LSP 1.20 DaVinci Linux AEW Driver

User's Guide

Literature Number: SPRUEP6
March 2008

Texas Instruments

Contents

List of Figures

List of Tables

LSP 1.20 DaVinci Linux AEW Driver

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

1 Overview

The AEW Driver provides the following functional services:

- The AEW Driver is a loadable module.
- The AEW Driver supports an image in bayer pattern.
- The AEW Driver supports input from CCDC.

1.1 System Requirements

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

1.2 Modules

The AEW 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 functionality for the driver.
- *Interrupt Handling*

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

  This module is responsible for providing the latest statistics to the application. Read call behavior will be as follows:
  - If the number of bytes requested by the application is less than the size of the available statistics, the driver returns an error to the application.
  - If the number of bytes requested by the application is greater than or equal to the size of the available statistics data, the driver copies all of the statistics data available in the latest statistics buffer to the application.
  - If the application issues a read call and if the new statistics are not available, the driver blocks the read call until new statistics are available or the timeout occurs. If new statistics are available before the timeout, the driver returns new statistics to the application, otherwise it returns an error.
  - The read function should return an error if it is already busy.

1.3 Layers

The AEW driver is divided into two horizontal layers.

- *Functional Layer*
  This layer implements all the functionalities and application interface.
- *Hardware Configuration Layer*
  This 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 AEW 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

The patch containing AEW Driver code, makefile, and Kconfig files.

2.2 Component Folder

The AEW 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 device 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 AEW Driver is provided through the `/dev/davinci_aew` device file. The `/dev/davinci_aew` device file is a character device that provides read/write access.

Use the following steps to enable AEW 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 *Davinci Auto Exposed/White Balancing Driver* option.

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

a. To make a static build, choose the *<*> Davinci Auto Exposed/White Balancing Driver* option.

b. To build as a module, choose the *<M> Davinci Auto Exposed/White Balancing Driver* option.

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

2.5 Steps to Load/Unload the AEW Driver

To load the driver module using dynamic-loadable modules, copy the modules (.ko files) to the target file system. Execute the following command to load the AEW Driver: `insmod davinci_aew_driver.ko`

Execute the following command to unload the AEW Driver: `rmmod davinci_aew_driver.ko`.

3 Run-Time Interfaces/Integration

This section discusses the AEW 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 either `#define` macros and/or enumerated C data types. The description column explains the meaning or value.

Table 1. Enumeration aew_alaw_enable

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
aew_alaw_enable	H3A_AEW_ENABLE = 1	Enable value.
aew_alaw_enable	H3A_AEW_DISABLE = 0	Disable value.

Table 2. Enumeration aew_config_flag

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
aew_config_flag	H3A_AEW_ENABLE = 1	The value indicates that the engine is configured.
aew_config_flag	H3A_AEW_DISABLE = 0	The value indicates that the engine is not configured.

Table 3. Defines

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
Macro	AEW_ERR_HZ_COUNT	Error code returned when user passes incorrect horizontal count value. The error code value is 820.
Macro	AEW_ERR_VT_COUNT	Error code returned when user passes incorrect horizontal count value. The error code value is 821.
Macro	AEW_ERR_HEIGHT	Error code returned when user passes incorrect window height value. The error code value is 822.
Macro	AEW_ERR_HZ_INCR	Error code returned when user passes incorrect horizontal increment value. The error code value is 823.
Macro	AEW_ERR_VT_INCR	Error code returned when user passes incorrect vertical increment value. The error code value is 824.
Macro	AEW_ERR_HZ_START	Error code returned when user passes incorrect horizontal start value. The error code value is 825.
Macro	AEW_ERR_VT_START	Error code returned when user passes incorrect vertical start value. The error code value is 826.
Macro	AEW_ERR_BLKWIN_HEIGHT	Error code returned when user passes incorrect black window height value. The error code value is 827.
Macro	AEW_ERR_BLKWIN_VT_START	Error code returned when user passes incorrect vertical black window start value. The error code value is 828.

