LSP 1.20 DaVinci Linux IPIPE Driver

User's Guide

Literature Number: SPRUFG1
April 2008

Contents

List of Figures

LSP 1.20 DaVinci Linux IPIPE Driver

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

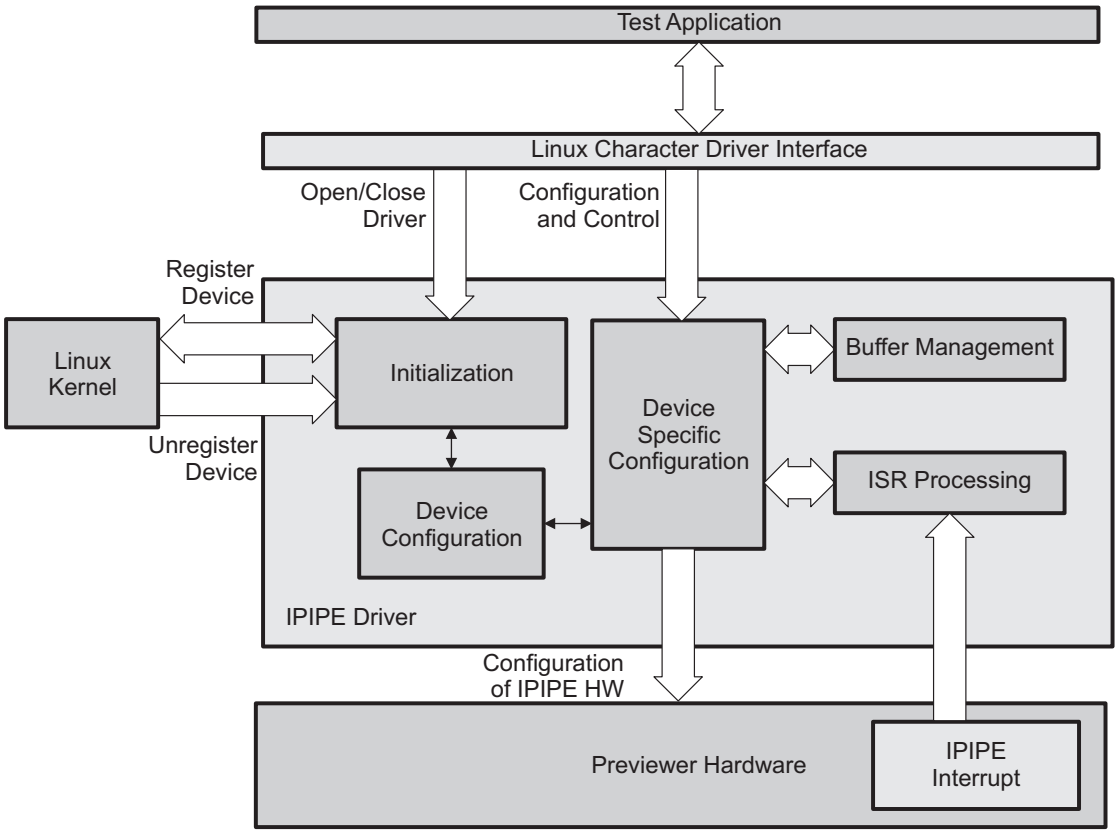
1 Architecture Overview

The IPIPE Driver is contained in the following files:

- dm350_ipipe.c
- dm350_ipipe_hw.c
- dm350_ipipe.h
- dm350_ipipe_hw.h

Figure 1 shows how the IPIPE Driver fits into the system architecture.

Figure 1. System Diagram



The IPIPE Driver is sub-divided into the following vertical modules:

- *Initialization*

  This module handles all the initialization activities including driver registration, driver un-registration, configuration creation, and configuration deletion.
- *Configuration and Control*

  This module handles input, previewing, and resizing functionality of the driver.
- *Interrupt Handling*

  This is the interrupt handler for the IPIPE Driver. It handles interrupts generated by the IPIPE hardware for MMR modification interrupt (`IPIPE_INT1_SDR`).
- *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.

