

LSP 1.20 DaVinci Linux AF Driver

User's Guide

Literature Number: SPRUEP5
March 2008



1	Overview	5
	1.1 System Requirements	5
	1.2 Modules	5
	1.3 Layers	6
2	Installation Guide	6
	2.1 List of Installable Components	6
	2.2 Component Folder	6
	2.3 Development Tools	6
	2.4 Build.....	6
	2.5 Steps to Load/Unload the AF Driver.....	7
3	Run-Time Interfaces/Integration	7
	3.1 Symbolic Constants and Enumerated Data Types	7
	3.2 Data Structures	9
	3.3 API Classification	10
	3.4 API Usage Scenarios/Integration Example.....	11
	3.5 API Specification.....	13
	3.6 API Usage Recommendations	17

List of Figures

1	Function Flow Diagram	11
2	Function Flow Diagram for Multiple Passes	12

List of Tables

1	Enumeration af_alaw_enable	7
2	Enumeration config_flag	7
3	Enumeration af_hmf_enable	7
4	Enumeration rgbpos	7
5	Enumeration af_mode	8
6	Defines	8
7	Structure af_paxel	9
8	Structure af_hmf	9
9	Structure af_iir	9
10	Structure af_configuration	9

LSP 1.20 DaVinci Linux AF Driver

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

1 Overview

The AF Driver provides the following functional services:

- The AF Driver is a loadable module.
- The AF Driver supports images in Bayer pattern.
- The AF Driver supports input from CCDs.

1.1 System Requirements

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

1.2 Modules

The AF Driver consists of the following 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 the AF 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 AF 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 AF 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 AF Driver code, makefile, and Kconfig files.

2.2 Component Folder

The AF 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

You need to install the following tools, in the order listed below, to set up the development environment for the AF Driver:

- 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 AF Driver is provided through the `/dev/davinci_af` device file. The `/dev/davinci_af` device file is a character device that provides read/write access.

Use the following steps to enable AF 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*. Finally, choose the *Video for Linux* option.
- Step 4. At this point, the AF Driver can be built as static or as a module:
 - a. To make a static build, choose the `<*>` *Davinci Auto Focus Driver* option.
 - b. To build as a module, choose the `<M>` *Davinci Auto Focus 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 AF Driver

To load the driver using the dynamically-loadable modules, copy the modules (.ko files) to the target file system. Execute the following command to load the AF Driver: `insmod davinci_af_driver.ko`

Execute the following command to unload the AF Driver: `rmmmod davinci_af_driver.ko`

3 Run-Time Interfaces/Integration

This section discusses the AF 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. Described alongside the macro or enumeration is the description.

Table 1. Enumeration af_alaw_enable

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
af_alaw_enable	H3A_AF_ENABLE = 1	Enable value.
af_alaw_enable	H3A_AF_DISABLE = 0	Disable value. The value is equal to 0.

Table 2. Enumeration config_flag

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
af_config_flag	H3A_AF_ENABLE = 1	The value indicates that the engine is configured.
af_config_flag	H3A_AF_DISABLE = 0	The value indicates that the engine is not configured.

Table 3. Enumeration af_hmf_enable

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
af_hmf_enable	H3A_AF_ENABLE = 1	Enable value.
af_hmf_enable	H3A_AF_DISABLE = 0	Disable value.

Table 4. Enumeration rgbpos

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
rgbpos	GR_GB_BAYER = 0	This value indicates GR and GB as Bayer pattern.
rgbpos	RG_GB_BAYER = 1	This value indicates RG and GB as Bayer pattern.
rgbpos	GR_BG_BAYER = 2	This value indicates GR and BG as Bayer pattern.
rgbpos	RG_BG_BAYER	This value indicates RG and BG as Bayer pattern.
rgbpos	GG_RB_CUSTOM	This value indicates GR and RB as custom pattern.
rgbpos	RB_GG_CUSTOM	This value indicates RB and GG as custom pattern.

Table 5. Enumeration af_mode

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
af_mode	ACCUMULATOR_SUM MED = 0	Summed mode of the AF accumulator.
af_mode	ACCUMULATOR_PEAK = 1	Peak mode of the AF accumulator.

