LSP 1.20 DaVinci Linux CCDC Driver

User's Guide

Literature Number: SPRUEP7
March 2008

Texas Instruments

Contents

List of Figures

LSP 1.20 DaVinci Linux CCDC Driver

This guide introduces the DaVinci Linux CCDC Driver by providing a brief overview of the driver and specifics concerning its use within a hardware/software environment. For LSP 1.20, the CCDC Driver is supported on the following EVMs: DM644x, DM355.

1 Overview

The CCDC Driver provides the following functional services:

- The CCDC Driver captures YUV data using TVP5146 decoder.
- The CCDC Driver captures RAW data using MT9T001/MT9T031 sensor.
- THe CCDC Driver is V4L2-standard compliance.
- The CCDC Driver is a loadable module.

1.1 System Requirements

The CCDC Driver is supported on platforms characterized by the following software and hardware requirements:

- Software
  - Monta Vista Linux 2.6.10
- Hardware
  - DaVinci EVM board for YUV mode capture
  - MT9T001/MT9T031 Image Sensor with DV-EVM connectivity for raw mode capture

1.2 Modules

This section provides the overview of the modules in CCDC Driver.

The CCDC Driver is sub-divided into following vertical modules:

- *Initialization*
  This module handles all the initialization activities including driver registration, driver un-registration, channel creation, and channel deletion.
- *Configuration and Control*
  This module is responsible for handling all configurations of the CCD Controller and capture device.
- *Interrupt Handling*
  This is the interrupt handler for the CCDC driver. It handles interrupts generated by CCD hardware for image capture notification.
- *Buffer Management*
  This module handles all buffer management activities including buffer creation, maintaining open buffers, and mapping/un-mapping of physical buffer to/from applications memory area.

1.3 Layers

The CCDC driver is divided into two horizontal layers:

- *VPFE/V4L2 Layer*

This layer of the CCDC driver provides Linux operating system specific routines. It is part of the V4L2 driver. In IOCTL system calls, the driver supports various commands that are part of the V4L2 standard. Applications use these commands and structures to interact with the driver. This layer also handles buffer management and interrupts.

- *Hardware Configuration Layer*

  This layer is responsible for all hardware-related configurations. It provides routines to set/get the value of registers and configures the following devices:

- CCD Controller through memory-mapped registers
- TVP5146 Decoder using I2C interface (for YUV data capture)
- MT9T001/MT9T031 CMOS Sensor using I2C interface (for raw (Bayer) data capture)

2 Installation Guide

This section discusses the CCDC Driver installation, what software and hardware components are available, and how to make these components available in order to complete a successful installation of the driver.

2.1 List of Installable Components

The patch containing DaVinci CCDC-V4L2 driver, MT9T001/MT9T031 driver, TVP5146 driver code, Makefile, and Kconfig files.

2.2 Component Folder

The CCDC Driver can be found in the following directory after final installation into the system:

`montavista/pro/devkit/lsp/ti-davinci/drivers/media/video/davinci`

2.3 Development Tools

Install the following tools, in the order listed below, to set up the development environment:

- MVL401, version 2.6.10
- MontaVista Linux Toolchain - arm_v5t_le-

2.4 Build

This section describes the steps required to build the driver.

2.4.1 Build Options

This driver does not require any specific build options at this time.

2.4.2 Build Steps

Access to the CCDC Driver is provided through the `/dev/video0` device file. The `/dev/video0` device file is a character device that provides read/write access.

Use the following steps to enable CCDC Driver support in the system:

Step 1. Choose your default kernel configuration by entering the command: `make davinci_xxxx_defconfig`.

Step 2. Choose the driver-specific kernel configuration options by entering the command: `make menuconfig`.

Step 3. Select the *Device Drivers* option and then, select *Multimedia Devices*. In *Multimedia Devices* choose the <*> *Video for Linux* option. Finally, choose the *Video for Linux* option.