The IPIPE Driver is divided into two horizontal layers:

- *Functional Layer*

  This layer implements all the functionalities and the application interface.
- *HW Configuration Layer*

  This layer contains functions to configure the hardware. These functions are used by the functional layer for configuration and control.

2 Application Level APIs

The following Linux driver APIs are supported by the IPIPE Driver

- open()
- close()
- mmap()
- munmap()
- ioctl()

OPEN

Prototype `int open(const char *pathname, int flags);`

Description Opens the IPIPE. Multiple opens are not supported by the IPIPE Driver.

Arguments

pathname	The location of the device file. The value normally is */dev/dm350_ipipe*; flags = O_RDWR.
flags	Only O_RDWR is supported.

Return Value A new file descriptor or -1, if an error occurs.

CLOSE

Prototype `int close(int fd);`

Description Closes the IPIPE.

Arguments

fd	File descriptor returned by open().

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

MMAP

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

Description Maps the frame buffers allocated by the IPIPE Driver in kernel space to user space.

Arguments

start	Usually 0.
length	Size of the memory to be mapped.
prot	Only PROT_READ | PROT_WRITE is supported.
flags	Only MAP_SHARED is supported.
fd	File descriptor returned by open().
offset	The kernel address of the physical memory to be mapped; for more information, see Section 3.

Return Value Pointer to the mapped area, on success, or MAP_FAILED (-1), if an error occurred.

MUNMAP

Prototype `int munmap(void *start, size_t length);`

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

Arguments

start	The user space virtual address of the memory block to be ummaped.
length	Size of the memory to be unmapped.

Return Value Pointer to the mapped area, on success, or MAP_FAILED (-1), if an error occurred.

IOCTL

Prototype `int ioctl(int fd, int request, void *argp);`

Description Used to program the IPIPE. The argument *fd* must be an open-file descriptor. An ioctl request has encoded in it whether the argument is an input, output, or read/write parameter, and the size of the argument *argp*, in bytes. Macros and defines specifying IPIPE ioctl requests are located in the dm350_ipipe.h header file. Applications should use their own copy and not include the version in the kernel sources on the system where they compile. All IPIPE ioctl requests and their respective functions and parameters are specified in Section 3.

Arguments

fd	File descriptor returned by open().
request	IPIPE ioctl request code as defined in the dm350_ipipe.h header file, for example IPIPE_SET_PARAM.
argp	Pointer to a function parameter, usually a structure.

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

3 IPIPE Driver IOCTLS

ioctl IPIPE_REQBUF

Prototype `int ioctl(int fd, int request, struct ipipe_reqbufs *argp);`

Description Used to request frame buffers to be allocated by the IPIPE Driver. The allocated buffers can be addressed by indexing; i.e., 0, 1, up to N-1.

The `ipipe_reqbufs` structure and its fields are defined in `dm350_ipipe.h`.

Arguments

fd	File descriptor returned by open().
request	IPIPE_REQBUF
argp	Pointer to the ipipe_reqbufs structure.

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

ioctl IPIPE_QUERYBUF

Prototype `int ioctl(int fd, int request, struct ipipe_buffer *argp);`

Description Used to query the status of a particular frame buffer.

The definition of the `ipipe_buffer` structure is in the `dm350_ipipe.h` header file.

Arguments

fd	File descriptor returned by open().
request	IPIPE_QUERYBUF
argp	Pointer to the ipipe_buffer structure.

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

ioctl IPIPE_SET_PARAM

Prototype `int ioctl(int fd, int request, struct ipipe_params *argp);`

Description Used to set the parameters (either default or user configurable) of the IPIPE hardware. If the argument passed is NULL, the driver sets the default parameters, including input and output image size, input source, output pixel format, RGB to RGB blending coefficients, etc. The application is responsible for specifying all these parameters. The driver sets these parameters in the appropriate registers.

The `ipipe_params` and its fields as defined in the `dm350_ipipe.h` header file.

