LSP 1.20 DaVinci Linux Resizer Driver

User's Guide

Literature Number: SRUFF0
March 2008

Texas Instruments

Contents

List of Figures

List of Tables

LSP 1.20 DaVinci Linux Resizer Driver

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

1 Overview

This section describes the functional scope of the Resizer Driver and its feature set. The section also details the various deployment environments, hardware and software, on which the Resizer Driver is presently supported. The chapter introduces the system architecture of the Resizer Driver along with the functional decomposition and run-time specifics regarding deployment of Resizer Driver in your application.

The Resizer Driver provides the following functional services:

- The Resizer Driver is a loadable module.
- The Resizer Driver supports YUV422 color interleaved and 8-bit color separate data input formats.
- The Resizer Driver supports input from SDRAM or DDRAM.
- The Resizer Driver is a standalone kernel module so that it can be used by multiple applications.
- The Resizer Driver supports the upscaling/downscaling up to one-fourth to 4x.

1.1 System Requirements

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

1.2 Device Driver Design

This section provides a functional view for each of the modules. The functional view elaborates on the construction mechanics of each of the modules (and associated layers).

1.2.1 Modules

The Resizer Driver is sub-divided into the 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 resize functionality of the driver.
- *Interrupt Handling*

  This is the interrupt handler for the Resizer Driver. It handles interrupt generated by Resizer hardware at the completion of processing a frame.
- *Buffer Management*

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

- *Multiple Channel Handling*

  This module handles priorities between multiple channels. This module provides support to add channel handles to the pending queue as per priority, to query highest priority pending request, and to remove specific channel handle.

1.2.2 Layers

The complete Resizer Driver is divided in to 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 configuring the hardware.

2 Installation Guide

This section discusses the Resizer Driver installation, what software and hardware components are used, and how to complete a successful installation of the Resizer Driver.

2.1 List of Installable Components

A patch containing Resizer 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 Tool Environment(s)

This section describes the development tool environment(s) you use for software development with the Resizer Driver. It describes the tools used for each supported environment.

Install the following tools in the given sequence 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 have any specific build options at the time of writing of this manual.

2.4.2 Build Steps

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

Use the following steps to enable resizer 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 Resizer Driver Support* option.

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

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

```
insmod davinci_rsz_driver.ko
```

Execute the following command to unload the Resizer Driver module:

```
rmmod davinci_rsz_driver.ko
```

3 Run-Time Interfaces/Integration

This section discusses the Resizer 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 value and explanation.

Table 1. Group or Enumerated Class

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
Macro	N_INBUFS	Number of input buffers to be allocated. The maximum number is restricted to 8.
Macro	N_OUTBUFS	Number of input buffers to be allocated. The maximum number is restricted to 8.
Macro	RSZ_BUF_IN	Indicates that the buffer ask is a input buffer. Its value is represented by 0.
Macro	RSZ_BUF_OUT	Indicates that the buffer ask is a output buffer. Its value is represented by 1.
Macro	RSZ_CBILIN_DISABLE	Indicates that luminance processing is disabled. Its value is represented by 0.
Macro	RSZ_CBILIN_ENABLE	Indicates that luminance processing is enabled. Its value is represented by 1.
Macro	RSZ_PIX_FMT_UYVY	Indicates that the input image is of form UYVY. Its value is represented by 0.
Macro	RSZ_PIX_FMT_YUYV	Indicates that the input image is of for YUYV. Its value is represented by 1.
Macro	RSZ_PIX_FMT_PLANAR	Indicates that the input image is 8-bit color separated. Its value is represented by 2.
Macro	RSZ_YENH_DISABLE	Indicates the luma-enhancement algorithm is disabled. Its value is represented by 0.
Macro	RSZ_YENH_3TAP_HPF	Indicates the luma-enhancement algorithm used is 3 Tap. Its value is represented by 1.
Macro	RSZ_YENH_5TAP_HPF	Indicates the luma-enhancement algorithm used is 5 Tap. Its value is represented by 2.
Macro	RSZ_INTYPE_YCBCR422_16BIT	Indicates the input image is 16-bit color interleaved. Its value is represented by 0.
Macro	RSZ_INTYPE_PLANAR_8BIT	Indicates the input image is 8-bit color separated. Its value is represented by 0.

3.2 Data Structures

This section summarizes user-visible data structure elements pertaining to the Resizer Driver run-time interfaces.

