LSP 1.20 DaVinci Linux Previewer Driver

User's Guide

Literature Number: SPRUFE9
March 2008

TEXAS INSTRUMENTS

User's Guide
SPRUFE9–March 2008

LSP 1.20 DaVinci Linux Previewer Driver

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

1 Overview

The Previewer Driver provides the following functional services:

- The Previewer driver is a loadable module.
- The Previewer driver supports input image in bayer pattern.
- The Previewer driver supports input from SDRAM or DDRAM.
- The Previewer driver converts input image in bayer pattern to image in YCbCr 4:2:2 format.

1.1 System Requirements

The driver is supported on DaVinci EVM Boards with Monta Vista Linux 2.6.10 software.

1.2 Modules

The Previewer 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 handles all configurations and previewing functionality of the driver.
- *Interrupt Handling*

  This is the interrupt handler for the driver. It handles interrupts generated by Previewer hardware for various events.
- *Buffer Management*

  This module handles all buffer management activities including buffer creation, maintaining open buffers, and mapping/un-mapping of the physical buffer to/from the applications memory area.

1.3 Layers

The Previewer driver is divided into two horizontal layers:

- Functional Layer: implements all the functionalities and application interface.
- Hardware Configuration Layer: contains functions to configure the hardware. These functions are used by the functional layer for configuration and control.

2 Installation Guide

This section discusses installation of the Previewer Driver, 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

A patch containing Previewer Driver code, Makefile, and Kconfig files.

2.2 Component Folder

The Previewer Driver can be found in the following directory after final installation into the system:
`montavista/pro/devkit/lsp/ti-davinci/drivers/char`

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 device driver.

2.4.1 Build Options

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

2.4.2 Build Steps

Access to the Previewer Driver is provided through the `/dev/davinci_previewer` device file. The `/dev/davinci_previewer` device file is a character device that provide read/write access.

Use the following steps to enable the Previewer 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. From the screen that appears next, select the *Character Devices* 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 Previewer Driver Support* option.

b. To build as a module, choose the *<M>* *DaVinci Previewer Driver Support* option.

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 Previewer Driver

To load the driver module using dynamically loadable modules, copy the modules (.ko files) to the target filesystem.

Execute the following command to load the Previewer Driver:

- `insmod davinci_previewer_driver.ko`

Execute the following command to unload the Previewer Driver:

- `rmmod davinci_previewer_driver.ko`

3 Run-Time Interfaces/Integration Guide

This section discusses the Previewer 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 Symbolic Constants and Enumerated Data Types

This section summarizes all the symbolic constants specified as #define macros and/or enumerated C data types. Described in Table 1 alongside the macro or enumeration is the symbolic constant name and description. It is typical to classify the data types into logical groups and list them in alphabetical order for ease of use.