Arguments

fd	File descriptor returned by open().
request	IPIPE_SET_PARAM
argp	Pointer to the ipipe_params structure.

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

ioctl IPIPE_GET_PARAM

Synopsis `int ioctl(int fd, int request, struct ipipe_params *argp);`

Description Used to get the IPIPE hardware settings. If it is called after SET_PARAM (with the argument as NULL), it gets the default parameters.

Arguments

fd	File descriptor returned by open().
request	IPIPE_GET_PARAMS
argp	Pointer to the ipipe_params structure.

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

ioctl IPIPE_START

Prototype `int ioctl(int fd, int request, struct ipipe_convert *argp);`

Description Submits an IPIPE processing task specified by the `ipipe_convert` structure to the hardware. The call is blocked until the task is complete.

The `ipipe_convert` structure is defined in the `dm350_ipipe.h` header file.

Arguments

fd	File descriptor returned by open().
request	IPIPE_START
argp	Pointer to the ipipe_convert structure.

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

4 Usage Examples

This section provides some code examples showing how to use the IPIPE Driver.

4.1 Driver Open and Close

```
/* open the ipipe*/
int fd;
fd = open(“/dev/dm350_ipipe, O_RDWR);
if(fd == -1) {
printf(“open IPIPE failed.\n”)
exit(-1);
}

/* close the ipipe*/
close(fd);
```

4.2 Buffer Allocation and Mapping

```
struct ipipe_reqbufs, req_inbufs, req_outbufs;
struct ipipe_buffer bufd;
void *inBufs[N_INBUFS];
void *outBufs[N_OUTBUFS];
int i;

/* request input buffers to be allocated */
req_inbufs.buf_type = IPIPE_BUF_IN;
req_inbufs.size = 720*240*2;
req_inbufs.count = N_INBUFS;

if(ioctl(fd, IPIPE_REQBUF, &req_inbufs) == -1){
    printf(“buffer allocation error.\n”);
```

```
    exit(-1);
}

/* request output buffers to be allocated */
req_outbufs.buf_type = IPIPE_BUF_OUT;
req_outbufs.size = 352*288*2;
req_outbufs.count = N_OUTBUFS;

if(ioctl(fd, IPIPE_REQBUF, &req_outbufs) == -1){
    printf("buffer allocation error.\n");
    exit(-1);
}

/* map the input buffers to user space */
bufd.buf_type = IPIPE_BUF_IN;
for(i = 0; i < N_INBUFS; i++){
    bufd.index = i;
    ioctl(fd, IPPIE_QUERYBUF, &bufd)
    inbufs[i] = mmap(0, 720*480*2, PROT_READ, MAP_SHARE, fd, bufd.offset);
}

/* map the output buffers to user space */
bufd.buf_type = IPIPE_BUF_OUT;
for(i = 0; i < N_OUTBUFS; i++){
    bufd.index = i;
    ioctl(fd, IPIPE_QUERYBUF, &bufd)
    outbufs[i] = mmap(0, 352*288*2, PROT_READ, MAP_SHARE, fd, bufd.offset);
}
```

4.3 Set Up IPIPE Parameters

```
/* setup the default parameter here */
ioctl(fd, IPIPE_SET_PARAM, NULL);
/*get the default parameters from kernel*/
ioctl(fd, IPIPE_GET_PARAM, g_param);
/*set parameters according to the required configuration*/
g_param->ipipeif_param.pack_mode = EIGHT_BIT;
g_param->ipipeif_param.hnum = I_WIDTH;
g_param->ipipeif_param.vnum = I_HEIGHT;
g_param->ipipeif_param.adofs = I_WIDTH;
g_param->ipipe_vsz = I_HEIGHT - 1;
g_param->ipipe_hsz = I_WIDTH - 1;
:
:
:
/* configure the ipipe */
ioctl(fd, IPIPE_SET_PARAM, &g_param);
```