Table 3. Defines (continued)

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
Macro	AEW_ERR_SETUP	Error code returned when user tries to enable the engine without configuring the hardware. The error code value is 829.
Macro	AEW_ERR_LIMIT	Error code returned when user passes incorrect saturation limit value. The error code value is 830.
Macro	AEW_ERR_ENGINE_BUSY	Error code returned to indicate that engine is busy. The error code value is 831.
Macro	AEW_ERR_WIDTH	Error code returned when user passes incorrect value of width of the window. The error code value is 832.
Macro	AEW_MAGIC_NO	Magic number for the IOCTL. The magic number is *e*.
Macro	AEW_WINDOW_SIZE	Size of the window. The window size is 18.

3.2 Data Structures

This section summarizes all visible data structures and their elements pertaining to the AEW Driver run-time interfaces.

The `aew_window` data structure is used to configure window parameters. The parameters are explained in Table 4.

Table 4. Structure aew_window

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
Unsigned int	width	Width of the window. The hardware increments this value by one and then multiplies it by 2. Range is 0 to 127.
unsigned int	height	Height of the window. The hardware increments this value by one and then multiplies it by 2. Range is 0 to 127.
unsigned int	hz_start	Horizontal window start. Range is 0 to 4095.
unsigned int	vt_start	Vertical window start. Range is 0 to 4095.
unsigned int	hz_cnt	Horizontal window count. Hardware increments this value by one. Range is 1 to 35.
unsigned int	vt_cnt	Horizontal window count. Hardware increments this value by one. Range is 0 to 127.
unsigned int	hz_line_incr	Horizontal window line increment. The hardware increments this value by one and then multiplies it by 2. Range is 0 to 15.
unsigned int	vt_line_incr	Vertical window line increment. The hardware increments this value by one and then multiplies it by 2. Range is 0 to 15.

The `aew_black_window` data structure is used to configure black window parameters. The parameters are explained in Table 5.

Table 5. Structure aew_black_window

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
unsigned int	height	Width of the window. The hardware increments this value by one and then multiplies it by 2. Range is 0 to 127.
unsigned int	vt_start	Vertical black window start. Range is 0 to 4095.

The `aew_configuration` data structure is used to configure the AEW Engine. The parameters are explained in Table 6.

Table 6. Structure aew_configuration

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
enum aew_alaw_enable	alaw_enable	Enumeration value that is used to enable or disable the AEW engine.
Int	saturation_limit	Saturation limit. Maximum value is 1023.
struct aew_window	window_config	Variable of the aew_window type structure. It stores the window configuration.
struct aew_black_window	blackwindow_config	Variable of the aew_window type structure. It stores the black window configuration.
enum aew_config_flag	aew_config	Enumeration value that indicates whether the engine is configured or not.

3.3 API Classification

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

3.3.1 Configuration

This section contains all the AEW Driver APIs that are used to specify the desired configuration parameters. The IOCTL AEW_S_PARAMS helps to customize the AEW Driver parameters. IOCTLs are detailed further in Section 3.5.2.

3.3.2 Creation

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

IOCTLs like AEW_S_PARAM will dynamically allocate buffers as per window count. It is mandatory to configure all parameters before enabling the engine. IOCTLs are detailed further in Section 3.5.2.

3.3.3 Initialization

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

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

3.3.4 Control

This section contains AEW Driver APIs that are intended for run-time control. The IOCTL AEW_ENABLE enables the AE/AWB engine. The IOCTL AEW_DISABLE disables the AE/AWB engine. IOCTLs are detailed further in Section 3.5.2.