Table 1. Symbolic Constants and Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
Macro	MAX_BUFFERS	Maximum numbers of buffers that can be allocated is restricted to 8.
Macro	PREV_BUF_IN	Indicates that buffer asked is an input buffer. Its value is represented by 0
Macro	PREV_BUF_OUT	Indicates that buffer asked is an output buffer. Its value is represented by 1
Macro	PREV_INPUT_FORMATTER	Enabled the support of Input formatter component.
Macro	PREV_INVERSE_ALAW	Enables support of Inverse A-Law
Macro	PREV_HORZ_MEDIAN_FILTER	Enabled Support of Horizontal Median Filter
Macro	PREV_NOISE_FILTER	Enabled Support of Noise Filter
Macro	PREV_CFA	Enabled Support of CFA Interpolation Filter
Macro	PREV_GAMMA	Enabled Support of Gamma Correction
Macro	PREV_LUMA_ENHANCE	Enabled Support of Luminance Enhance
Macro	PREV_CHROMA_SUPPRESS	Enabled Support of Chrominance Suppression
Macro	PREV_DARK_FRAME_SUBTRACT	Enabled Support of Dark Frame Subtract.
Macro	PREV_LENS_SHADING	Enabled Support of Lens shading.
Macro	PREV_INWIDTH_8BIT	Indicates that the input image's pixel width is 8 bits.
Macro	PREV_INWIDTH_10BIT	Indicates that the input image's pixel width is 10 bits.
Macro	LUMA_TABLE_SIZE	Size of the Luminance Enhancement table. Its value is 128.
Macro	GAMMA_TABLE_SIZE	Size of the Gamma Correction Coefficient's table. Its value is 1024.
Macro	CFA_COEFF_TABLE_SIZE	Size of the CFA Interpolation Coefficient's table. Its value is 576.
Macro	NOISE_FILTER_TABLE_SIZE	Size of the Noise Filter Coefficients table. Its value is 256.
Macro	MAX_IMAGE_WIDTH	Maximum image width supported by the driver. Its value is 1280.
Macro	MAX_IMAGE_HEIGHT	Maximum image height supported by the driver. Its value is 1920.
enum prev_pixorder	PREV_PIXORDER_YCBYCR	Indicates pixel output format is Y0, Cb0, Y1 and Cr0 from lower address to higher
enum prev_pixorder	PREV_PIXORDER_YCRYCB	Indicates pixel output format is Y0, Cr0, Y1 and Cb0 from lower address to higher
enum prev_pixorder	PREV_PIXORDER_CBYCRY	Indicates pixel output format is Cb0, Y0, Cr0 and Y1 from lower address to higher
enum prev_pixorder	PREV_PIXORDER_CRYCBY	Indicates pixel output format is Cr0, Y0, Cb0 and Y1 from lower address to higher
Macro	PREV_DARK_FRAME_CAPTURE	Enable support of dark frame capture

3.2 Data Structures

This section summarizes all user-visible data structures elements pertaining to the Previewer Driver run-time interfaces.

1. Buffer-allocation structure:

```
struct prev_reqbufs
{
    int buf_type;      /* type of frame buffer */
    int size;             /* size of the frame buffer to be allocated */
```

```
    int count;         /* number of frame buffer to be allocated */
}
```

2. Buffer-status query structure

```
struct prev_buffer
{
    unsigned char index;        /* index number, 0 -> N-1 */
    unsigned char buf_type;     /* buffer type, input or output */
    unsigned long offset;       /* physical address of the buffer used in the
                                                     mmap() system call */
    unsigned short size;        /* size of the buffer */
};
```

3. Configuration and parameter structure

```
struct prev_params
{
unsigned short                 features;              /* Set of features enabled */
prev_size_params            size_params;           /* size parameters */
prev_white_balance             white_balance_params;  /* white balancing parameters */
prev_black_adjst            black_adjst_params;    /* black adjustment parameters */
prev_rgbblending            regblending_params;    /* rgb blending parameters */
prev_rgb2ycbcr_coeffs       rgb2ycbcr_params;      /* rgb to ycbcr parameters */
unsigned char               sample_rate;           /* down sampling rate for averager */
short                       hmf_threshold;         /* horizontal median filter threshold */
prev_cfa_coeffs             cfa_coeffs;            /* CFA coefficients */
prev_gamma_coeffs           gamma_coeffs;          /* gamma coefficients */
prev_noiseflt_coeffs        nf_coeffs;             /* noise filter coefficients */
unsigned int                luma_enhance[128];     /* luma enhancement coeffs*/
prev_chroma_spr             chroma_suppress_params;/* chroma suppression coefficients  */
void                       *dark_frame_addr;       /* dark frame address for dark frame
subtract */
unsigned short              dark_frame_pitch;      /* line offset for dark frame */
unsigned char               lens_shading_sift;     /* number of bits to be shifted for lens
shading */
prev_pixorder               pix_fmt;                  /* output pixel format */
int                         contrast               /* Contrast */
int                         brightness             /* Brightness */
};
```

4. Size-parameter structure

```
struct prev_size_params
{
            unsigned int     hstart;   /* starting pixel */
            unsigned int     hstart;   /* starting line */
            unsigned int     hsize;    /* width of input image */
            unsigned int     vsize;    /* height of input image */
            unsigned char    pixsize;  /* pixel size of the image in terms of bits */
            unsigned short      in_pitch;  /* input image line offset */
            unsigned short   out_pitch; /* output image line offset */
};
```