3.2.1 Structures for Buffer Management

1. Buffer allocation structure

```
struct rsz_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 rsz_buffer
{
     int index;      //  index number, 0 -> –1
     int buf_type;   // buffer type, input or output
     int offset;     // physical address of the buffer used in the mmap() system call
     int size;       // Hold the size of the buffer  requested
};
```

3.2.2 Structures for Configuration and Control

1. Luma enhancement Configure structure

```
struct rsz_yenh
{
     int type;               // represents luma enable or disable
     unsigned char gain;     // Represents gain
     unsigned char slop;     // represents slop
     unsigned char core;     // Represents core value
};
```

2. Configuration Parameters structure

```
struct rsz_params
{
    int in_hsize;             // input frame horizontal size
    int in_vsize;             // input frame vertical size
    int in_pitch;            // offset between two rows of input frame
    int inptyp               // for determining 16 bit or 8 bit data
    int vert_starting_pixel  // for specifying vertical starting pixel in input
    int horz_starting_pixel  // for specyfing horizontal starting pixel in input
    int cbilin;              // defined, filter with luma or bi-linear interpolation
    int pix_fmt;             // defined, UYVY or YUYV
    int out_hsize;           // output frame horizontal size
    int out_vsize;           // output frame vertical size
    int out_pitch;           // offset between two rows of output frame
    int hstph;               // for specifying horizontal starting phase
    int vstph;               // for specifying vertical starting phase
     short hfilt_coeffs[32];      // horizontal filer coefficients
     short vfilt_coeffs[32];      // vertical filter coefficients
     struct rsz_yenh yenh_params; // Luma enhancement structure
};
```

3. Resize structure

```
struct rsz_resize
{
     struct rsz_buffer in_buf;  // address of the input buffer
     struct rsz_buffer out_buf; // address of the output buffer

};
```

4. Resize status structure

```
struct rsz_status
{
    int chan_busy;  // 1: channel is busy, 0: channel is not busy
    int hw_busy;    // 1: hardware is busy, 0: hardware is not busy
    int src;        // defined, can be either SD-RAM or CCDC/PREVIEWER
};
```

5. Priority structure

```
struct rsz_priority
{
     int priority;   // Priority = 0=>5, with 5 the highest priority
};
```

6. Resizer 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 Resizer Driver.

3.3.1 Configuration

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

3.3.2 Creation

This section contains Resizer 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 RSZ_REQBUFF and RSZ_QUERYBUFF and APIs like mmap are used for creation of different components statically and dynamically. Section 3.5.2 elaborates on each such mechanism in greater detail.

3.3.3 Initialization

This contains the Resizer Driver APIs that are intended for use in component initialization. The API open is used for initializing of the resizer channel configuration structure.

3.3.4 Control

This contains Resizer Driver APIs that is intended for use in controlling the functioning of Resizer Driver during run-time. The IOCTL RSZ_RESIZE enables the resize bit after completing the hardware register configuration.

3.3.5 Data Acquisition

This section lists Resizer Driver APIs that help to output parameters out of Resizer Driver.

IOCTLs like RSZ_G_STATUS are used to get the status of the hardware and channel.

The IOCTL RSZ_G_PRIORITY is used to get the priority of the resizing request.

The IOCTL RSZ_G_PARAMS is used to get the resizing parameters configuration.

3.3.6 Termination

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

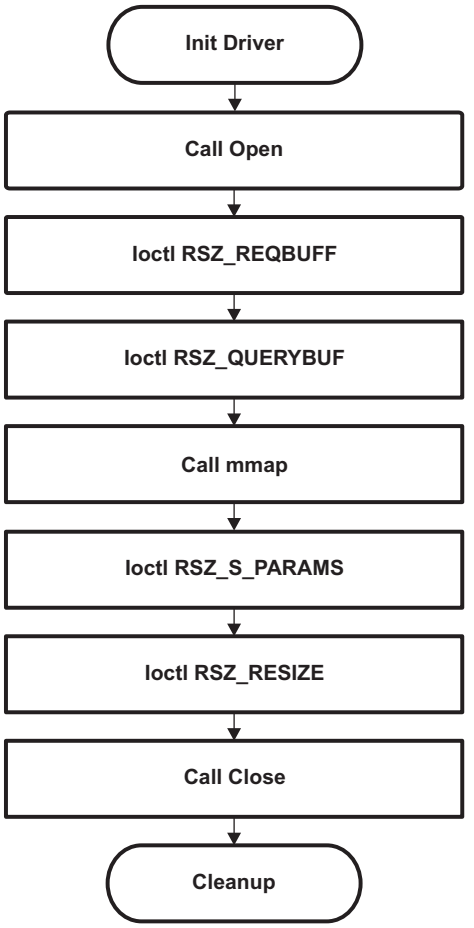
The API close is used to free all the resources that are being acquired at channel initialization time.