Step 4. At this point, the driver can be built as static or as a module.

a. To make a static build, choose the <*> *Davinci Video Capture* option. By enabling this option, TVP5146 and MT9T001 driver support will also be added into the kernel. Note that

enabling the MT9T001 option will work for both MT9T031 as well as MT9T001 image sensors.

b. To build as a module, choose the *<M> Davinci Video Capture* option. By enabling this option, TVP5146 and MT9T001 driver support will also be added into the kernel. Note that enabling the MT9T001 option will work for both MT9T031 as well as MT9T001 image sensors. The following .ko files are created for the driver: `davinci_capture.ko`, `mt9t001.ko`, and `tvp5146.ko`.

Step 5. Save your kernel configration options and build the kernel by entering the following command: `make uImage modules`.

2.5 Steps to Load/Unload the CCDC Driver

To run the driver using the dynamically loadable modules, copy the modules (.ko files) to the target filesystem. Execute the following commands to load the CCDC Driver modules:

- `insmod tvp5146.ko`
- `insmod mt9t001.ko`
- `insmod davinci_capture.ko device_type=<device number>`
  where device = 0 (for MT9T001), 1 (for TVP5146), or 2 (for MT9T031). For example, when using the MT9T031 as the raw capture device, enter the following:
  `insmod davinci_capture.ko device_type=2`

Execute the following commands to unload the CCDC Driver modules:

- `rmmod davinci_capture.ko`
- `rmmod mt9t001.ko`
- `rmmod tvp5146.ko`

When the CCDC driver is built as a static module (built using <*>), configure the `device_type` value in the bootargs of the boot loader as given below:

- To use MT9T001 for raw capture, set device_type=0 in bootargs. For example:
  ```
  setenv bootargs mem=120M console=ttyS0,115200n8 root=/dev/nfs
  nfsroot=192.168.8.49:/home/wofbrkdir/filesys rw noinitrd ip=dhcp
  davinci_capture.device_type=0
  ```
- Use the device_type=1 for TVP5146
- Use the device_type=2 for MT9T031

3 Run-Time Interfaces/Integration Guide

This section discusses the CCDC Driver run-time interfaces that comprise the API classification and usage scenarios, and the API specification, itself, in association with its data types and structure definitions.

3.1 Header Files

The CCDC driver uses the following proprietary header files (available under the `include/media/davinci` directory):

- davinci_vpfe.h (application include)
- ccdc_dm355.h (application include for DM355)
- ccdc_davinci.h (application include for DM6446)
- mt9t001.h (already included in davinci_vpfe.h)
- tvp5146.h (already included in davinci_vpfe.h)

3.2 Symbolic Constants and Enumerated Data Types

For all symbolic constants and enumerated data types required to be used by the application, refer to the header files mentioned in Section 3.1.

3.3 Data Structures

For all data structures required to be used by the application, refer to the header files mentioned in Section 3.1.

3.4 API Classification

This section introduces the Application Programming Interface (API) for the CCDC Driver.

3.4.1 Configuration

This section contains all the CCDC Driver APIs that allow you to specify the desired configuration parameters. Section 3.6.2 elaborates on each such mechanism in greater detail.

3.4.2 Creation

This section contains all the CCDC Driver APIs that are intended for use in component creation. The term creation is indicative of possible need to allocate system resources, typically memory.

IOCTLs like VIDIOC_REQBUFS will dynamically allocate buffers as per buffer count given by application. Section 3.6.2 elaborates on each such mechanism in greater detail.

3.4.3 Initialization

This contains the CCDC Driver APIs that are intended for use in component initialization.

The API open is used to initialize the CCDC configuration structure.

3.4.4 Control

This section contains CCDC Driver APIs are intended for run-time control. Section 3.6.2 elaborates on each such mechanism in greater detail. The IOCTLs VIDIOC_STREAMON and VIDIOC_STREAMOFF are used to start/stop capture operation.

3.4.5 Data Acquisition

This section lists all CCDC Driver APIs that help to output data out of CCDC Driver Termination.

The IOCTLs IOCTL VIDIOC_G_FMT, VIDIOC_G_STD, VIDIOC_G_INPUT, and VPFE_CMD_G_MT9T001_PARAMS are used to get the status of the driver or capture device.

The API close is used to free all the resources that are being acquired at the time of initialization and creation.