5. White-balance parameters structure

```
struct prev_white_balance
{
            unsigned short wb_dgain;           /* white balance common gain */
            unsigned char  wb_gain[4];         /* individual color gains */
            unsigned char  wb_coefmatrix[4][4];/* 16 position, out of 4 values */
};
```

6. Black-adjustment parameter structure

```
struct prev_black_adjst    /* black adjustments for three colors */
{
            char redblkadj;   /* black adjustment offset for RED color */
            char greenblkadj; /* black adjustment offset for GREEN color */
            char blueblkadj;  /* black adjustment offset for BLUE color */
}
```

7. RGB-blending parameter structure

```
struct prev_rgbblending
{
            short blending[3][3];  /* color correlation 3x3 matrix */
            short offset[3];       /* color correlation offsets */
}
```

8. RGB-to-YCbCr parameter structure

```
struct prev_rgb2ycbcr_coeffs
{
            short coeff[3][3]; /* color conversion gains in a 3x3 matrix */
            short offset[3];   /* color conversion offsets */
}
```

9. CFA-interpolation parameter structure

```
struct prev_cfa_coeffs
{
            char   hthreshold, vthreshold; /* horizontal an vertical threshold */
            int coeffs[576];               /* cfa coefficients */
}
```

10. Gamma-coefficients structure

```
struct prev_gamma_coeffs
{
            unsigned char red[1024];   /* table of gamma correction values for
                                               red color */
            unsigned char green[1024]; /* table of gamma correction values for
                                          green color */
            unsigned char blue[1024];  /* table of gamma correction values for blue
                                         color */
};
```

11. Noise-coefficients structure

```
struct prev_noise_coeffs
{
            unsigned char noise[256];/* noise filter table */
            unsigned char strength;  /* to find out weighted average */
}
```

12. Chroma-suppression parameters structure

```
struct prev_chroma_suppress
{
            char hpfy; /* whether to use high passed version of Y or
                                  normal Y */
            char threshold;    /* threshold for chroma suppression */
            unsigned char gain; /* chroma suppression gain */
}
```

13. Previewing structure

```
struct prev_convert
{
            struct prev_buffer in_buf;  /* address of the input buffer */
            struct prev_buffer out_buf; /* address of the output buffer */
};
```

14. Preview-status structure

```
struct prev_status
{
          unsigned char hw_busy; /* 1: hardware is busy, 0: hardware is not busy
};
```

15. Preview-cropsize structure

```
struct prev_cropsize
{
            unsigned int hcrop; /*  number of pixels per line cropped in output
                                                image */
            unsigned int vcrop;   /*  number of lines cropped in output image */
};
```

3.3 API Classification

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

3.3.1 Configuration

This section contains Previewer Driver APIs that allow you to specify the desired configuration parameters. IOCTLs like PREV_SET_PARAMS help you to customize the Previewer Driver parameters. Section 3.5.2 elaborates on each such mechanism in greater detail.

3.3.2 Creation

This section contains all Previewer 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 PREV_REQBUFF and PREV_QUERYBUFF, and APIs like mmap are used for creating different components statically and dynamically. Section 3.5.2 elaborates on each such mechanism in greater detail.

3.3.3 Initialization

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

The API open is used for initializing of the Previewer Driver

3.3.4 Control

This section contains Previewer Driver APIs that are intended for use in controlling the functioning the Previewer Driver during run time. The IOCTL `PREV_PREVIEW` starts the previewing task by enabling previewing in the register.

3.3.5 Data Acquisition/Processing

This section contains the list of the Previewer Driver APIs that help to output parameters from the Previewer Driver.

IOCTLs like PREV_GET_STATUS are used to get the status of the hardware.

The IOCTL PREV_GET_PARAMS is used to get the previewing parameters configuration.

3.3.6 Termination

This section contains the Previewer Driver APIs that help in gracefully terminating the deployed 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.

3.4 API Usage Scenarios/Integration Example

The following figures show the usage scenarios for the Previewer Driver. Figure 1 shows a simple single-pass previewing task. Figure 2 shows multiple previewing tasks to be submitted to the Previewer driver.