3.4 API Usage Scenarios/Integration Example

The API usage scenarios are illustrated by suitable methods such as state charts, annotated graphs, or sequence diagrams etc., First, the generic scenario applicable to all APIs in the run-time interface is discussed. Next, the scenarios applicable to specific functional groups are detailed.

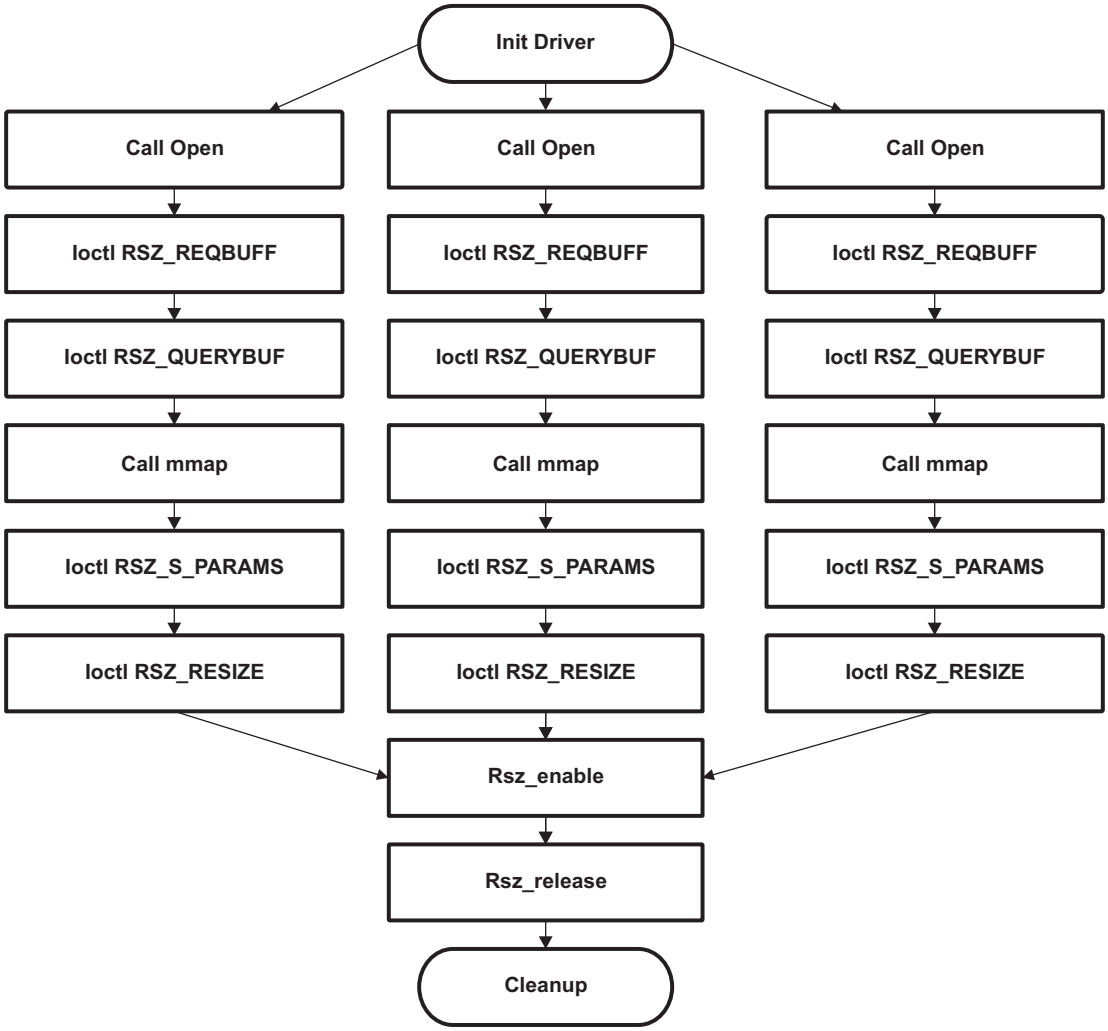
The generic function flow diagram is shown in Figure 1.

Figure 1. Generic Function Flow Diagram



The diagram that describes the multiple channels is shown in Figure 2. In case of multiple channels, only one channel can use the hardware at a time.

Figure 2. Multiple-Channel Flow Diagram



3.5 API Specification

3.5.1 Naming Conventions

The naming conventions are followed as per the GA guidelines.

3.5.2 Resizer 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 logic channel associated with the file descriptor.

Arguments

Arg1	Int fd
Arg2	None
Arg3	None

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

Calling Constraints Only the open channel can be closed via the file descriptor.