Table 6. Defines

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
Macro	AF_ERR_HZ_COUNT	Error code returned when user passes incorrect horizontal count value. The error code value is 800.
Macro	AF_ERR_VT_COUNT	Error code returned when user passes incorrect vertical count value. The error code value is 801.
Macro	AF_ERR_HEIGHT	Error code returned when user passes incorrect window height value. The error code value is 802.
Macro	AF_ERR_WIDTH	Error code returned when user passes incorrect window width value. The error code value is 803.
Macro	AF_ERR_INCR	Error code returned when user passes incorrect increment value. The error code value is 804.
Macro	AF_ERR_HZ_START	Error code returned when user passes incorrect horizontal start value. The error code value is 805.
Macro	AF_ERR_VT_START	Error code returned when user passes incorrect vertical start value. The error code value is 806.
Macro	AF_ERR_IIRSH	Error code returned when user passes incorrect value black window height. The error code value is 807.
Macro	AF_ERR_IIR_COEF	Error code returned when user passes incorrect value black window vertical start. The error code value is 808.
Macro	AF_ERR_SETUP	Error code returned when user tries to enable the engine without configuring the hardware. The error code value is 809.
Macro	AF_ERR_THRESHOLD	Error code returned when user passes incorrect saturation limit value. The error code value is 810.
Macro	AF_ERR_ENGINE_BUSY	Error code returned to indicate that the engine is busy. The error code value is 811.
Macro	AF_NUMBER_OF_COEFF	Error code returned when user passes incorrect window width value. The number of coefficient is 11.
Macro	AF_MAGIC_NO	Magic number of AF. The magic number is a.
Macro	AF_PAXEL_SIZE	Size of the paxel. The paxel size is 48.

3.2 Data Structures

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

The `af_paxel` data structure is used to configure paxel parameters. The parameters are explained in [Table 7](#).

Table 7. Structure af_paxel

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 start of the paxel. Range is 2 to 4094.
unsigned int	vt_start	Vertical start of the paxel. Range is 0 to 4095.
unsigned int	hz_cnt	Horizontal count of the paxel. Hardware increments this value by one. Range is 0 to 35.
unsigned int	vt_cnt	Horizontal count of the paxel. Hardware increments this value by one. Range is 0 to 127.
unsigned int	line_incr	Line increment of the paxel. The hardware increments this value by one and then multiplies it by 2. Range is 0 to 14.

The `af_hmf` data structure is used to configure HMF Filter parameters. The parameters are explained in [Table 8](#).

Table 8. Structure af_hmf

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
enum af_hmf_enable	enable	Status of horizontal median filter.
unsigned int	threshold	Horizontal filter threshold. Maximum value is 255.

The file `davinci_af.h` has the `af_iir` data structure that is used to configure IIR filter parameters. The parameters are explained in [Table 9](#).

Table 9. Structure af_iir

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
Unsigned int	hz_start_pos	IIR start register value. Range is 0 to 4094.
Int	coeff_set0[AF_NUMBER_OF_COEF];	Coefficients of the IIR Filter 0. Each coefficient is 12 bits wide with 6 bits of decimal precision. Coefficients are signed.
Int	coeff_set1[AF_NUMBER_OF_COEF]	Coefficients of the IIR Filter 1. Each coefficient is 12 bits wide with 6 bits of decimal precision. Coefficients are signed.

The `af_configuration` data structure is used to configure AF Engine. The parameters are explained in [Table 10](#).

Table 10. Structure af_configuration

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
enum af_alaw_enable	alaw_enable	Enumeration value that is used to enable or disable a-law.
enum rgbpos	rgb_pos	RGB position.

Table 10. Structure af_configuration (continued)

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
struct af_hmf	hmf_config	Variable of the type structure af_hmf. It stores the configuration for HMF filter.
struct af_iir	iir_config	Variable of the type structure af_iir. It stores the configuration for IIR filters.
struct af_paxel	paxel_config;	Variable of the type structure af_paxel. It stores the configuration for paxels.
enum af_mode	mode	Accumulator mode.
enum af_config_flag	af_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 AF Driver.