Figure 1. Function Flow Diagram

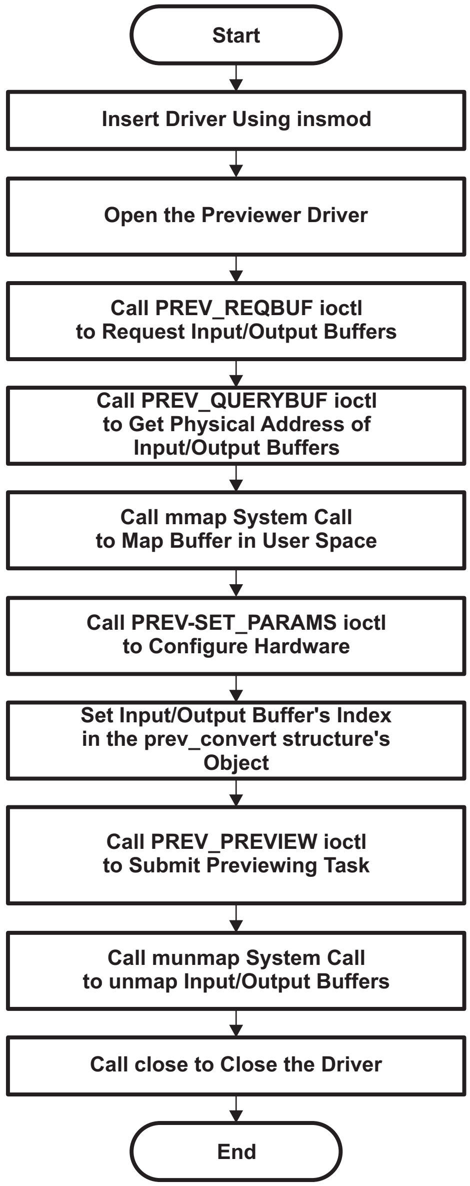
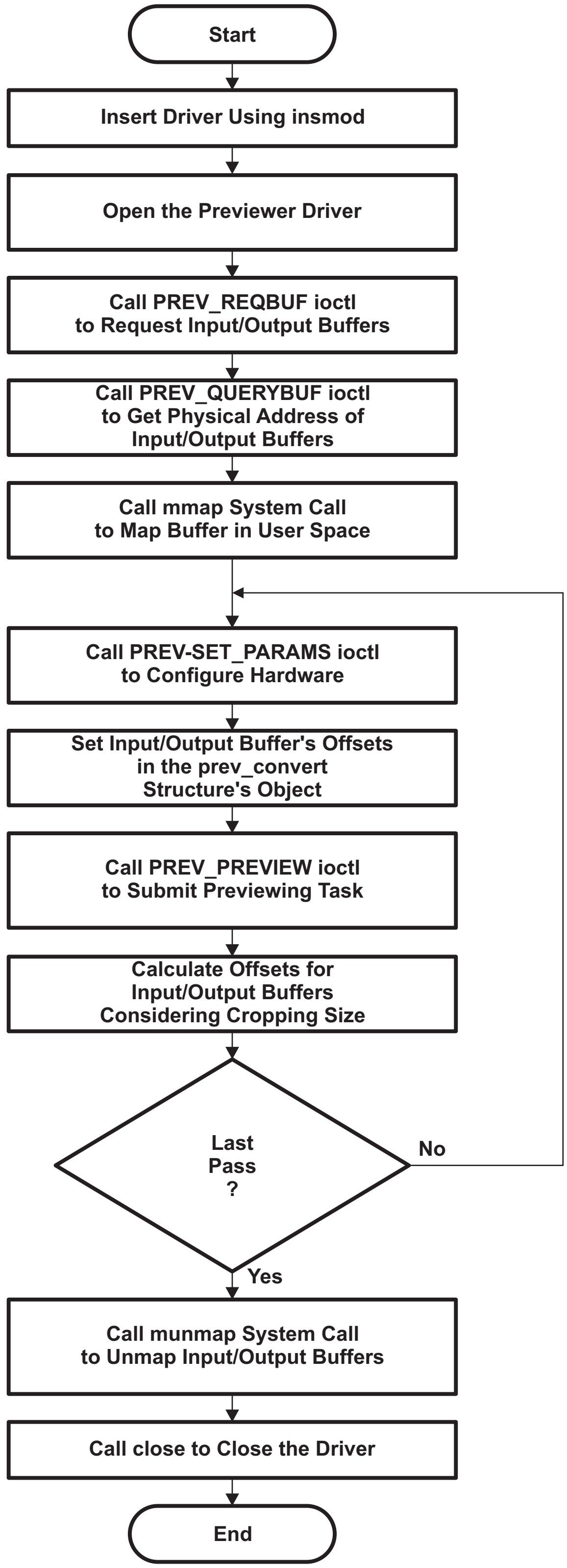