Example `Close(fd);`

Side Effects None

See Also None

Errors None

IOCTL RSZ_GET_CROPSIZE

Prototype `int ioctl(int fd, int command, struct rsz_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 the enabled features.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct rsz_cropsize *argp

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

Calling Constraints None

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

Side Effects None

See Also None

Errors None

IOCTL RSZ_G_PARAMS

Prototype `int ioctl(int fd, RSZ_G_PARAMS, struct rsz_params *argp)`

Description Gets the resizer hardware parameters associated with this logic channel.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct rsz_params *argp

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

Calling Constraints The ioctl only can be called; the logic channel parameters are set.

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

Side Effects None

See Also None

Errors None

IOCTL RSZ_G_PRIORITY

Prototype `int ioctl(int fd, RSZ_G_PRIORITY, struct rsz_priority *argp)`

Description Gets the current priority setting of the logic channel identified by fd.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct rsz_priority *argp

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

Calling Constraints None

Example
```
ioctl(fd, RSZ_G_PRIORITY, &priority);
```

Side Effects None

See Also None

Errors None

IOCTL RSZ_G_STATUS

Prototype `int ioctl(int fd, RSZ_G_STATUS, struct rsz_status *argp)`

Description Gets the current status of the hardware.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct rsz_status *argp

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

Calling Constraints None

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

Side Effects None

See Also None

Errors None

IOCTL RSZ_RESIZE

Prototype `int ioctl(int fd, RSZ_RESIZE, struct rsz_resize *argp)`

Description Submits a resizing task to the logic channel associated with fd.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct rsz_resize *argp

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

Calling Constraints It should be called after parameters are configured and memory allocation is complete.

Example `ioctl(fd, RSZ_RESIZE, &resize);`

Side Effects None

See Also None

Errors None

As the actual value for the RSZ register is calculated, the actual value for the maximum and minimum scaling factors are little less than 4x and little more than 0.25x. This value can be calculated using the following formula:

*RSZ = ((IW – 7) * 256 – 16 – 32*SPH) / (OW – 1)*

Resize Ratio = 256 / RSZ

Where,

IW = Input Width

OW = Output Width

SPH = Starting Phase

IOCTL RSZ_REQBUF

Prototype `int ioctl(int fd, RSZ_REQBUF, struct rsz_reqbufs *argp)`

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

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct rsz_reqbufs *argp

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

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

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

Side Effects If the requested frame is not free at the end, it may cause a memory shortage.

See Also None

Errors None

IOCTL RSZ_S_EXP

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

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

Arguments

Arg1	int fd
Arg2	int request
Arg3	Int *arg

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

Calling Constraints None

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

Side Effects None

See Also None

Errors None

IOCTL RSZ_S_PARAMS

Prototype `int ioctl(int fd, RSZ_S_PARAMS, struct rsz_params *argp)`

Description Sets the resizer hardware parameters associated with this logic channel.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct rsz_params *argp

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

Calling Constraints All the members of the rsz_params struct should be configured; there are no default values.

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

Side Effects None

See Also None

Errors None

IOCTL RSZ_S_PRIORITY

Prototype `int ioctl(int fd, RSZ_S_PRIORITY, struct rsz_priority *argp)`

Description Sets the current priority setting of the logic channel identified by fd.

Arguments

Arg1	int fd
Arg2	Int request
Arg3	struct rsz_priority *argp

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

Calling Constraints None

Example
```
ioctl(fd, RSZ_S_PRIORITY, &priority);
```

Side Effects None

See Also None

Errors None

IOCTL RSZ_QUERYBUF

Prototype `int ioctl(int fd, RSZ_QUERYBUF, rsz_buffer *argp)`

Description Requests the physical address of the buffers allocated by the RSZ_REQBUF.

Arguments

Arg1	int fd
Arg2	int request
Arg3	struct rsz_buffer *argp

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

Calling Constraints None

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

Side Effects None

See Also None

Errors None

API MMAP

Prototype	`void * mmap(void *,size_t image_size,int prot,int flags,int fd,off_t offset)`
Description	Maps the frame buffers allocated by the RSZ module in kernel space to user space.
Arguments	

Arg1	void *start :-
Arg2	size_t length
Arg3	int prot
Arg4	int flags *(Only MAP_SHARED is supported)*
Arg5	int fd
Arg6	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, ch2_fd, offset);`
Side Effects	None
See Also	None
Errors	None

API MUNMAP

Prototype	`int munmap(void *start,size_t 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(0, image_size)`
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.

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
Copyright © 2008, Texas Instruments Incorporated