3.4.6 Termination

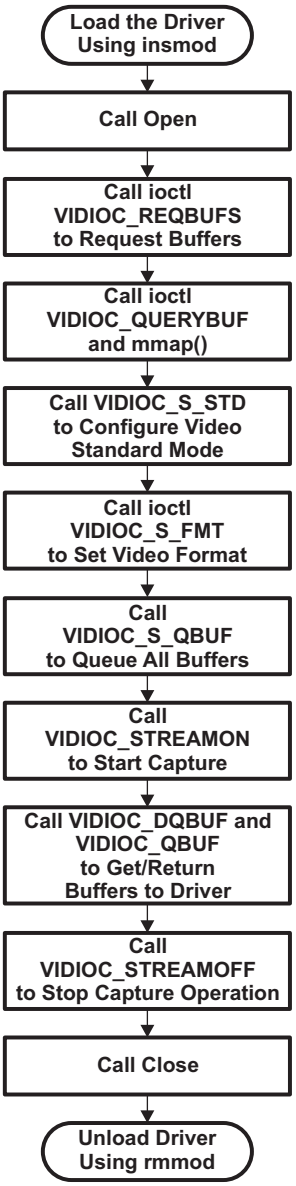
This section contains the CCDC Driver APIs that help in gracefully terminating the deployed CCDC Driver run-time entities.

The API close is used to free all the resources that are being acquired at the time of initialization and creation.

3.5 API Usage Scenarios/Integration Example

Figure 1 shows the usage scenario for the CCDC Driver.

Figure 1. Function Flow Diagram



3.6 API Specification

3.6.1 Naming Conventions

The naming conventions are followed as per the GA license.

3.6.2 CCDC Driver Functions

The detailed descriptions of APIs discussed above are described below, in alphabetical order.

API close

Prototype	`int close(int fd)`
Description	Closes the logical channel associated with fd.
Arguments	
	<arg1> int fd : File Descriptor
	<arg2> NA
	<arg3> NA
Return Value	On success, 0 is returned; on error -1 and the errno variable is set appropriately.
Calling Constraints	None
Example	`close(fd)`
Side Effects	None
See Also	None
Errors	None

IOCTL VIDIOC_ENUM_FMT

Prototype `int ioctl(int fd, int command, struct v4l2_fmtdesc *argp)`

Description Used to enumerate image formats.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_fmtdesc *fmt = pointer to format descriptor structure.

Return Value On success, 0 is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_ENUM_FMT, fmt);`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_G_FMT

Prototype `int ioctl(int fd, int command, struct v4l2_fmtdesc *argp)`

Description Used to enumerate image formats.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_fmtdesc *fmt = pointer to format descriptor structure.

Return Value On success, 0 is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_ENUM_FMT, fmt);`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_S_FMT

Prototype `int ioctl(int fd, int command, struct v4l2_format *arg)`

Description Used to change the current format parameters.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_format * arg : pointer to v4l2_format structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_S_FMT, config)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_TRY_FMT

Prototype `int ioctl(int fd, int command, struct v4l2_format *arg)`

Description Used to change the current format parameters.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_format * arg : pointer to v4l2_format structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_TRY_FMT, fmt)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_ENUMSTD

Prototype `int ioctl(int fd, int command, struct v4l2_standard *arg)`

Description Used to enumerate supported video standards.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_ standard * arg : pointer to v4l2_ standard structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_ENUMSTD, std)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_S_STD

Prototype `int ioctl(int fd, int command, struct v4l2_standard *arg)`

Description Used to select the video standard of the current input.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_ standard * arg : pointer to v4l2_ standard structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_S_STD, std)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_G_CTRL

Prototype `int ioctl(int fd, int command, struct v4l2_control *arg)`

Description Used to get the value of a control.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_ control * arg : pointer to v4l2_ control structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_G_CTRL, ctrl)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_S_CTRL

Prototype `int ioctl(int fd, int command, struct v4l2_control *arg)`

Description Used to set the value of a control.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_ control * arg : pointer to v4l2_ control structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_S_CTRL, ctrl)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_QUERYCTRL

Prototype `int ioctl(int fd, int command, struct v4l2_queryctrl *arg)`

Description Used to enumerate controls.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_ queryctrl * arg : pointer to v4l2_ queryctrl structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_QUERYCTRL, ctrl)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_STREAMON

