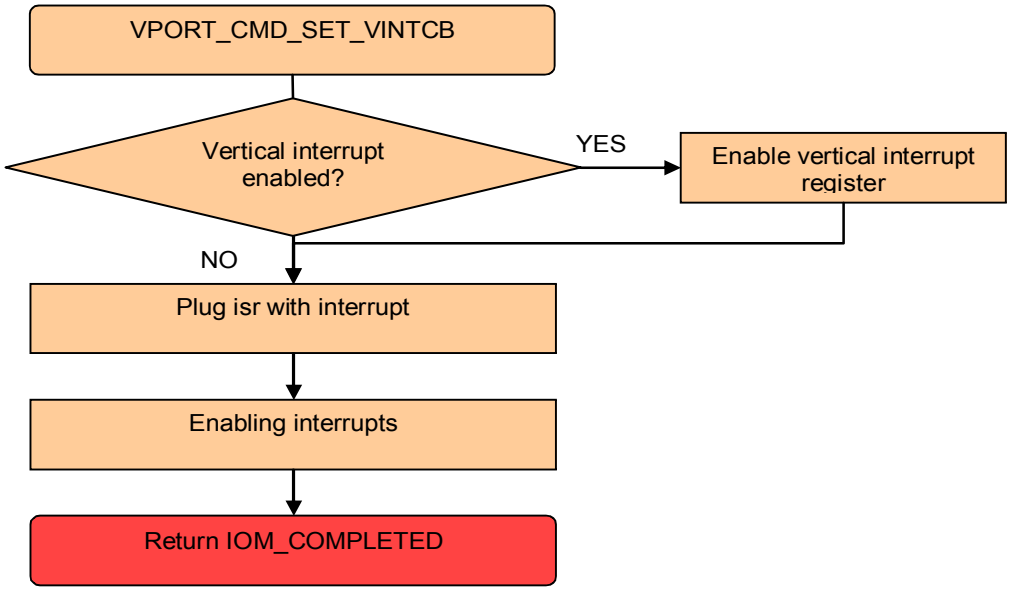

Figure 14 VPORT_CMD_SET_VINTCB flow diagram

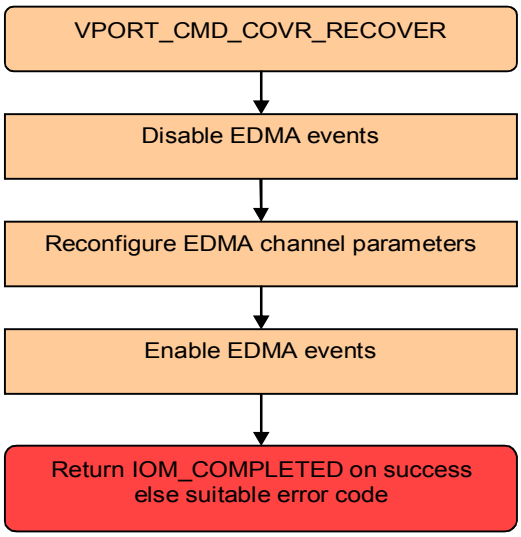

Figure 15 VPORT_CMD_COVR_RECOVER flow diagram

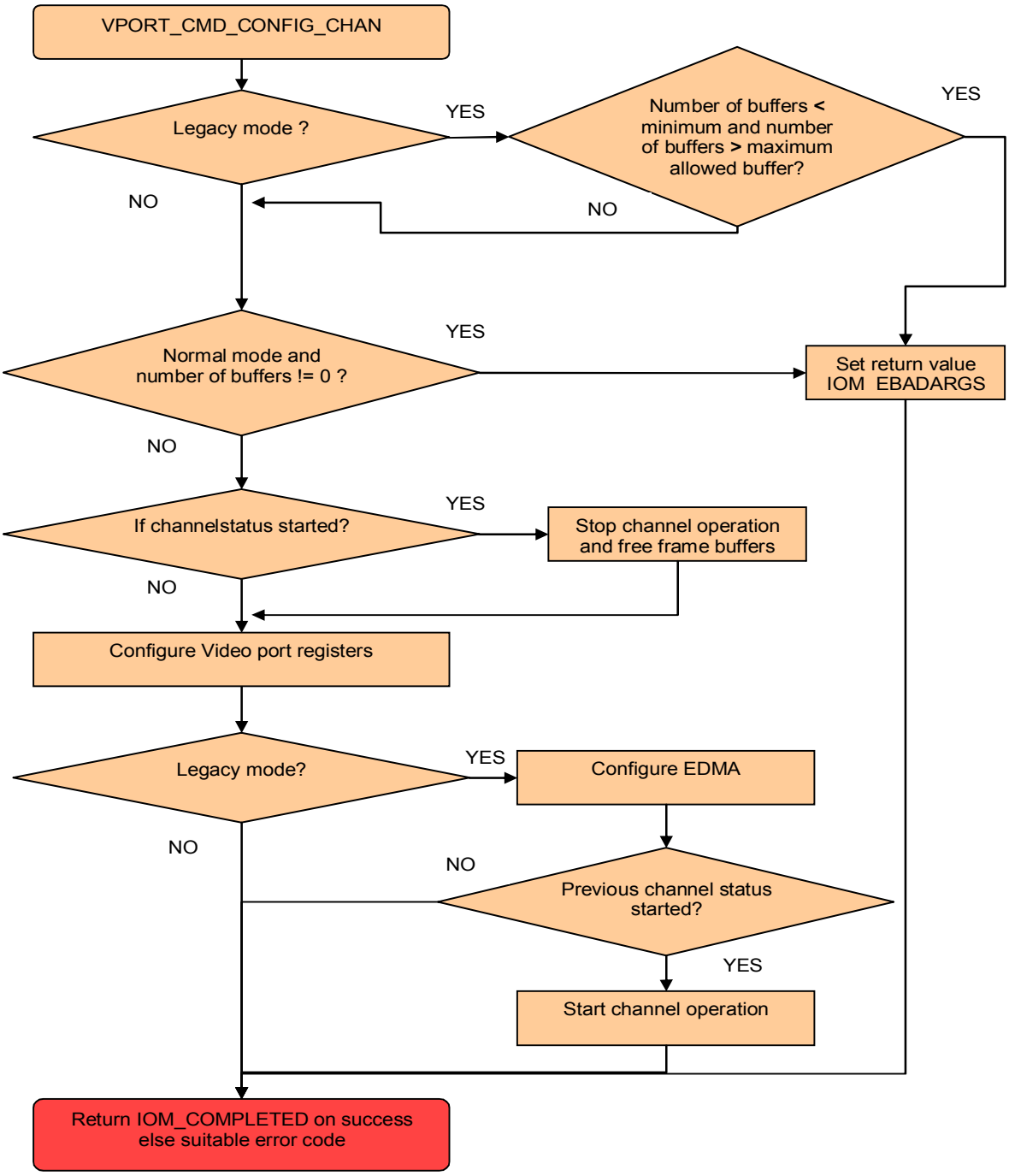

Figure 16 VPORT VPORT_CMD_CONFIG_CHAN flow diagram

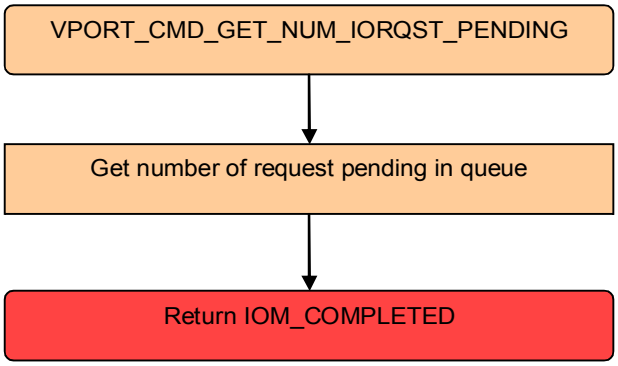

Figure 17 VPORT_CMD_GET_NUM_IORQST_PENDING flow diagram

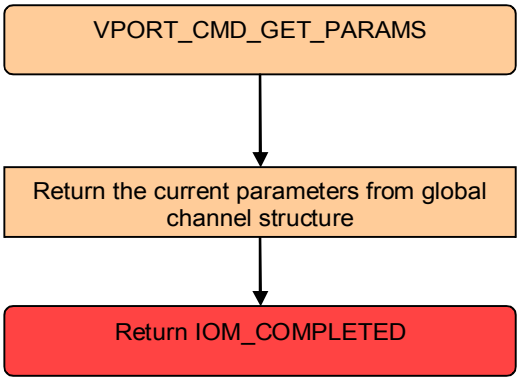

Figure 18 VPORT_CMD_GET_PARAMS flow diagram

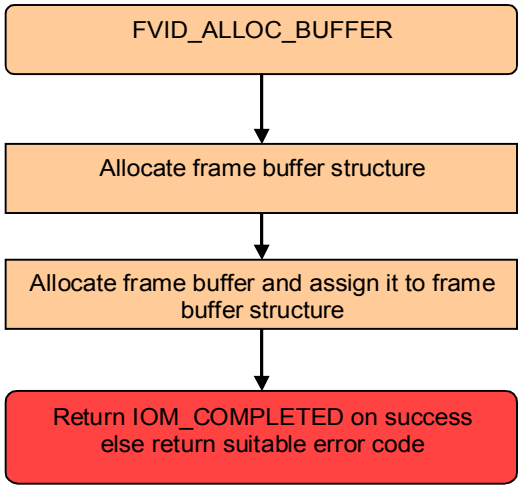

Figure 19 FVID_ALLOC_BUFFER flow diagram

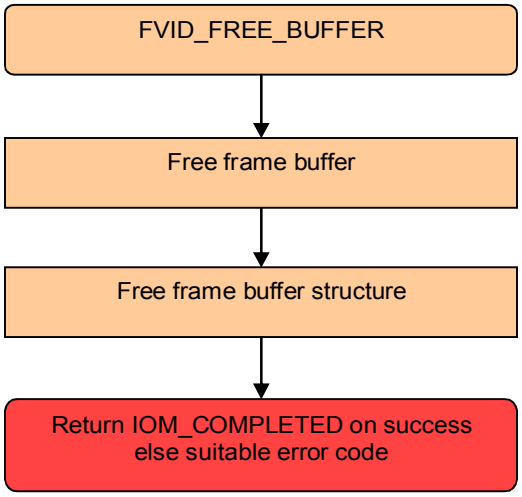

Figure 20 FVID_FREE_BUFFER flow diagram

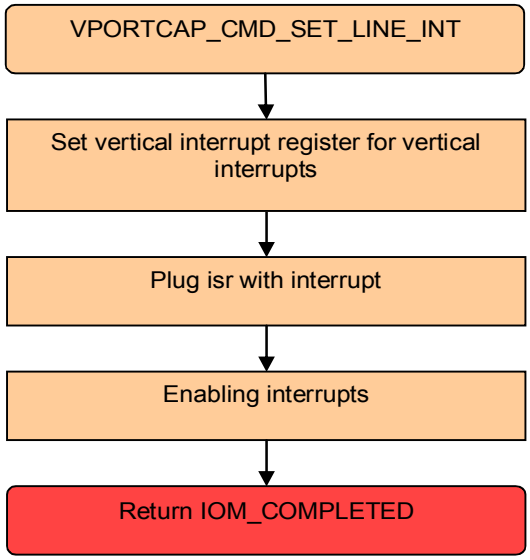

Figure 21 VPORTCAP_CMD_SET_LINE_INT flow diagram

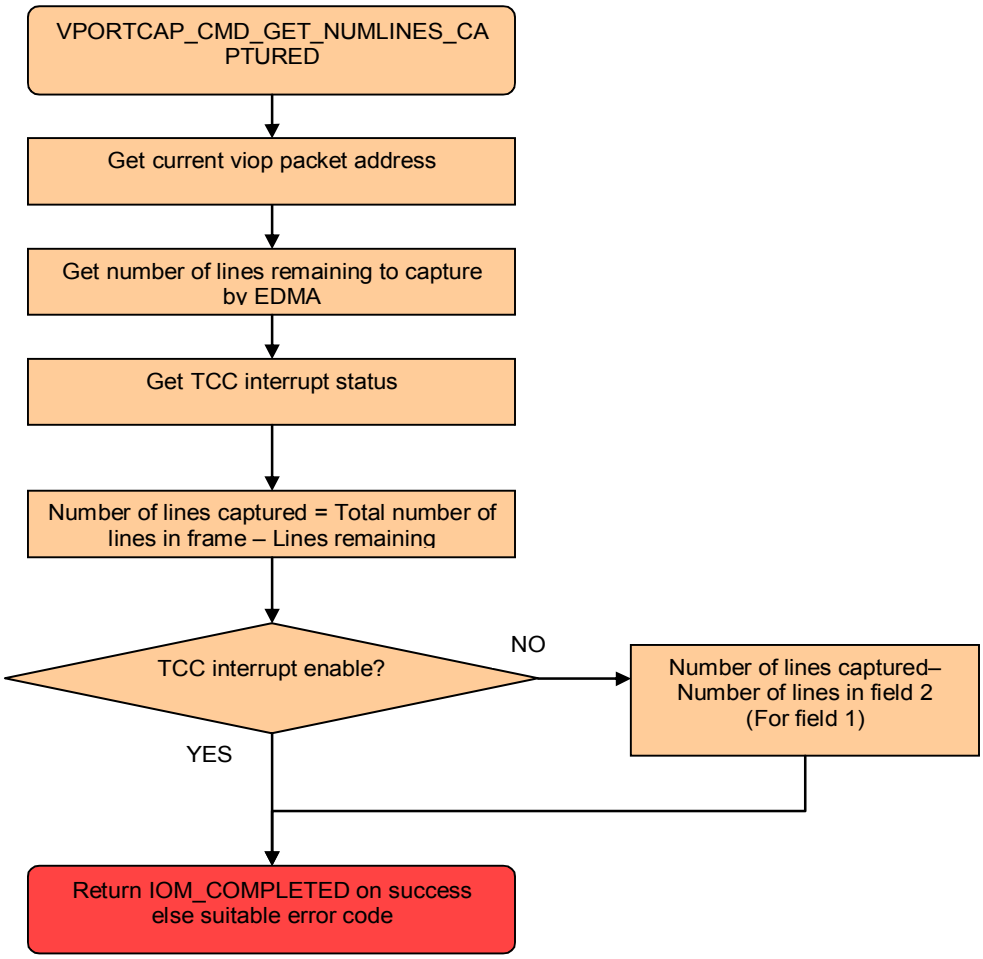

Figure 22 VPORTCAP_CMD_GET_NUMLINES_CAPTURED flow diagram

2.2.3.1.5 mdSubmitChan

This function handles buffer management and exchange between video driver and application. In case of NORMAL mode, FVID_allocBuffer and FVID_freeBuffer calls can be used to allocate and free frame buffers.

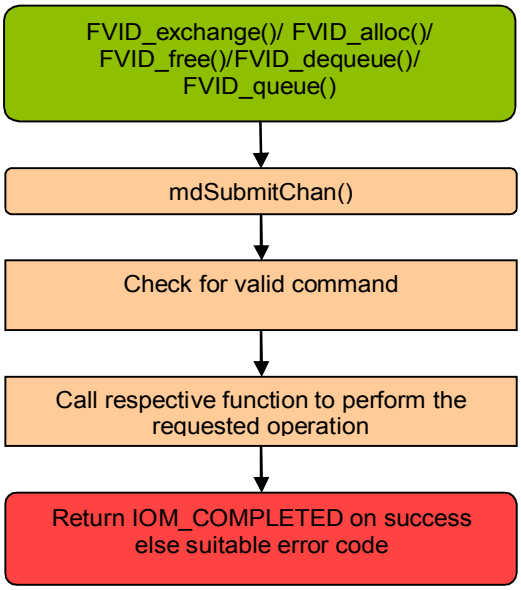

Figure 23 mdSubmitChan () flow diagram

mdSubmitChan is called by three different FVID layer calls for LEGACY Mode:

1) FVID_alloc

2) FVID_exchange

3) FVID_free

mdSubmitChan is called by three different FVID layer calls for NORMAL Mode:

1) FVID_dequeue

2) FVID_exchange

3) FVID_queue

The detailed flow diagram of above FVID layer calls are as follow:

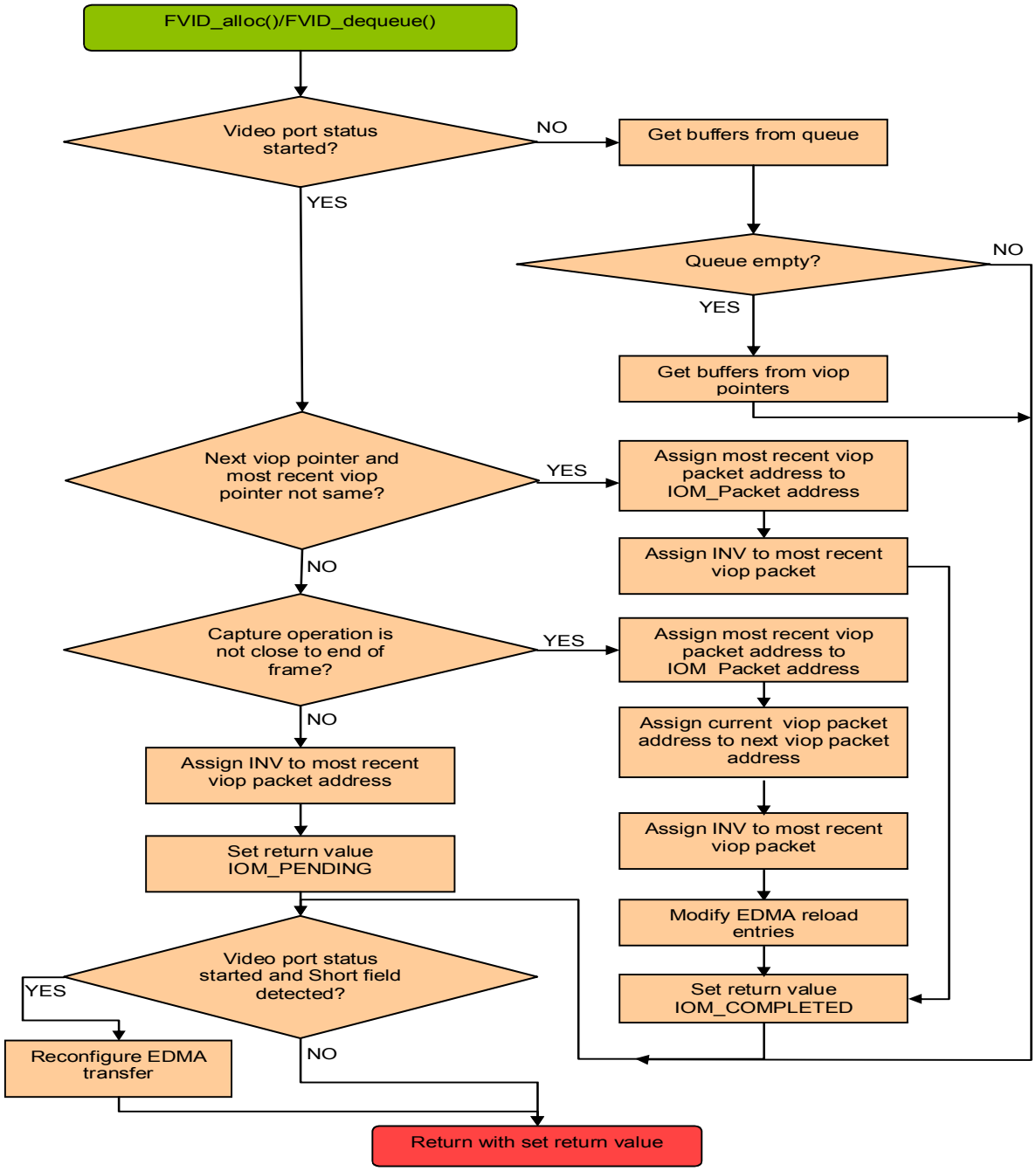

Figure 24 FVID_alloc()/FVID_dequeue() flow diagram

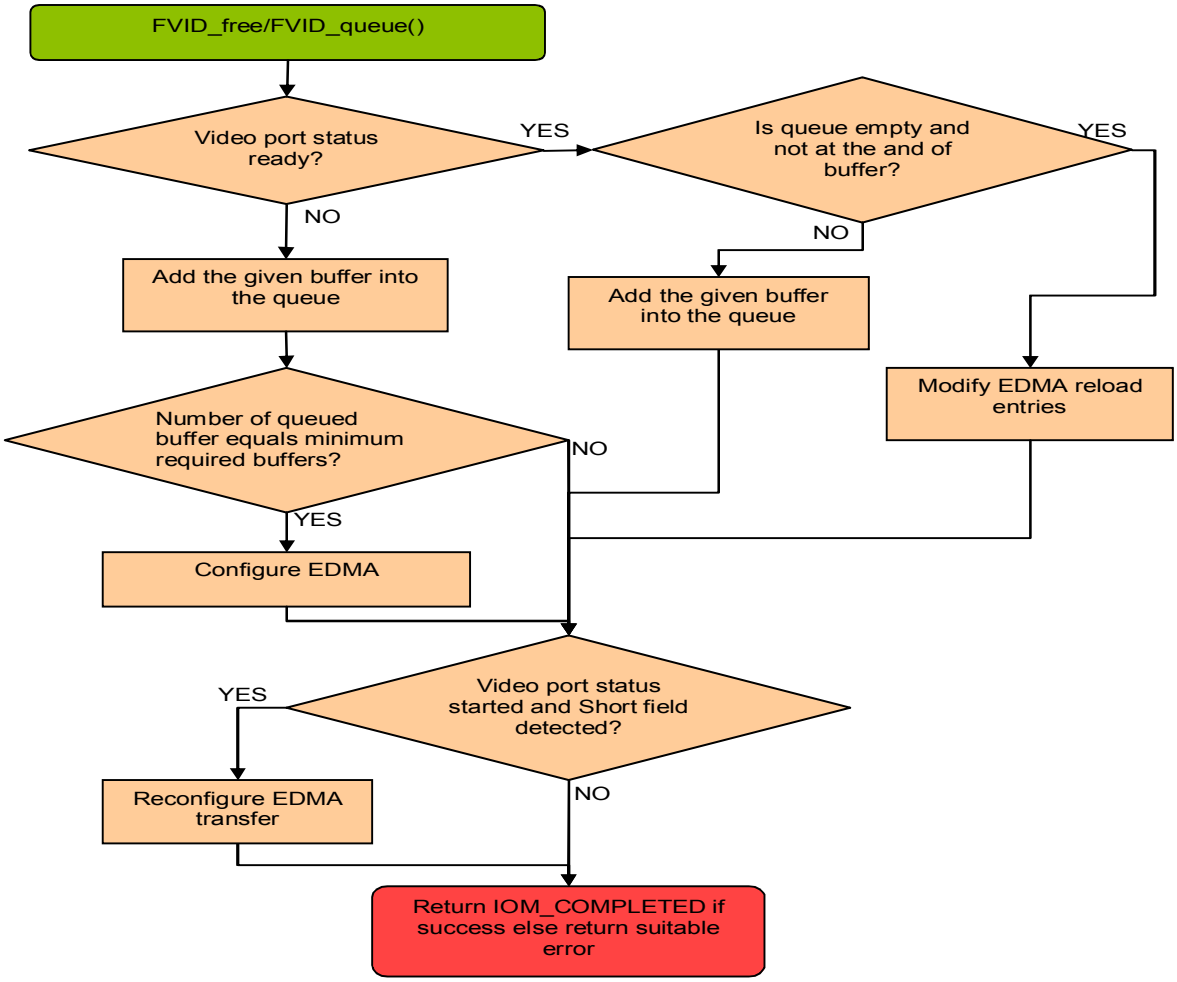

Figure 25 FVID_free()/FVID_queue() flow diagram

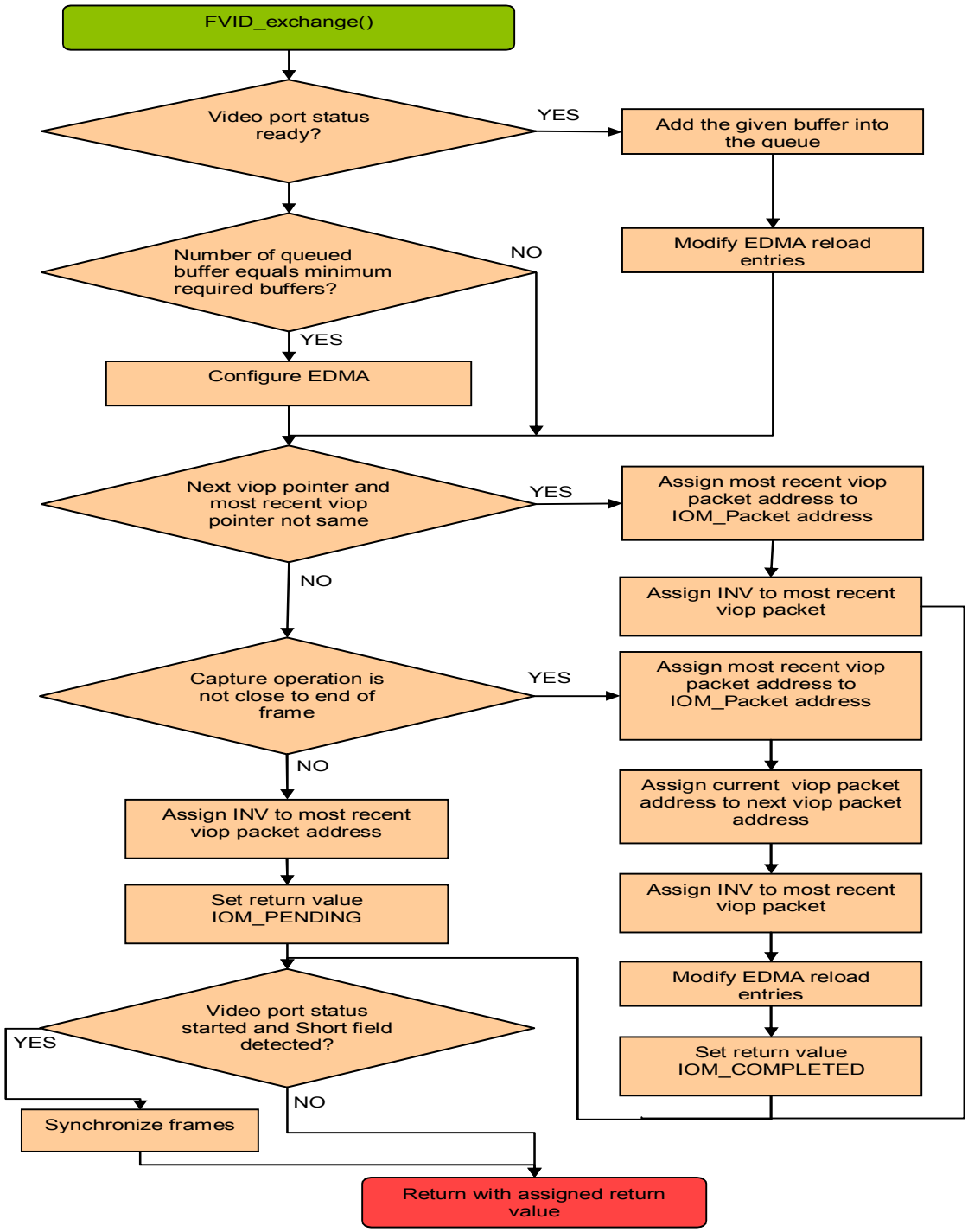

Figure 26 FVID_exchange() flow diagram

2.2.3.1.6 captureEdmaISR

captureEdmaISR is called when Capture EDMA interrupts are generated. These EDMA interrupts are generated after EDMA transfer completion for single frame capture. captureEdmaISR updates EDMA parameters of EDMA channels and manages frame buffers.