3.3.5 Data Acquisition

This section lists all AEW Driver APIs that help to output data out of the driver. The IOCTL AEW_G_PARAM helps to get configuration parameters out of the AEW Driver. The API read is used to get statistics. The behavior of the read API is as follows:

- If the number of bytes requested by the application is less than the size of the available statistics, the driver returns an error to the application.
- If the number of bytes requested by the application is more than the size of the available statistics data, the driver copies all statistics data available in the latest statistics buffer to the application.
- If the application issues a read call and if new statistics are not available, the driver blocks the read call until either new statistics are available or the timer expires. If new statistics are available before the timer expires, the driver returns new statistics to the application.
- The read function should return an error if it is already blocked.

3.3.6 Termination

This section contains the AEW 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 AEW Driver. Figure 1 shows a simple single-pass task. Figure 2 shows an example of multiple tasks submitted to the AEW driver.

Figure 1. Function Flow Diagram

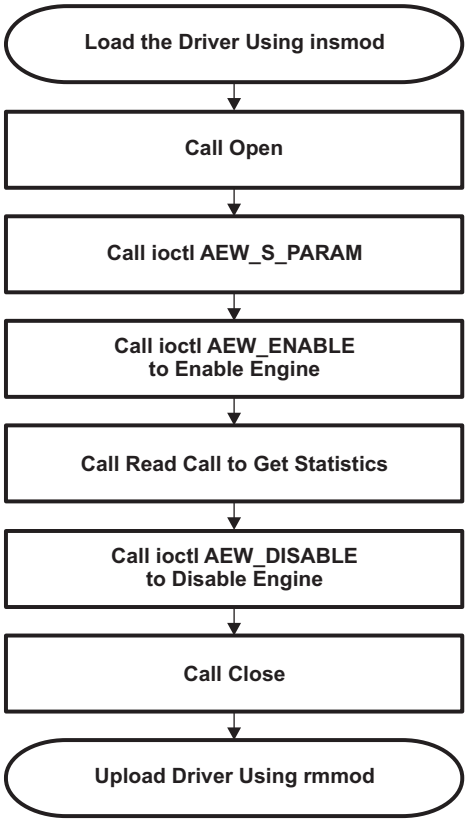
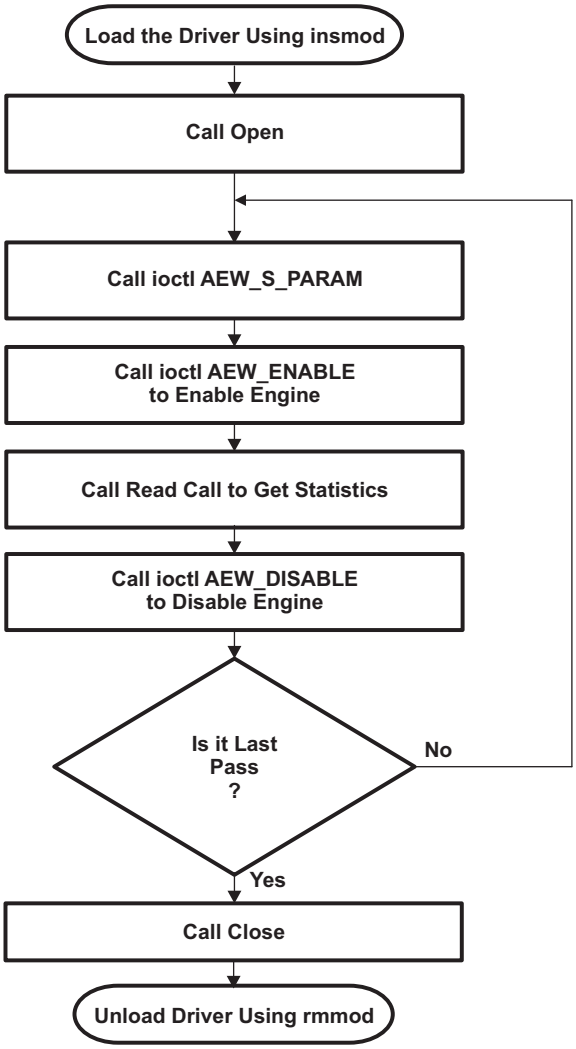


Figure 2. Function Flow Diagram for Multiple Passes



3.5 API Specification

This section describes the APIs and IOCTLs used in the AEW Driver.