4.4 Perform the IPIPE Operation

```
struct ipipe_convert conv;

conv.in_buff.index = conv.out_buff.index = -1; /* Set buffer index to -1 */
conv.in_buff.size = 720 * 240 * 2; /* Size of user provided input buffer */
conv.out_buff.size = 352 * 288 * 2; /* Size of user provided output buffer */
conv.in_buf.offset = inbufs[0]; /* physical address of the buffer or a user pointer */
conv.out_buf.offset = outbufs[0];

/* perform the ipipe operation */
ioctl(fd, IPIPE_START, &conv);
```

4.5 Enable RSZ0 for Resize Operation

```
/* setup the default parameter here */
ioctl(fd, IPIPE_SET_PARAM, NULL);
/*get the default parameters from kernel*/
ioctl(fd, IPIPE_GET_PARAM, g_param);
/*set parameters according to the required configuration*/
g_param->ipipeif_param.pack_mode = EIGHT_BIT;
g_param->ipipeif_param.hnum = I_WIDTH;
g_param->ipipeif_param.vnum = I_HEIGHT;
g_param->ipipeif_param.adofs = I_WIDTH;
```

```
g_param->ipipe_vsz = I_HEIGHT - 1;
g_param->ipipe_hsz = I_WIDTH - 1;
:
:
:
g_param->rsz_en[0]  = ENABLE
g_param->rsz_en[ 1] = DISABLE

/* configure the ipipe */
ioctl(fd, IPIPE_SET_PARAM, &g_param);
```

4.6 Enable RSZ1 for Resize Operation

```
/* setup the default parameter here */
ioctl(fd, IPIPE_SET_PARAM, NULL);
/*get the default parameters from kernel*/
ioctl(fd, IPIPE_GET_PARAM, g_param);
/*set parameters according to the required configuration*/
g_param->ipipeif_param.pack_mode = EIGHT_BIT;
g_param->ipipeif_param.hnum  = I_WIDTH;
g_param->ipipeif_param.vnum  = I_HEIGHT;
g_param->ipipeif_param.adofs = I_WIDTH;
g_param->ipipe_vsz = I_HEIGHT - 1;
g_param->ipipe_hsz = I_WIDTH  - 1;
:
:
:
g_param->rsz_en[0]  = ENABLE
g_param->rsz_en[ 1] = DISABLE

/* configure the ipipe */
ioctl(fd, IPIPE_SET_PARAM, &g_param);
```

4.7 Slicing of an Image

This use case is for slicing the image of horizontal size “x”. Suppose x has to be sliced to sizes “a”, ”b”, and “c”. So x = a + b + c

```
/*for slice 1*/
/* setup the default parameter here */
ioctl(fd, IPIPE_SET_PARAM, NULL);
/*get the default parameters from kernel*/
ioctl(fd, IPIPE_GET_PARAM, g_param);
/*set parameters according to the required configuration*/
g_param->ipipeif_param.pack_mode = SIXTEEN_BIT;
g_param->ipipeif_param.hnum = x;
g_param->ipipeif_param.vnum = I_HEIGHT;
g_param->ipipeif_param.adofs = x * 2;
g_param->ipipe_vsz = I_HEIGHT;
g_param->ipipe_hsz = a - 1;
:
:
:
g_param->rsz_rsc_param[0].rsz_o_hsz = a - 1;
g_param->ext_mem_param[0].rsz_sdr_oft = a * 2;
g_param->rsz_en[0] = ENABLE
g_param->rsz_en[ 1] = DISABLE

/* configure the ipipe */
ioctl(fd, IPIPE_SET_PARAM, &g_param);

/*for slice 2*/
/* setup the default parameter here */
ioctl(fd, IPIPE_SET_PARAM, NULL);
/*get the default parameters from kernel*/
ioctl(fd, IPIPE_GET_PARAM, g_param);
/*set parameters according to the required configuration*/
g_param->ipipeif_param.pack_mode = SIXTEEN_BIT;
g_param->ipipeif_param.hnum = x;
g_param->ipipeif_param.vnum = I_HEIGHT;
g_param->ipipeif_param.adofs = x * 2;
```

```
g_param->ipipe_vsz = I_HEIGHT;
g_param->ipipe_hsz = b - 1;
:
:
:
g_param->rsz_rsc_param[0].rsz_o_hsz = b - 1;
g_param->ext_mem_param[0].rsz_sdr_oft = b * 2;

g_param->rsz_en[0] = ENABLE
g_param->rsz_en[ 1] = DISABLE

/* configure the ipipe */
ioctl(fd, IPIPE_SET_PARAM, &g_param);


/*for slice 3*/
/* setup the default parameter here */
ioctl(fd, IPIPE_SET_PARAM, NULL);
/*get the default parameters from kernel*/
ioctl(fd, IPIPE_GET_PARAM, g_param);
/*set parameters according to the required configuration*/
g_param->ipipeif_param.pack_mode = SIXTEEN_BIT;
g_param->ipipeif_param.hnum = x;
g_param->ipipeif_param.vnum = I_HEIGHT;
g_param->ipipeif_param.adofs = x * 2;
g_param->ipipe_vsz = I_HEIGHT;
g_param->ipipe_hsz = c - 1;
:
:
:
g_param->rsz_rsc_param[0].rsz_o_hsz = c - 1;
g_param->ext_mem_param[0].rsz_sdr_oft = c * 2;
g_param->rsz_en[0] = ENABLE
g_param->rsz_en[ 1] = DISABLE

/* configure the ipipe */
ioctl(fd, IPIPE_SET_PARAM, &g_param);
```