Figure 2. Function Flow Diagram for Multiple Passes



3.5 API Specification

This section describes the APIs and IOCTLs used in the driver.

3.5.1 Naming Conventions

The naming conventions are followed as per the Linux standard.

3.5.2 Previewer Driver Functions

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

API close

Prototype `int close(int fd)`

Description Closes the device driver that was opened with file descriptor.

Arguments

Arg1	int fd
Arg2	NA
Arg3	NA

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

Calling Constraints None

Example `close(fd);`

Side Effects None

See Also None

Errors None

IOCTL PREV_GET_PARAMS

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

Description Gets the Previewer driver hardware parameters.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct prev_params *argp

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

Calling Constraints None

Example `ioctl(fd, PREV_GET_PARAM, &params);`

Side Effects None

See Also None

Errors None

IOCTL PREV_GET_STATUS

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

Description Gets the current status of the hardware.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct prev_status *argp

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

Calling Constraints None

Example `ioctl(fd, PREV_GET_STATUS, &status);`

Side Effects None

See Also None

Errors None

IOCTL PREV_PREVIEW

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

Description Submits a previewing task to the hardware.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct prev_convert *argp

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

Calling Constraints It should be called after the parameters are configured.

Example `ioctl(fd, PREV_PREVIEW, &convert);`

Side Effects None

See Also None

Errors None

IOCTL PREV_REQBUF

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

Description Requests frame buffers to be allocated by the Previewer module.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct prev_reqbufs *argp

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

Calling Constraints The number of buffers requested cannot be greater than 8.

Example `ioctl(fd, PREV_REQBUF, &req_buf);`

Side Effects None

See Also None

Errors None

IOCTL PREV_SET_EXP

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

Description Sets the allowable delay between consecutive read requests from the Previewer module.

Arguments

Arg1	int fd
Arg2	int request
Arg3	int *argp

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

Calling Constraints All mandatory components of the hardware should be configured.

Example `ioctl(fd, PREV_SET_EXP, &arg);`

Side Effects None

See Also None

Errors None

IOCTL PREV_SET_PARAMS

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

Description Sets the Previewer hardware parameters.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct prev_params *argp

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

Calling Constraints All mandatory components of the hardware should be configured.

Example `ioctl(fd, PREV_SET_PARAM, &params);`

Side Effects None

See Also None

Errors None

IOCTL PREV_QUERYBUF

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

Description Requests the physical address of buffers allocated by the PREV_REQBUF ioctl.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct prev_buffer *argp

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

Calling Constraints None

Example `ioctl(fd, PREV_QUERYBUF, &buff);`

Side Effects None

See Also None

Errors None

IOCTL PREV_GET_CROPSIZE

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

Description Returns the size reduction in the output image compared to the input image, in terms of number of pixels per line and number of lines, depending on which features are enabled.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct prev_cropsize *argp

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

Calling Constraints None

Example `ioctl(fd, PREV_GET_CROPSIZE, &buff);`

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 Previewer module in kernel space to user space.

Arguments

Arg1	void *start
Arg2	size_t length
Arg3	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 None

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

Side Effects None

See Also None

Errors None

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
Arg3	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
Arg3	NA

Return Value	File descriptor on success or -1, if an error occurred.
Calling Constraints	None
Example	`open(“/dev/davinci_previewer”, O_RDWR)`
Side Effects	None
See Also	None
Errors	None

3.6 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.