3.5.1 Naming Conventions

The naming conventions are followed as per the GA license.

3.5.2 AEW 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 device driver opened with file descriptor.
Arguments	
	<arg1> int fd: File Descriptor
	<arg2> NA
	<arg3> NA
Return Value	0, on success, or -1, on error, and the errno variable is set appropriately.
Calling Constraints	None
Example	`close(fd)`
Side Effects	None
See Also	None
Errors	None

IOCTL AEW_DISABLE

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

Description Used to disable the AEW engine.

Arguments

<arg1>	int fd: File Descriptor
<arg2>	int request
<arg3>	NA

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

Calling Constraints None

Example `ioctl(fd, AEW_DISABLE)`

Side Effects None

See Also None

Errors None

IOCTL AEW_ENABLE

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

Description Used to enable the AEW engine with parameters set by AEW_S_PARAM.

Arguments

<arg1>	int fd: File Descriptor
<arg2>	int request
<arg3>	None

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

Calling Constraints None

Example `ioctl(fd, AEW_ENABLE)`

Side Effects None

See Also None

Errors AEW_ERR_SETUP: If user tries to enable the engine without configuring the engine.

IOCTL AEW_G_PARAM

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

Description Used to get the AEW hardware settings.

Arguments

<arg1>	int fd: File Descriptor
<arg2>	int request
<arg3>	struct aew_configuration * config: pointer to configuration structure.

Return Value Buffer size, on success, or -1, on error, and the errno variable is set appropriately.

Calling Constraints None

Example `ioctl(fd, AEW_G_PARAM,config)`

Side Effects None

See Also None

Errors EFAULT: If copy to user fails.

AEW_ERR_SETUP: If user tries to enable the engine without configuring the hardware.

IOCTL AEW_S_PARAM

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

Description Used to set the following parameters/modules of the AEW hardware:

- Window parameters
- Black window parameters
- Saturation check module
- A-Law compression module

Arguments

<arg1>	int fd: File Descriptor
<arg2>	int request
<arg3>	struct aew_configuration * config: pointer to configuration structure.

Return Value Buffer size, on success, or -1, on error, and the errno variable is set appropriately.

Calling Constraints All the members into the aew_configuration should be configured. There are no default values.

Example `ioctl(fd, AEW_S_PARAM,config)`

Side Effects None

See Also None

Errors ENOMEM: If memory is not available.

EFAULT: If copy from user fails.

ENOTTY: If ioctl is incorrect.

AEW_ERR_ENGINE_BUSY: If engine is busy.

Driver-specific error codes: If parameters are invalid. For driver-specific error codes, see Section 3.1.

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>	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

API read

Prototype `ssize_t read(int fd, void *buf, size_t count)`

Description Gets statistics from the driver.

- If the number of bytes requested by the application is less than the size of the available statistics, the driver returns an error to the application.
- If the number of bytes requested by the application is more than the size of the available statistics data, the driver copies all statistics data available in the latest statistics buffer to the application.
- If the application issues a read call and if new statistics are not available, the driver blocks the read call until either new statistics are available or the timer expires. If new statistics are available before the timer expires, the driver returns new statistics to the application.
- The read function should return an error if it is already blocked.

Arguments

<arg1>	int fd: File Descriptor
<arg2>	void *buffer: buffer which is used to store statistics
<arg3>	size_t count: size of buffer

Return Value Number of bytes read by the read call.

Calling Constraints None

Example `read(fd, buffer, count)`

Side Effects None

See Also None

Errors EBUSY: If read call is busy.

-1: If count is less than buffer size.

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.

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