5 Resizer Performance Calculation

The IPIPE has a resizer module consisting of two resizers (resizer-0 and resizer-1) and is capable of re-scaling images into various sizes ranging from x1/16 scale down to x8 scale-up in both horizontal and vertical directions. After the resizing process, the processed data is transferred to the SDRAM. This section describes how to fine tune some of the resizer parameters to achieve the desired scale performance. In this section, a typical scenario of scaling up a 88x60 image by 8x in the horizontal and vertical direction is considered to understand what are all the parameters that are to be fine tuned at the driver level to achieve this performance for one resizer. This serves as a template for calculating the values of these parameters for the desired scenario.

5.1 Resizer Performance Limit Equation

The VPFE user guide provides an equation for calculating the resizer performance limit. Listed below are the terms used in this equation:

Pixel per line of input (PPLN) — This is the number of pixel clock per horizontal line of IPIPE input.

VPSSCLK frequency (vpssclk) — When SDRAM is used as input for the IPIPE resizer, the pixel clock is derived by the IPIPE interface from the VPSSCLK. The CLOCKDIV field of the IPIPE interface configuration register (CFG) is used to divide this clock down to the desired value which forms the pixel clock (PCLK).

h_0 —

$$h_0 = \frac{\textit{horizontal size of image - 0 output} + \textit{output start position of image - 0}}{\min(1, \textit{horizontal resize ratio of image - 0})}$$

where image-0 refers to resizer-0 output image.

h_1 —

$$h_1 = \frac{\textit{horizontal size of image - 1 output} + \textit{output start position of image - 1}}{\min(1, \textit{horizontal resize ratio of image - 1})}$$

where image-1 refers to resizer-1 output image.

r_0 — r_0 = ceil((vertical resize ratio of image-0) / 2)

r_1 — r_1 = ceil((vertical resize ratio of image-1) / 2)

overhead (o) — o = 100 x (r_0 + r_1)

The vertical resize ratio for image-0 and image-1 should satisfy the following equation:

$$ppln \times (vpssclk / pclk) > h_0 \times r_0 + h_1 \times r_1 + 0$$

$$where\ vpssclk / pclk = clockdiv$$

$$\Rightarrow$$

$$ppln \times clockdiv > h_0 \times r_0 + h_1 \times r_1 + 0$$

Actual performance of the resize output is also limited by the bandwidth of the attached SDRAM and usage of this bandwidth by other entities in the system. With the Linux kernel running, this additional overhead is measured to be about 1.45 times the value of h_0 x r_0 + h_1 x r_1 + o calculated above.