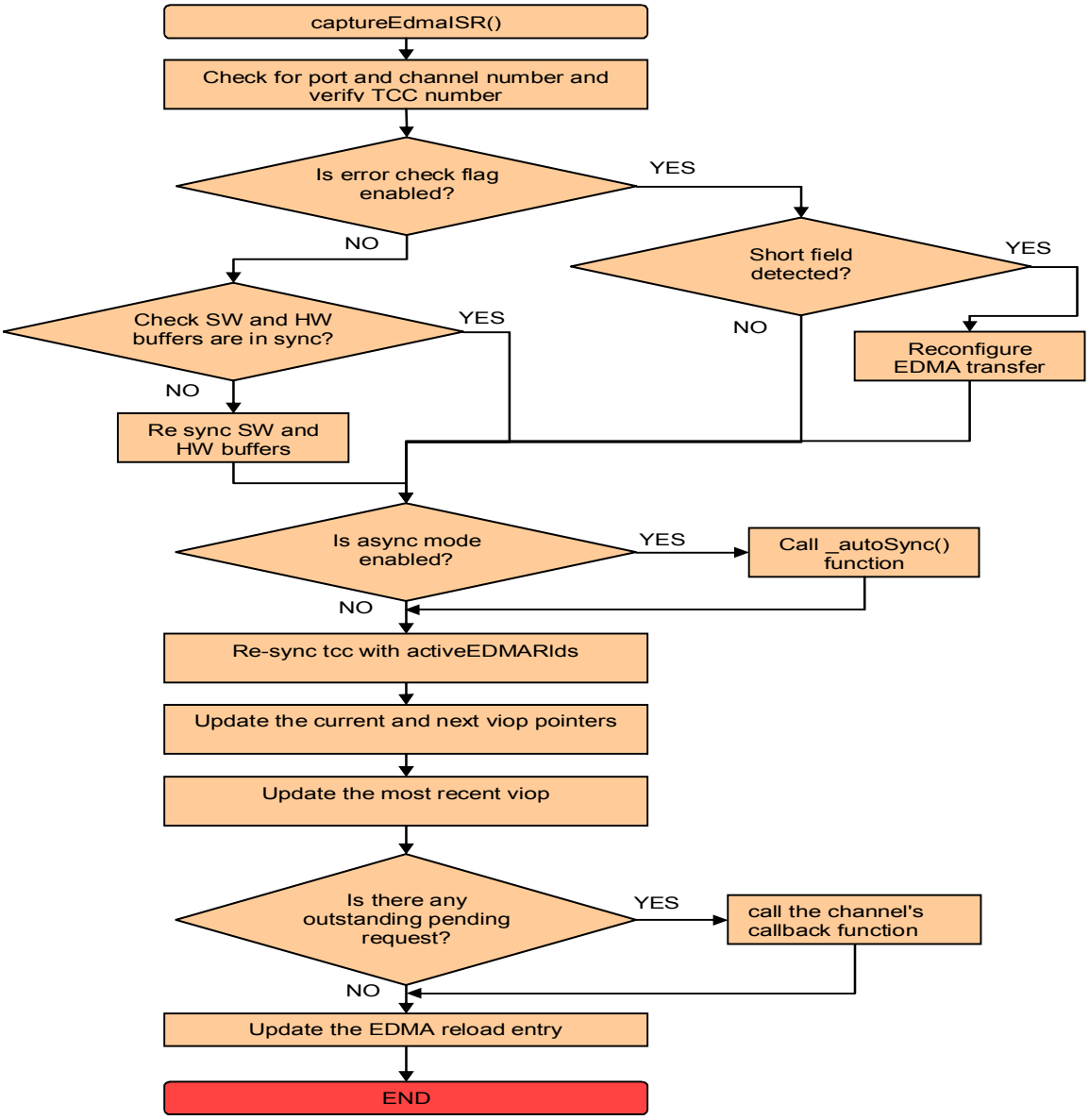

Figure 27 captureEdmaISR () flow diagram

2.2.3.1.7 captureISR

captureISR is called when VPORT generates any interrupt. If application has enabled any video port interrupt then on interrupt generation captureISR is called.

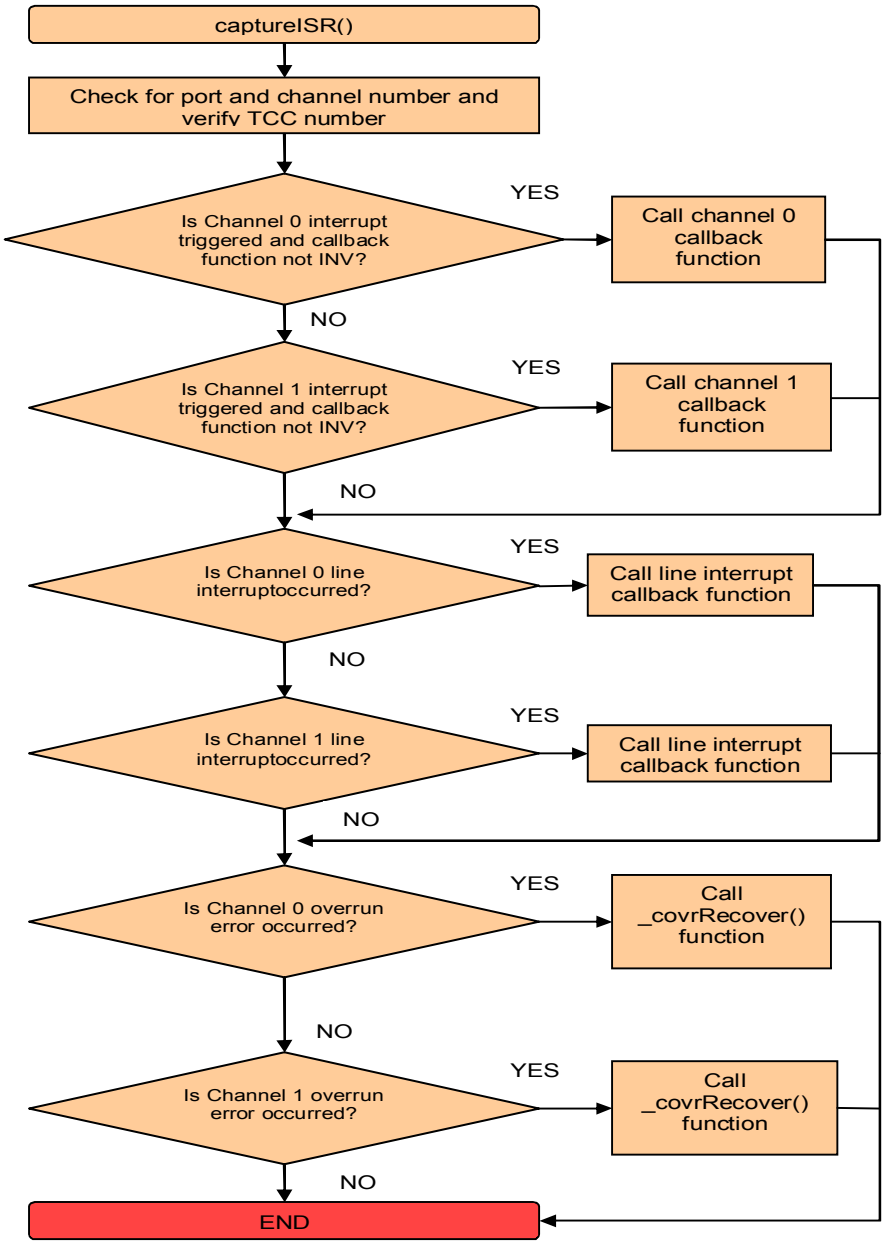

Figure 28 captureISR () flow diagram

2.2.3.2 Display Driver

2.2.3.2.1 mdBindDev

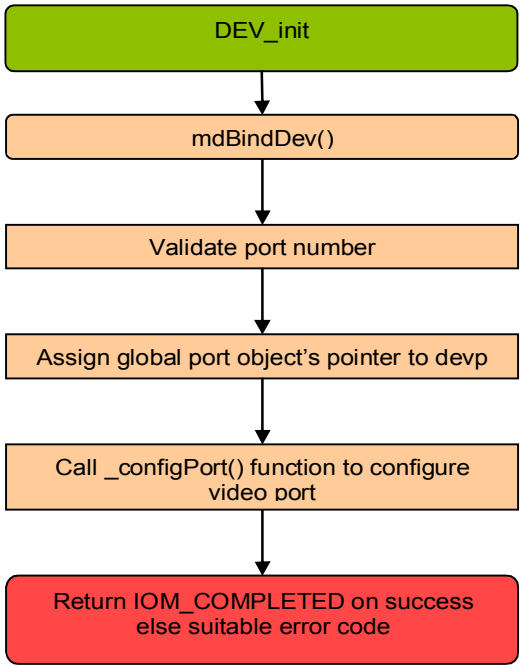

Figure 29 mdBindDev () flow diagram

2.2.3.2.2 mdCreateChan

When application calls FVID_create(), mdCreateChan() will get called internally. Application has to pass proper channel configuration parameters as an argument to FVID_create().

In Legacy mode this API is compatible to usage of DM642 FVID create API. The 'no. of buffers' ('numFrmBufs' member) in the channel parameters is the flag used to identify this mode. The value must be greater than or equal to 3 for the driver to work in this mode.

In this legacy mode, this call will allocate frame buffers (Number of Buffers provided as parameter), initialize EDMA channels and configure video port registers. These frame buffers will be used later for FVID exchange and other APIs.

In the normal mode (where the 'numFrmBufs' in the channel parameters is 0), the driver will not allocate frame buffers for FVID exchange and other APIs. Applications have to create buffers for this purpose. It is suggested that applications should use the APIs provided with driver for frame buffer allocation purpose.

As part of channel creation, VPORT driver will request 3 EDMA channels and 12 (3*4) EDMA parameter tables. Here, Y, Cb and Cr each data transfer will use 1 EDMA channel and 4 EDMA parameter tables.

After successful operation of create channel, handle is returned to the application. This handle is used afterwards to make calls for different channel operations.

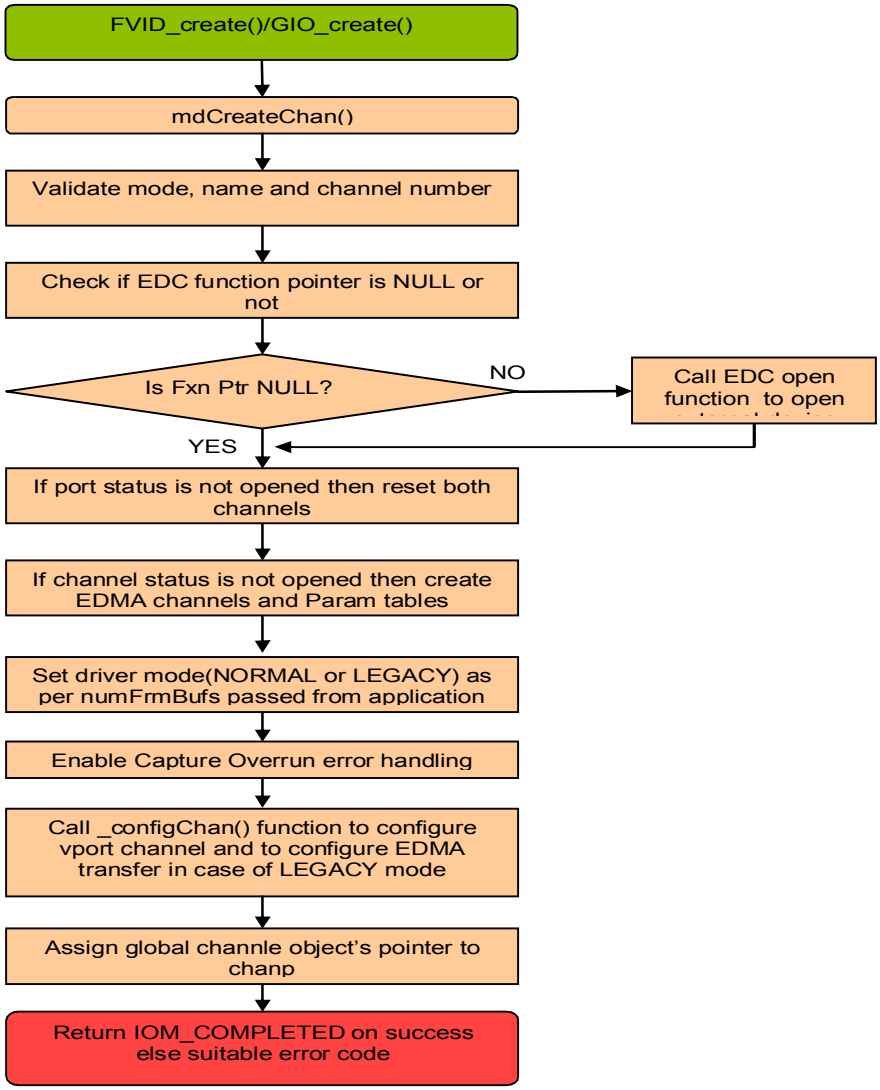

Figure 30 mdCreateChan () flow diagram

2.2.3.2.3 *mdDeleteChan*

mdDeleteChan will release all resources created for Display channel and make channel status closed.