Prototype `int ioctl(int fd, int command, int *arg)`

Description Used to enumerate controls.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	int *argp = pointer to integer for desired buffer or stream type.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_STREAMON, NULL)`

Side Effects None

See Also None

Errors None

IOCTL VPFE_CMD_S_MT9T001_PARAMS

Prototype `int ioctl(int fd, int command, struct MT9T001_params *argp)`

Description Used to configure MT9T001/MT9T031 parameters.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct MT9T001_params *argp = pointer to mt9t001_params structure containing configuration parameters.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VPFE_CMD_S_MT9T001_PARAMS, params)`

Side Effects None

See Also None

Errors None

IOCTL VPFE_CMD_G_MT9T001_PARAMS

Prototype `int ioctl(int fd, int command, struct MT9T001_params *argp)`

Description Used to get MT9T001/MT9T031 configuration parameters.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct MT9T001_params *argp = pointer to mt9t001_params structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VPFE_CMD_G_MT9T001_PARAMS, params)`

Side Effects None

See Also None

Errors None

IOCTL VPFE_CMD_CONFIG_CCDC_RAW

Prototype `int ioctl(int fd, int command, struct ccdc_params_raw *argp)`

Description Used to configure CCDC for raw mode.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct ccdc_params_raw *argp = pointer to ccdc_params_raw structure containing configuration parameters.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VPFE_CMD_CONFIG_CCDC_RAW, params)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_REQBUFS

Prototype `int ioctl(int fd, int command, struct v4l2_requestbuffers *argp)`

Description Used to used to allocate the buffers.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct v4l2_requestbuffers *argp = pointer to v4l2_requestbuffers structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_REQBUFS, reqbufs)`

Side Effects None

See Also None

Errors None

IOCTL VPFE_CMD_CONFIG_TVP5146

Prototype `int ioctl(int fd, int command, struct tvp5146_params *argp)`

Description Used to configure the TVP5146 decoder.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	int request
<arg3>	struct tvp5146_params *argp = pointer to tvp5146_params structure.

Return Value On success, buffer size is returned; on error, -1 is returned and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VPFE_CMD_CONFIG_TVP5146, params)`

Side Effects None

See Also None

Errors None

API MMAP

Prototype `void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)`

Description Maps the frame buffers allocated by the PREV module in kernel space to user space.

Arguments

Arg1	void *start
Arg2	size_t length
Arg 3	int prot
Arg 4	int flags Only MAP_SHARED is supported
Arg 5	int fd
Arg 6	off_t offset

Return Value Zero, on success, or -1, if an error occurred.

Calling Constraints Zero on success, or -1 if an error occurred

Example `mmap(0, image_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, offset);`

Side Effects Zero on success, or -1 if an error occurred

See Also Zero on success, or -1 if an error occurred

Errors Zero on success, or -1 if an error occurred

API MUNMAP

Prototype `Int munmap(void *start, int length)`

Description Unmaps the frame buffers that were previously mapped to user space using mmap().

Arguments

Arg1	void *start
Arg2	size_t length
Arg 3	NA

Return Value Zero, on success, or -1, if an error occurred.

Calling Constraints None

Example `munmap(offset, image_size)`

Side Effects None

See Also None

Errors None

API open

Prototype `int open(char *name, int mode)`

Description Opens the driver in the mode specified in the last parameter.

Arguments

Arg1	char *name
Arg2	Int mode
Arg 3	None

Return Value File descriptor, on success, or -1, if an error occurred.

Calling Constraints None

Example `open(“/dev/davinci_aew”, O_RDWR)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_QUERYCAP

Prototype `int ioctl(int fd, int command, struct v4l2_capability *argp`

Description Used to query device capabilities.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_QUERYCAP
<arg3>	struct v4l2_capability *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_QUERYCAP, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_G_STD

Prototype `int ioctl(int fd, int command, struct v4l2_std_id *argp)`

Description Used to query the video standard of the current input.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_G_STD
<arg3>	struct v4l2_std_id *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_G_STD, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_ENUMINPUT

Prototype `int ioctl(int fd, int command, struct v4l2_input *argp)`

Description Used to enumerate video inputs.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_ENUMINPUT
<arg3>	struct v4l2_input *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_ENUMINPUT, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_G_INPUT