$$\Rightarrow$$

$$ppln \times clockdiv > 1.45 \times (h_0 \times r_0 + h_1 \times r_1 + 0)$$

5.2 Calculating PPLN for Resizing of 88x60 to 704x480

The following parameters are required to be adjusted for getting the desired resizer performance. In this example, the parameter values required to obtain 8x horizontal and vertical zoom are calculated. The following fields in the ipipeif structure are involved in the performance calculation:

```
struct ipipeif {
    ipipeif_clkdiv clk_div;     /*clock divisor*/
    unsigned int glob_hor_size; /*global frame horizontal size*/
    unsigned int glob_ver_size; /*global frame vertical size*/
};
```

where clk_div is the CLOCKDIV field of the IPIPE interface configuration register (CFG), glob_hor_size is the PPLN, and glob_ver_size is the Lines Per Frame (LPFR) register values. The values configured for these must satisfy the equations detailed in Section 5.1.

For this example, the input image size is 88x60 and output image size is 704 x 480. This requires a scale factor of 8x in both the horizontal and vertical direction. The VPFE uses a VPSSCLK of 108 Mhz and thus $PCLK = \frac{108}{CLOCKDIV\ MHz}$. For clockdiv = 6, $PCLK = \frac{108}{6\ MHz} = 18\ MHz$.

In this example, it is assumed that the output image start position is 0 and only one resizer is used. The LPFR is set to the number of lines in the input image plus + 10 (as per VPFE user guide) = 60 + 10 = 70.

$$1.45 \times (h_0 \times r_0 + h_1 \times r_1 + 0)$$

$$h_0 = \frac{horizontal\ size\ of\ image - 0\ output + output\ start\ position\ of\ image - 0}{\min(1, horizontal\ resize\ ratio\ of\ image - 0)} = \frac{704 + 0}{\min(1,8)} = \frac{704}{1} = 704$$

$$r_0 = ceil((vertical\ resize\ ratio\ of\ image - 0)/2) = ceil\ (8/2) = 4$$

$$Overhead, o = 100 \times (r_0 + r_1) = 100 \times 4 = 400$$

Since you are not using resizer-1, h_1 = 0, r_1 =0

$$1.45 \times ((704 \times 4) + 400) = 4664\ (rounded)$$

$$PPLN\ for\ CLOCKDIV\ of\ 6\ should\ be\ > \frac{4664}{CLOCKDIV} > \frac{4664}{6} > 777$$

So use a PPLN value of 778, CLOCKDIV value of 6 to get 8x zoom in the horizontal and vertical direction. With these values the following calculation can be used to obtain the expected IPIPE processing time at the hardware.

Expected processing time is

$$\frac{PPLN \times LPFR}{PCLK} = \frac{777 \times 70}{18 \times 1000000} = 3.02\ msec$$

The IPIPE Driver is instrumented to measure the processing time at the hardware for doing IPIPE for the above scenario and the measured values are given in the table below for 10 frames. The time is measured just before enabling IPIPE processing and in the ISR after the frame is written to SDRAM.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
ipipe start (usecs)	862888	896150	929520	962894	996272	29628	62990	96384	129728	163101
ipipe_stop (usecs)	865821	899052	932425	965794	999172	32532	65889	99281	132631	165999
IPIPE processing time (usecs)	2933	2902	2905	2900	2900	2904	2899	2897	2903	2898
measured average processing time – msec	2.9041									

The Application is instrumented to measure the processing time at the Kernel for the IPIPE_START ioctl call (Time measured just before and after the IOCTL call) and the values are given in the table below for 10 frames.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
ipipe start (usecs)	392234	425530	458882	492267	525621	559000	592353	625714	659088	692461
ipipe_stop (usecs)	396008	429284	462655	496033	529402	562781	596128	629478	662868	696215
IPIPE processing time (usecs)	3774	3754	3773	3766	3781	3781	3775	3764	3780	3754
measured average processing time – msec	3.777									

These measurements are done with the Kernel in the RT Pre-emption mode.

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