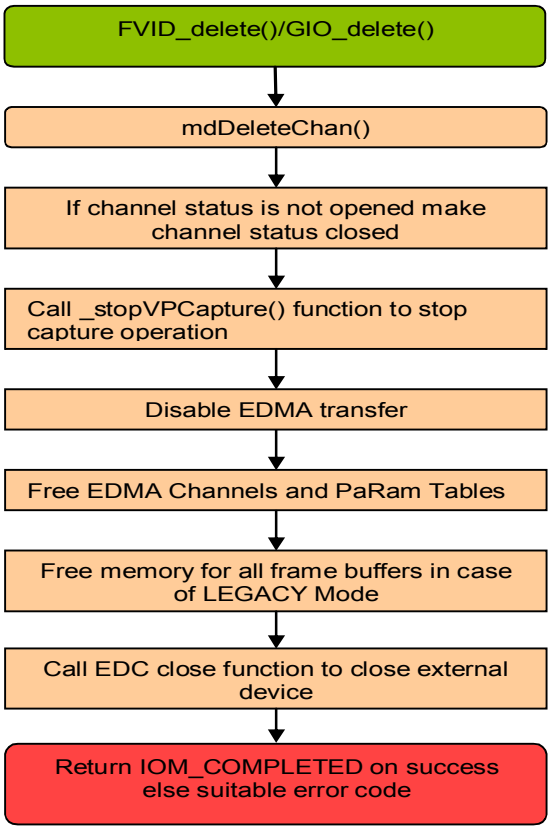

Figure 31 mdDeleteChan () flow diagram

2.2.3.2.4 *mdControlChan*

mdControlChan is used to call different ioctls supported by driver. External device configuration is also done by calling mdControlChan with EDC specific control commands.

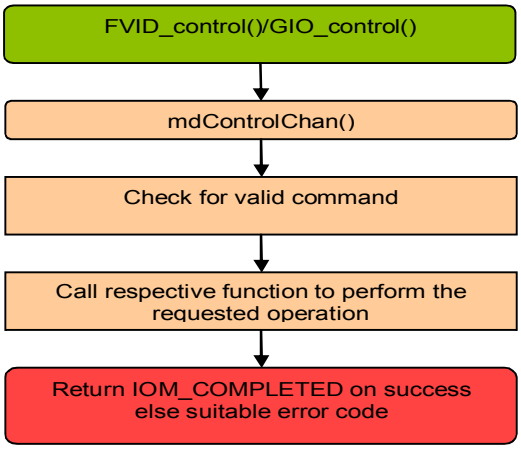

Figure 32 mdControlChan () flow diagram

mdControlChan is supporting following ioctls:

1) VPORT_CMD_START
2) VPORT_CMD_STOP
3) VPORT_CMD_SET_VINTCB
4) VPORT_CMD_DUND_RECOVER
5) VPORTDIS_CMD_ASYNC_MODE_ENABLE
6) VPORTDIS_CMD_ASYNC_MODE_DISABLE
7) VPORTDIS_CMD_ASYNC_MODE_RESET_FRAMECT
8) VPORT_CMD_CONFIG_CHAN
9) VPORT_CMD_GET_NUM_IORQST_PENDING
10) VPORT_CMD_GET_PARAMS
11) FVID_ALLOC_BUFFER
12) FVID_FREE_BUFFER

The detailed flow diagrams of above ioctls are as follow:

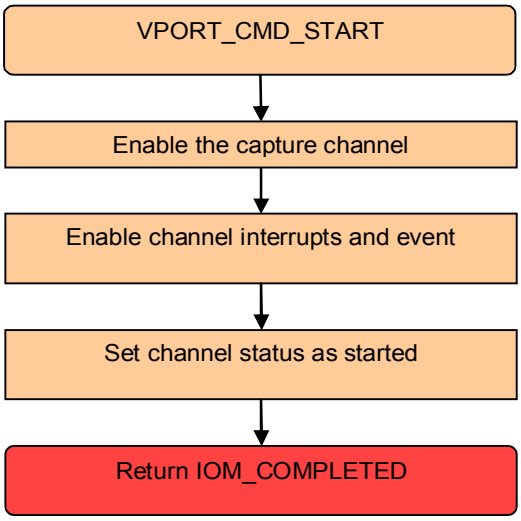

Figure 33 VPORT_CMD_START flow diagram

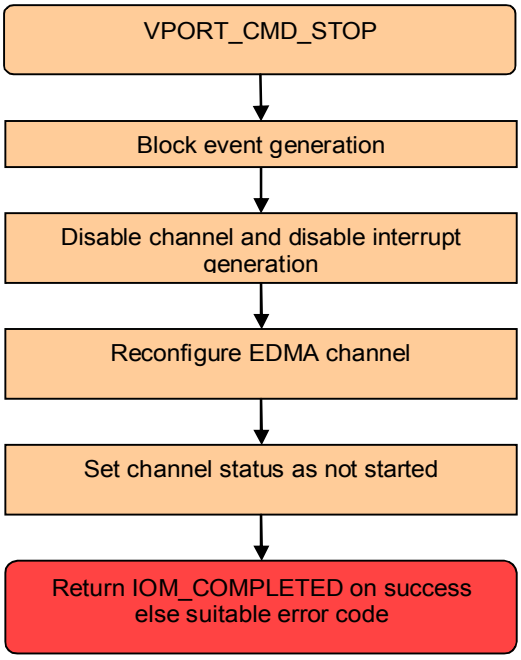

Figure 34 VPORT_CMD_STOP flow diagram

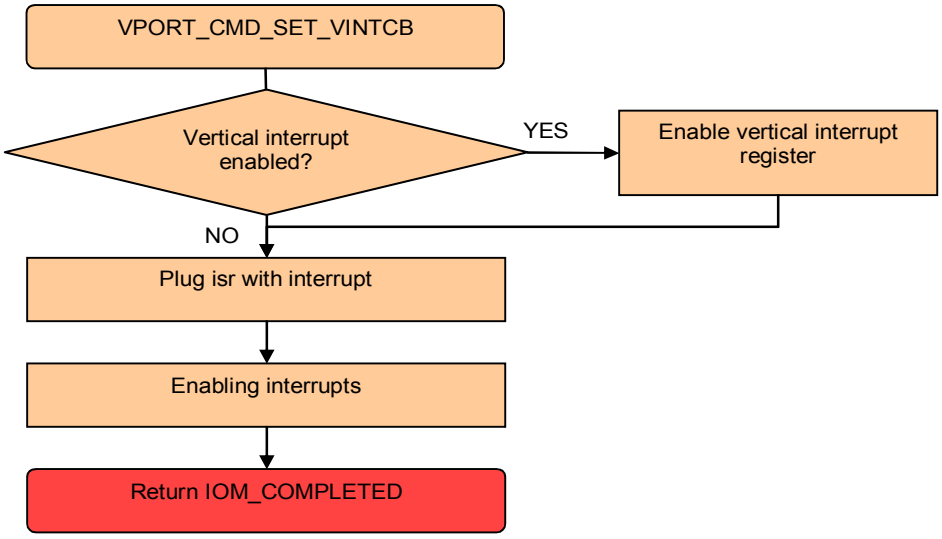

Figure 35 VPORT_CMD_SET_VINTCB flow diagram

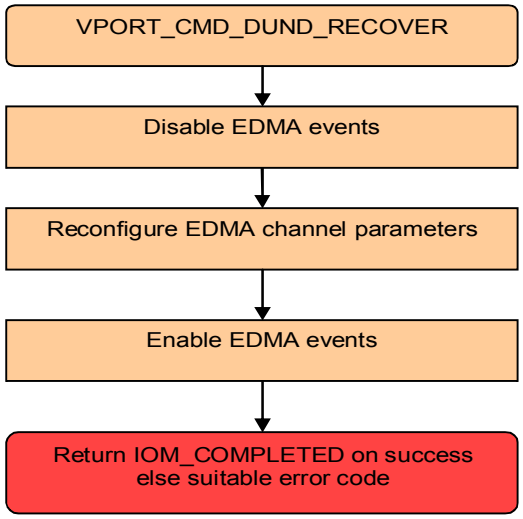

Figure 36 VPORT_CMD_DUND_RECOVER flow diagram

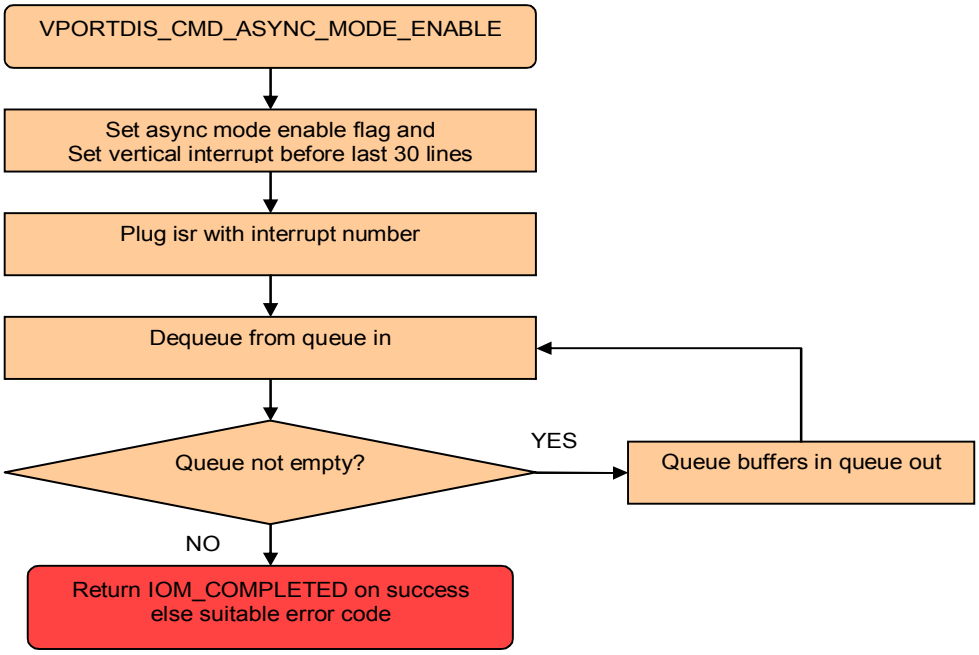

Figure 37 VPORTDIS_CMD_ASYNC_MODE_ENABLE flow diagram

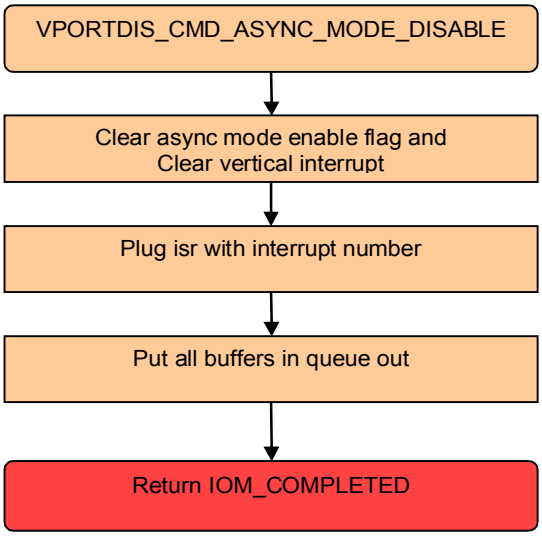

Figure 38 VPORTDIS_CMD_ASYNC_MODE_DISABLE flow diagram

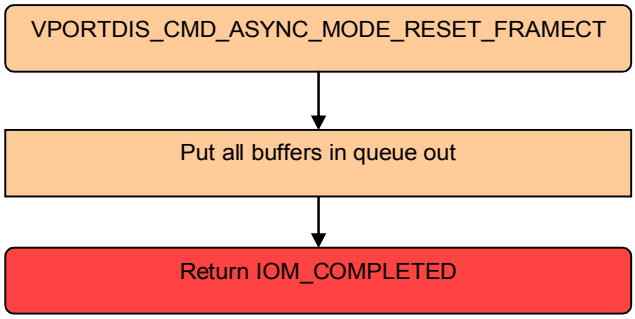

Figure 39 VPORTDIS_CMD_ASYNC_MODE_RESET_FRAMECT flow diagram

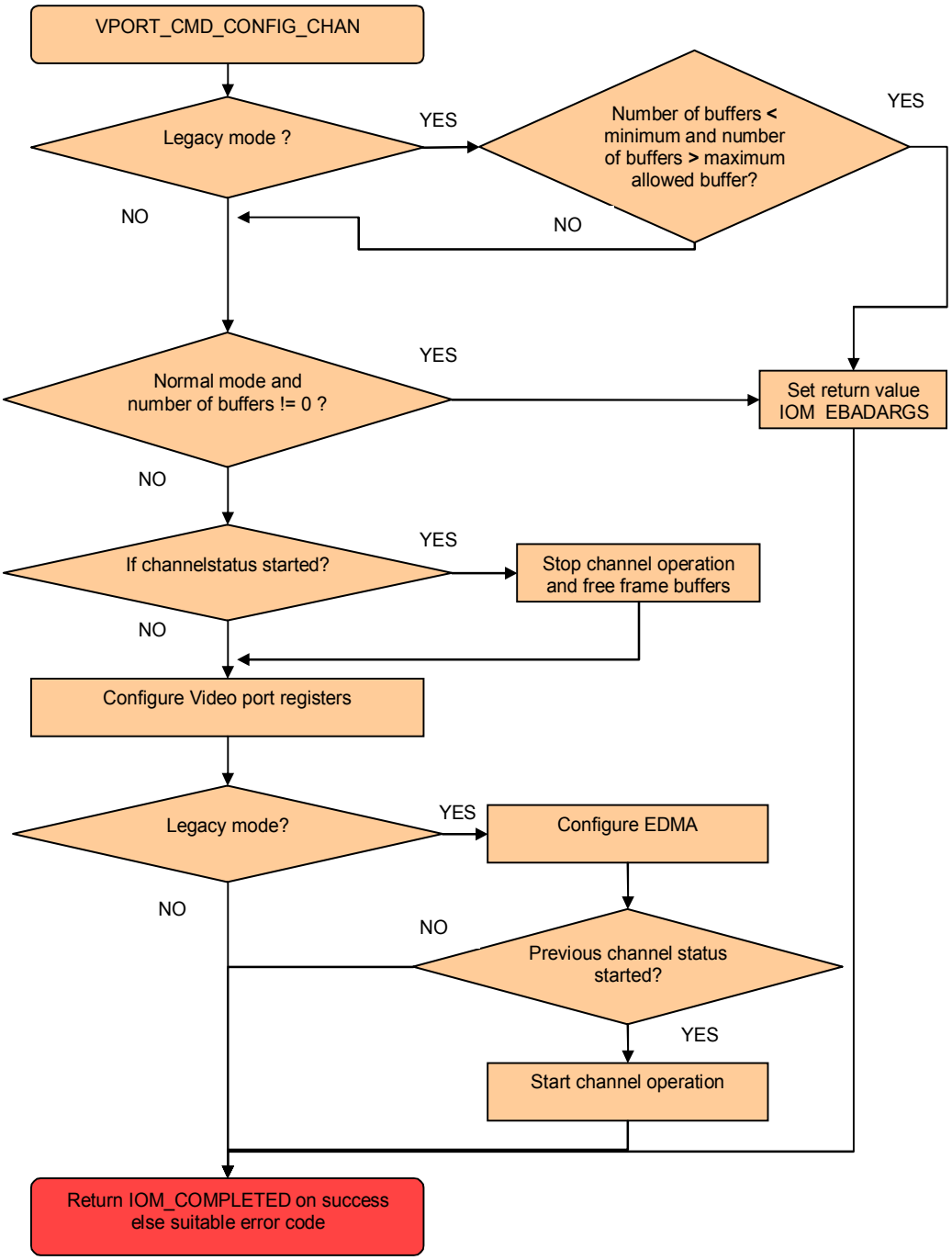

Figure 40 VPORT_CMD_CONFIG_CHAN flow diagram

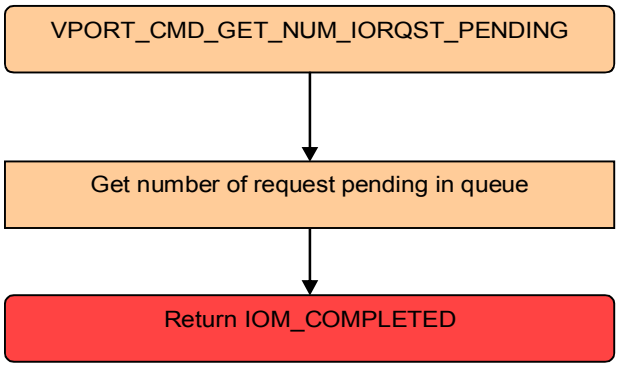

Figure 41 VPORT_CMD_GET_NUM_IORQST_PENDING flow diagram

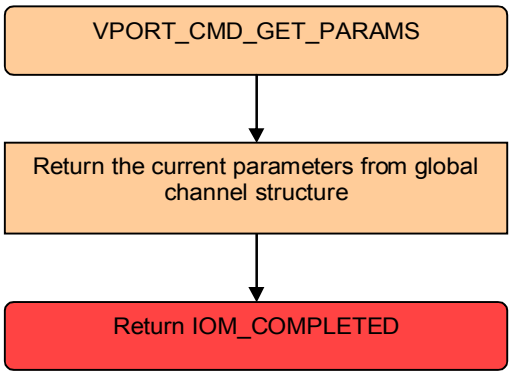

Figure 42 VPORT_CMD_GET_PARAMS flow diagram

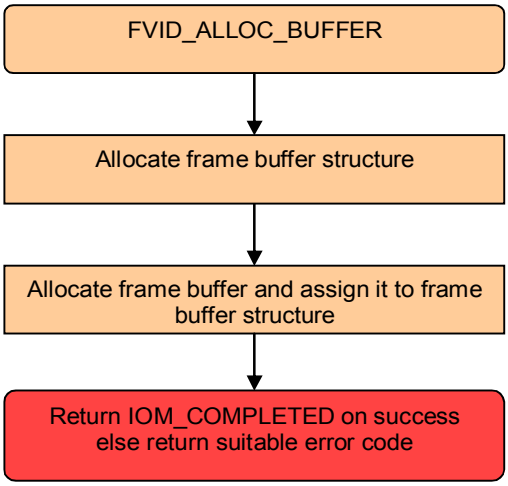

Figure 43 FVID_ALLOC_BUFFER flow diagram

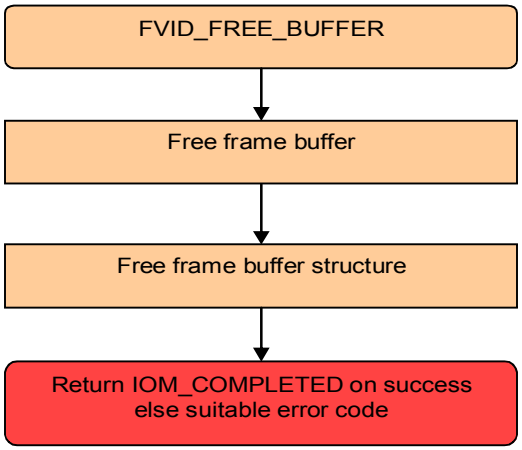

Figure 44 FVID_FREE_BUFFER flow diagram

2.2.3.2.5 mdSubmitChan

This function handles buffer management and exchange between video driver and application. In case of NORMAL mode, FVID_allocBuffer and FVID_freeBuffer calls can be used to allocate and free frame buffers.