Prototype `int ioctl(int fd, int command, int *argp)`

Description Used to query current video input.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_G_INPUT
<arg3>	int *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_G_INPUT, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_S_INPUT

Prototype `int ioctl(int fd, int command, int *argp)`

Description Used to select current video input.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_S_INPUT
<arg3>	int *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_S_INPUT, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_G_CROP

Prototype `int ioctl(int fd, int command, struct v4l2_crop *argp)`

Description Used to get the current cropping rectangle.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_G_CROP
<arg3>	struct v4l2_crop *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_G_CROP, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_S_CROP

Prototype `int ioctl(int fd, int command, struct v4l2_crop *argp)`

Description Used to set the current cropping rectangle.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_S_CROP
<arg3>	struct v4l2_crop *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_S_CROP, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_CROPCRAP

Prototype `int ioctl(int fd, int command, struct v4l2_cropcap *argp)`

Description Used to get the information about the video cropping and scaling abilities.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_CROPCRAP
<arg3>	struct v4l2_cropcap *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_CROPCRAP, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_QUERYSTD

Prototype `int ioctl(int fd, int command, v4l2_std_id *argp)`

Description Used to sense the video standard received by the current input.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_QUERYSTD
<arg3>	v4l2_std_id *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_QUERYSTD, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_G_PARM

Prototype `int ioctl(int fd, int command, v4l2_streamparm *argp)`

Description Used to set streaming parameters.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_G_PARM
<arg3>	v4l2_streamparm *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_G_PARM, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_G_PRIORITY

Prototype `int ioctl(int fd, int command, enum v4l2_priority *argp)`

Description Used to query the access priority associated with a file descriptor.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_G_PRIORITY
<arg3>	enum v4l2_priority *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_G_PRIORITY, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_S_PRIORITY

Prototype `int ioctl(int fd, int command, enum v4l2_priority  *argp)`

Description Used to request the access priority associated with a file descriptor.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_S_PRIORITY
<arg3>	enum v4l2_priority *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_S_PRIORITY, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_QUERYBUF

Prototype `int ioctl(int fd, int command, struct v4l2_buffer *argp)`

Description Used to query the status of a buffer.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_QUERYBUF
<arg3>	struct v4l2_buffer *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_QUERYBUF, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_QBUF

Prototype `int ioctl(int fd, int command, struct v4l2_buffer *argp)`

Description Used to exchange a buffer with the driver. Used for queuing purposes.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_QBUF
<arg3>	struct v4l2_buffer *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_QBUF, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_DQBUF

Prototype `int ioctl(int fd, int command, struct v4l2_buffer*argp)`

Description Used to exchange a buffer with the driver. Used for dequeuing purposes.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_QBUF
<arg3>	struct v4l2_buffer *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_DQBUF, argp)`

Side Effects None

See Also None

Errors None

IOCTL VIDIOC_STREAMOFF

Prototype `int ioctl(int fd, int command, const int*argp)`

Description Used to stop the streaming.

Arguments

<arg1>	int fd : File Descriptor
<arg2>	VIDIOC_STREAMOFF
<arg3>	Const int *argp

Return Value On success, 0; on error, -1 and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, VIDIOC_STREAMOFF, argp)`

Side Effects None

See Also None

Errors None

IOCTL VPFE_CMD_CONFIG_CCDC_YCBCR

Prototype	`int ioctl(int fd, int command, ccdc_params_ycbcr*argp)`
Description	Used to configure CCDC for YUV mode.
Arguments	
	<arg1> int fd : File Descriptor
	<arg2> VPFE_CMD_CONFIG_CCDC_YCBCR
	<arg3> struct ccdc_params_ycbcr *
Return Value	On success, 0; on error, -1 and the errno variable is set appropriately.
Calling Constraints	None
Example	`ioctl(fd, VPFE_CMD_CONFIG_CCDC_YCBCR, argp)`
Side Effects	None
See Also	None
Errors	None

3.7 API Usage Recommendations

This section provides recommendations on how to use the provided APIs for achieving the best results on different aspects: performance, overall system stability, and balance, etc.

- Optimum performance can be achieved if line offsets are 256 bytes aligned.

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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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