3.3.1 Configuration

This section contains all the AF Driver APIs that are used to specify the desired configuration parameters. IOCTLs like AF_S_PARAMS help to customize the AF Driver parameters. IOCTLs are detailed further in [Section 3.5.2](#).

3.3.2 Creation

This section contains all the AF 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 AF_S_PARAM dynamically allocate buffers as per the paxel 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 AF Driver APIs that are intended for use in component initialization.

The API open is used for initializing of the AF configuration structure.

3.3.4 Control

This section contains AF Driver APIs that are intended for run-time control. The IOCTL AF_ENABLE enables the AF engine. The IOCTL AF_DISABLE disables the AF engine. IOCTLs are detailed further in [Section 3.5.2](#).

3.3.5 Data Acquisition

This section lists all AF Driver APIs that help to output data out of the driver. The IOCTL AF_G_PARAM helps to get configuration parameters out of the AF 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 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.

3.3.6 Termination

This contains the AF Driver APIs that help in gracefully terminating the deployed driver run-time entities. The API close is used to free all resources that are being acquired at the time of initialization and creation.

3.4 API Usage Scenarios/Integration Example

Figure 1 and Figure 2 show the usage scenarios for the AF Driver. Figure 1 shows an example of a single-pass task. Figure 2 shows an example of multiple tasks submitted to the AF driver.

Figure 1. Function Flow Diagram

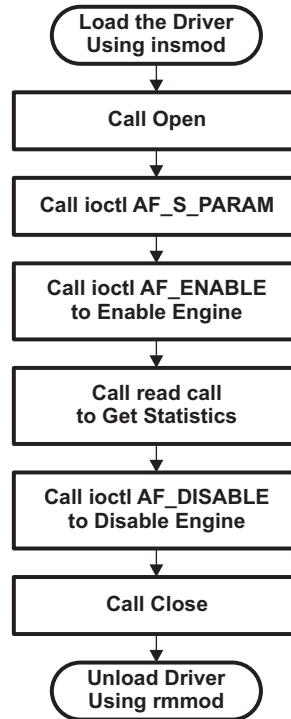
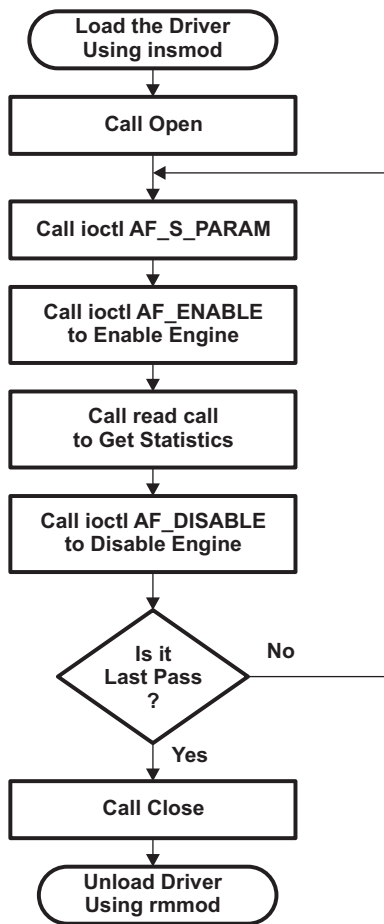


Figure 2. Function Flow Diagram for Multiple Passes



3.5 API Specification

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

3.5.1 Naming Conventions

The naming conventions are followed as per the GA license.