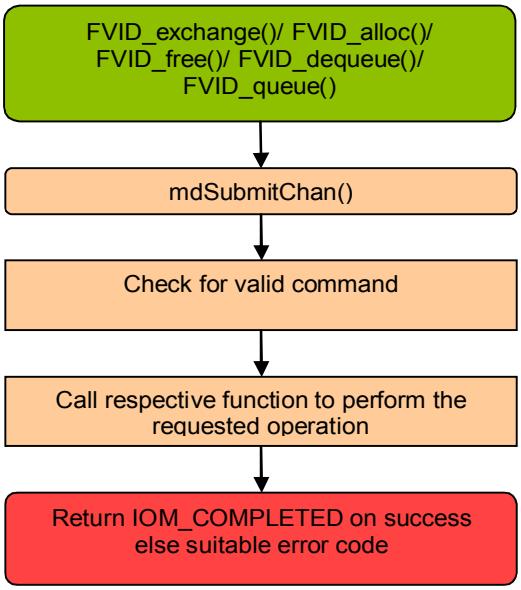

Figure 45 mdSubmitChan () flow diagram

mdSubmitChan is called by three different FVID layer calls for LEGACY Mode:

1) FVID_alloc

2) FVID_exchange

3) FVID_free

mdSubmitChan is called by three different FVID layer calls for LEGACY Mode:

1) FVID_dequeue

2) FVID_exchange

3) FVID_queue

The detailed flow diagram of above FVID layer calls are as follow:

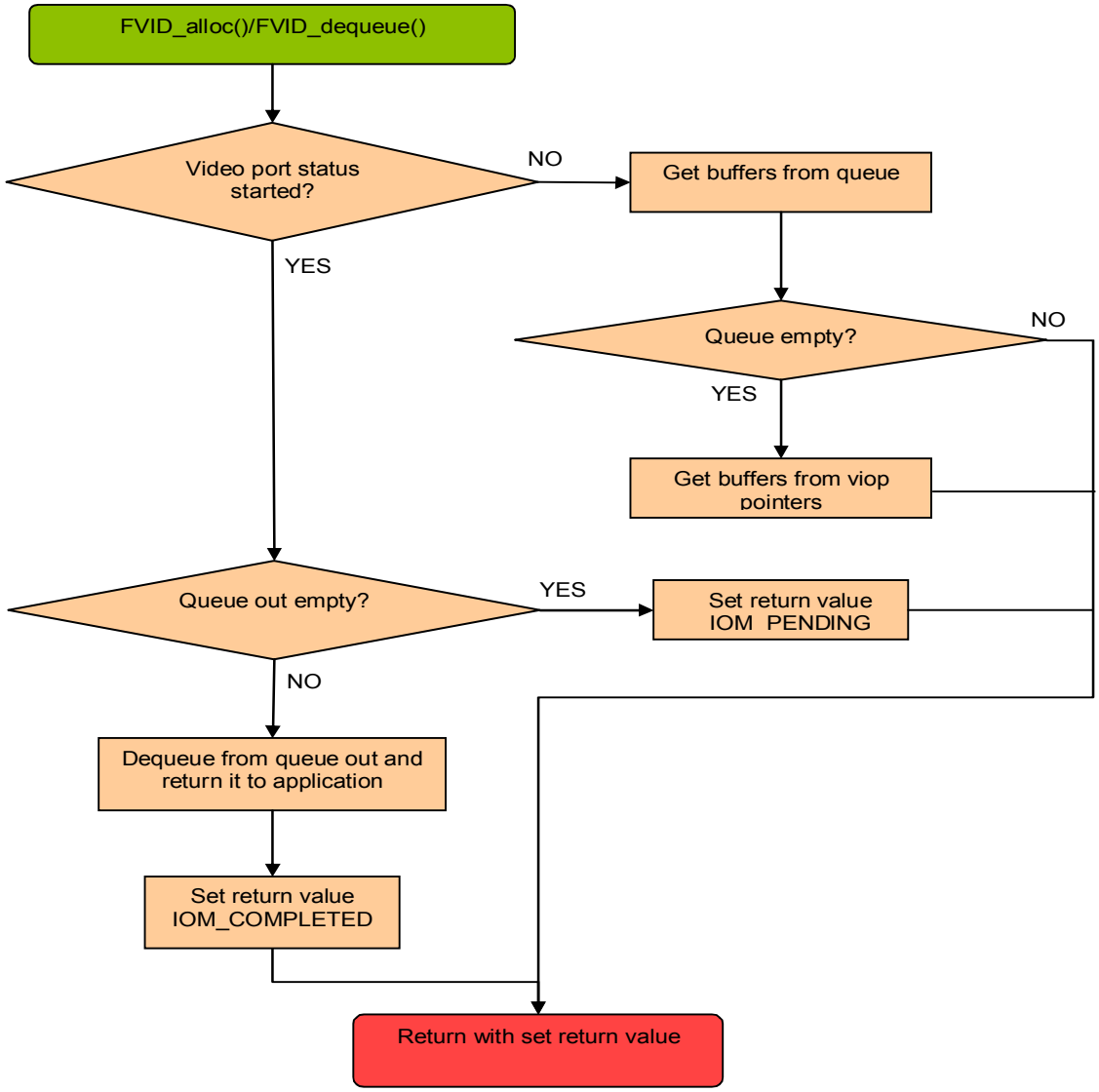

Figure 46 FVID_alloc()/FVID_dequeue() flow diagram

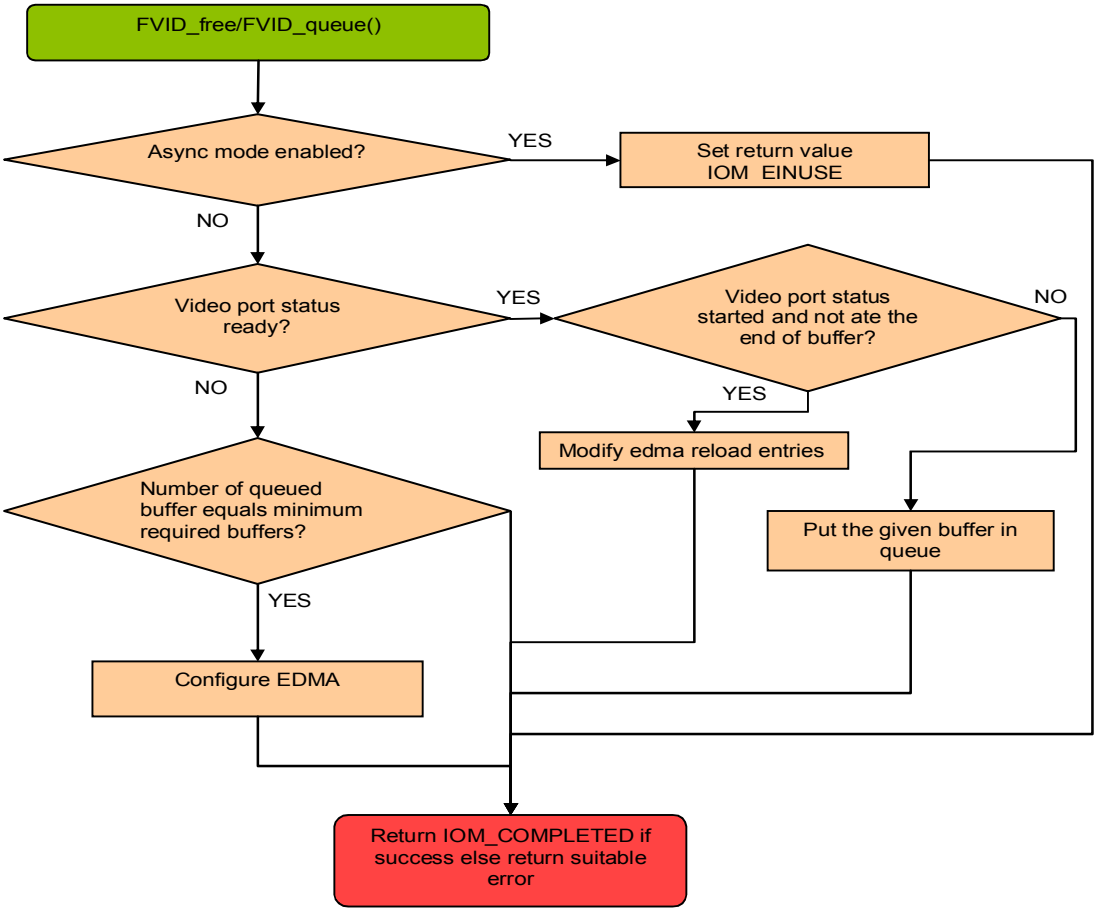

Figure 47 FVID_free()/FVID_queue() flow diagram

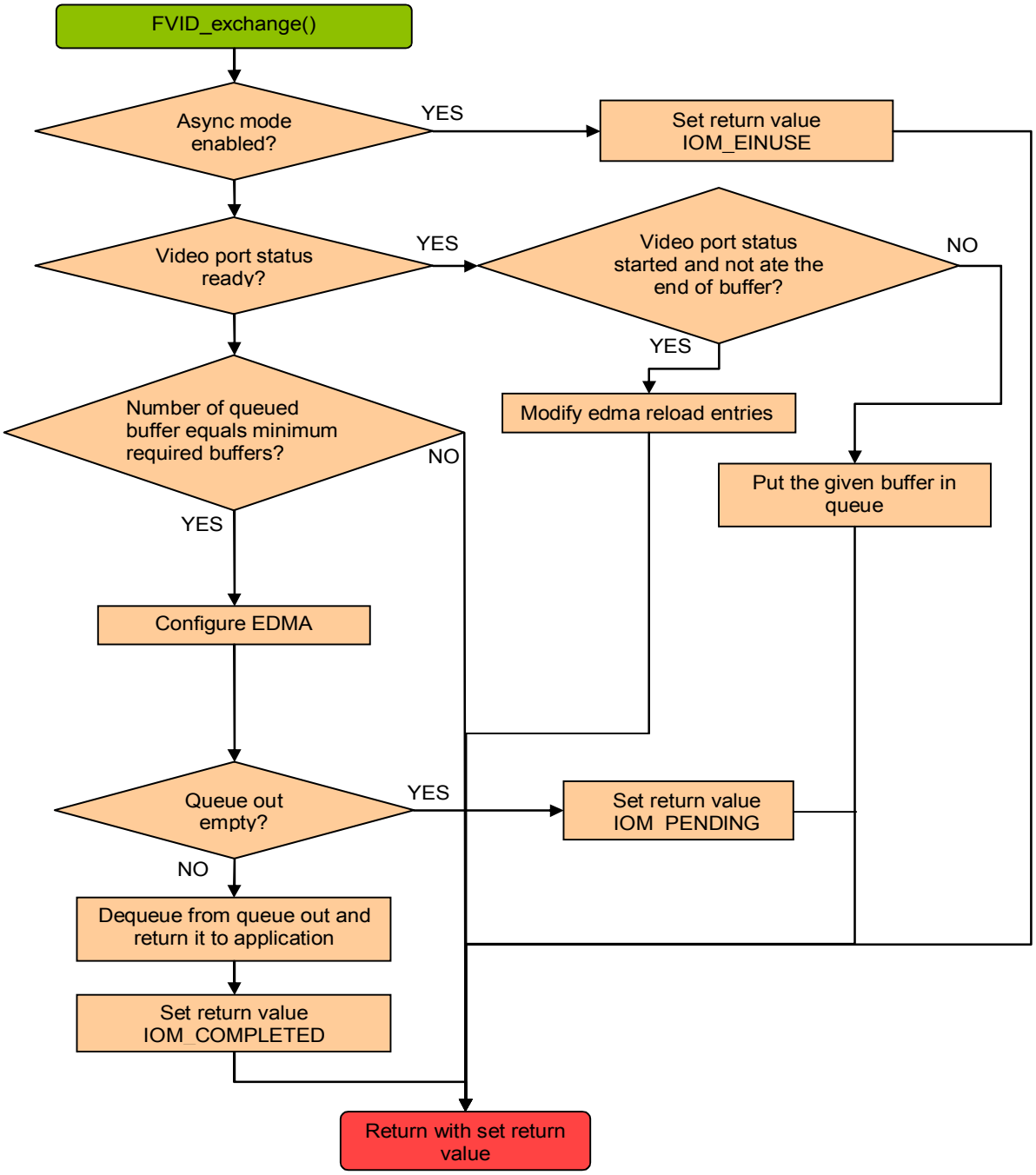

Figure 48 FVID_exchange() flow diagram

2.2.3.2.6 displayEdmaISR

displayEdmaISR is called when Display EDMA interrupts are generated. These EDMA interrupts are generated after EDMA transfer completion for single frame display. displayEdmaISR updates EDMA parameters of EDMA channels and manages frame buffers.

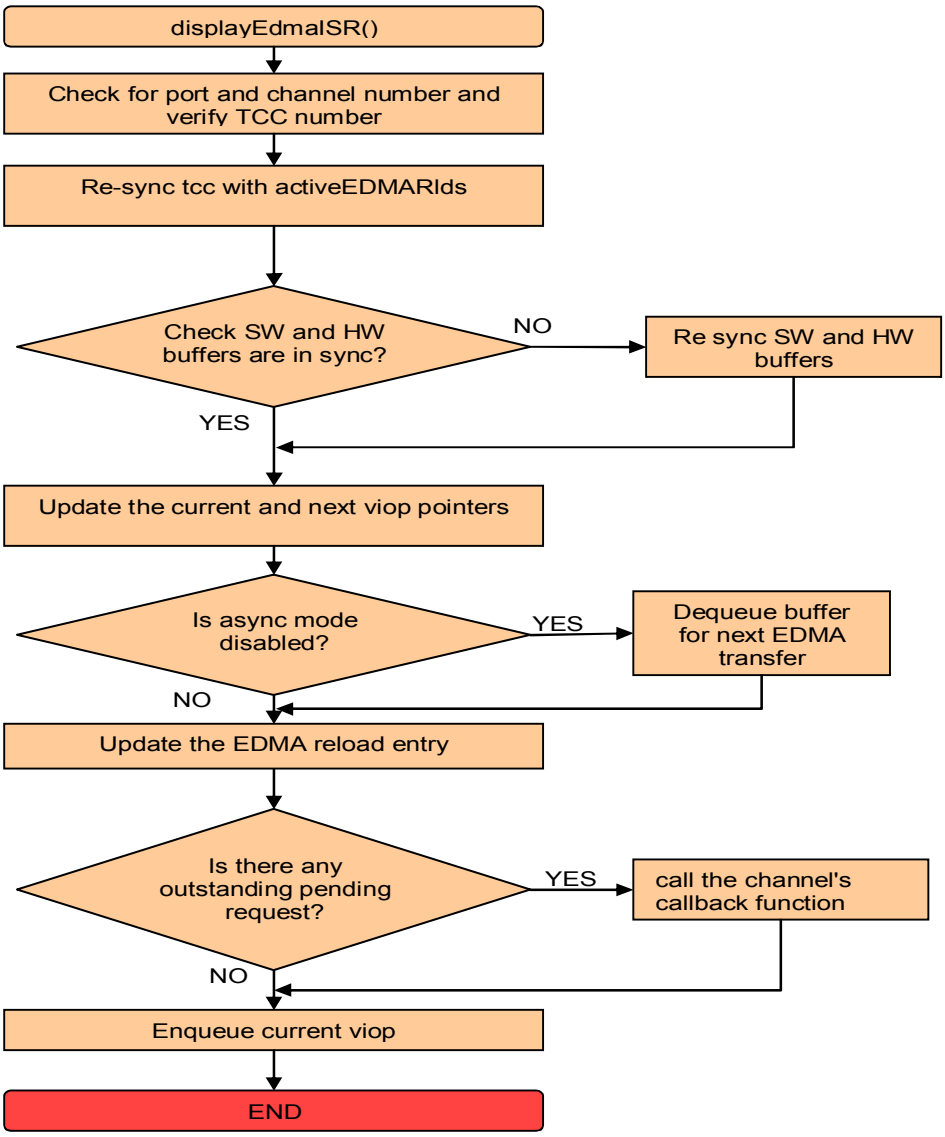

Figure 49 displayEdmaISR () flow diagram

2.2.3.2.7 displayISR

displayISR is called when VPORT generates any interrupt. If application has enabled any video port interrupt then on interrupt generation displayISR is called.

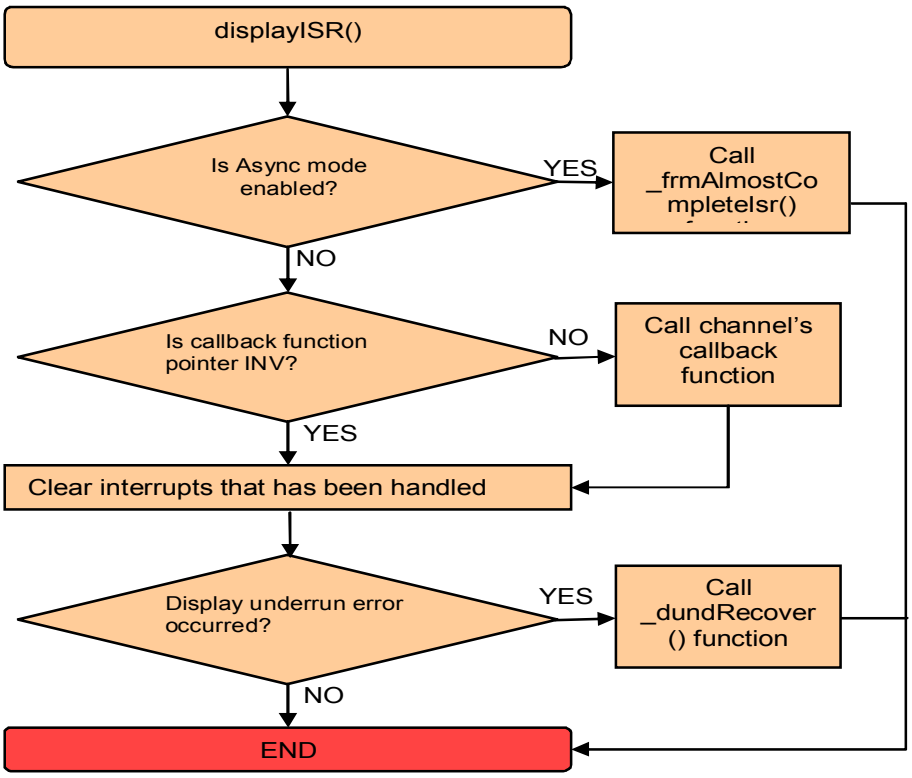

Figure 50 displayISR () flow diagram

3 APPENDIX A – IOCTL commands

The application can perform the following IOCTL on the channel.

S.No	IOCTL Command	Description
1	VPORT_CMD_CONFIG_CHAN	Configure video channel
2	VPORT_CMD_START	Start video port operation
3	VPORT_CMD_STOP	Stop video port operation
4	VPORT_CMD_SET_VINTCB	Set video channel error call-back function
5	VPORT_CMD_DUND_RECOVER	Force the recovery of display under-run
6	VPORT_CMD_COVR_RECOVER	Force the recovery of capture over-run
7	VPORT_CMD_GET_NUM_IORQST_PENDING	Getting the number of pending request at driver level
8	VPORTDIS_CMD_ASYNC_MODE_ENABLE/ VPORTDIS_CMD_ASYNC_MODE_DISABLE/ VPORTDIS_CMD_ASYNC_MODE_RESET_FRAMECT	Configurations to get an interrupt after display of specified lines. Supported by Display driver only.
9	VPORT_CMD_GET_PARAMS	Get the current channel configuration of driver
10	VPORTCAP_CMD_SET_LINE_INT	Command to enable call-back after specified number of lines. Supported by Capture driver only.
11	VPORTCAP_CMD_GET_NUMLINES_CAPTURED	Command to get number of lines captured by EDMA. Supported by Capture driver only.
12	Default IOCTL (further has specific commands)	To Configure the external encoders and decoders. Interface will depend on the encoder/decoder