3.5.2 AF 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 AF_DISABLE

Prototype	<code>int ioctl(int fd, int command)</code>						
Description	Used to disable the AF engine.						
Arguments	<table> <tr> <td><arg1></td> <td>int fd : File Descriptor</td> </tr> <tr> <td><arg2></td> <td>int request</td> </tr> <tr> <td><arg3></td> <td>NA</td> </tr> </table>	<arg1>	int fd : File Descriptor	<arg2>	int request	<arg3>	NA
<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	<code>ioctl(fd, AF_DISABLE)</code>						
Side Effects	None						
See Also	None						
Errors	None						

IOCTL AF_ENABLE

Prototype	<code>int ioctl(int fd, int command)</code>						
Description	Used to enable the AF engine with parameters set by AF_S_PARAM.						
Arguments	<table> <tr> <td><arg1></td> <td>int fd : File Descriptor</td> </tr> <tr> <td><arg2></td> <td>int request</td> </tr> <tr> <td><arg3></td> <td>NA</td> </tr> </table>	<arg1>	int fd : File Descriptor	<arg2>	int request	<arg3>	NA
<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	<code>ioctl(fd, AF_ENABLE)</code>						
Side Effects	None						
See Also	None						
Errors	AF_ERR_SETUP: If user tries to enable the engine without configuring the hardware.						

IOCTL AF_G_PARAM

Prototype	<code>int ioctl(int fd, int command, struct af_configuration *arg)</code>
Description	Used to get the AF hardware settings.
Arguments	<p><arg1> int fd : File Descriptor <arg2> int request <arg3> struct af_configuration * config : pointer to configuration structure.</p>
Return Value	Buffer size, on success, or -1, on error, and the errno variable is set appropriately.
Calling Constraints	None
Example	<code>ioctl(fd, AF_G_PARAM, config)</code>
Side Effects	None
See Also	None
Errors	EFAULT: If copy to user fails. AF_ERR_SETUP: If user tries to enable the engine without configuring the hardware.

IOCTL AF_S_PARAM

Prototype	<code>int ioctl(int fd, int command, struct af_configuration *arg)</code>
Description	Used to set the following parameters/modules of the AF hardware: <ul style="list-style-type: none">• Paxel parameters• IIR filter parameters• HMF filter Parameter• A-Law compression module
Arguments	<p><arg1> int fd : File Descriptor <arg2> int request <arg3> struct af_configuration * config : pointer to configuration structure.</p>
Return Value	Buffer size, on success, or -1, on error, and the errno variable is set appropriately.
Calling Constraints	All members into the af_configuration should be configured. There are no default values.
Example	<code>ioctl(fd, AF_S_PARAM, config)</code>
Side Effects	None
See Also	None
Errors	ENOMEM: If memory is not available. EFAULT: If copy from user fails. ENOTTY: If ioctl is incorrect. AF_ERR_ENGINE_BUSY: If engine is busy. Driver-specific error codes: If parameter are invalid. For driver-specific error codes, see Section 3.1 .

API open

Prototype	<code>int open(char *name, int mode)</code>						
Description	Opens the driver in the mode specified in the last parameter.						
Arguments	<table> <tr> <td><arg1></td> <td>char *name</td> </tr> <tr> <td><arg2></td> <td>int mode</td> </tr> <tr> <td><arg3></td> <td>NA</td> </tr> </table>	<arg1>	char *name	<arg2>	int mode	<arg3>	NA
<arg1>	char *name						
<arg2>	int mode						
<arg3>	NA						
Return Value	File descriptor, on success, or -1, if an error occurred.						
Calling Constraints	None						
Example	<code>open("/dev/davinci_af", O_RDWR)</code>						
Side Effects	None						
See Also	None						
Errors	None						

API read

Prototype	<code>ssize_t read(int fd, void *buf, size_t count)</code>						
Description	<p>Gets statistics from the driver.</p> <ul style="list-style-type: none"> • 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. 						
Arguments	<table> <tr> <td><arg1></td> <td>int fd : File Descriptor</td> </tr> <tr> <td><arg2></td> <td>void *buffer : buffer which is used to store statistics</td> </tr> <tr> <td><arg3></td> <td>size_t count : size of buffer</td> </tr> </table>	<arg1>	int fd : File Descriptor	<arg2>	void *buffer : buffer which is used to store statistics	<arg3>	size_t count : size of buffer
<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	<code>read(fd, buffer, count)</code>						
Side Effects	None						
See Also	None						
Errors	<p>EBUSY: If read call is busy.</p> <p>-1: If count is less than buffer size.</p>						

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

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Medical	www.ti.com/medical
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2008, Texas Instruments Incorporated