KeyStone Architecture
Chip Interrupt Controller (CIC)

User Guide

Literature Number: SPRUGW4A
March 2012

Release History

Release	Date	Description/Comments
A	March 2012	• Added Control Register section (Page 2-3) • Added Global Prioritized Index Register section (Page 2-8) • Added Host Interrupt Prioritized Index Registers section (Page 2-12) • Corrected "system interrupt" to "channel" in the description of Host Interrupt Map Register (Page 2-12) • Added "Interrupt Serice Sequence" section (Page 1-5) • Added disable/re-enable steps in the ISR function of the example code. (Page 2-11) • Changed TPCC to EDMA3CC (Page 1-2) • Changed chip interrupt controller name from INTC to CIC (Page 1-2)
SPRUGW4	September 2011	Initial release

Contents

List of Tables

List of Figures

Preface

About This Manual

This document describes the functionality, operational details, and programming information for the KeyStone Architecture Chip Interrupt Controller (CIC).

Notational Conventions

This document uses the following conventions:

- Commands and keywords are in **boldface** font.
- Arguments for which you supply values are in *italic* font.
- Terminal sessions and information the system displays are in `screen font`.
- Information you must enter is in **`boldface screen font`**.
- Elements in square brackets ([ ]) are optional.

Notes use the following conventions:

Note—Means reader take note. Notes contain helpful suggestions or references to material not covered in the publication.

The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

CAUTION—Indicates the possibility of service interruption if precautions are not taken.

WARNING—Indicates the possibility of damage to equipment if precautions are not taken.

Related Documentation from Texas Instruments

C66x CorePac User Guide	SPRUGW0
C66x CPU and Instruction Set Reference Guide	SPRUGH7

Trademarks

TMS320C66x and C66x are trademarks of Texas Instruments Incorporated.

All other brand names and trademarks mentioned in this document are the property of Texas Instruments Incorporated or their respective owners, as applicable.

Chapter 1

Introduction

1.1 Overview

The KeyStone Architecture has many peripherals and a large number of event sources. The use of events is completely dependent on a user's specific application, which, therefore, drives a need for maximum flexibility in which event sources are used in the system. It is also completely up to software control as to how interrupts or events are serviced. Both the EDMA3 channel controllers (EDMA3CC) and the C66x CorePac are capable of receiving events directly. However, the number of accepted events for each EDMA3CC and C66x CorePac is limited.

A KeyStone device can have hundreds of events. Therefore, some of these events need to be aggregated at the chip level through the interrupt controller (the chip-level interrupt controller (CIC)— not the interrupt controller inside a C66x CorePac) before they are routed to the EDMA3CC and C66x CorePacs. To achieve the requirement, some new chip-level interrupt controllers are added to the SoC. The CIC takes chip-level events (system events) and generates event inputs to the EDMA3CC and C66x CorePac by combining and/or selecting those chip-level events.

The C66x CorePac internal interrupt controller allows a large number of system events to be used by C66x CorePac and any of these system events to be routed to up to 12 maskable interrupts. It also lets any of these system events be grouped together for a single exception input to the CorePac and allows any of these system events to be an AEG trigger. The EDMA3CC supports up to 64 events and there is event detection logic that recognizes the EDMA3CC system events.

This document does not include details about the internal interrupt controller of the C66x CorePac. For operation of the internal interrupt controller, see the *C66x CorePac Reference Guide* (SPRUGW0).

1.2 Terminology Used in This Document

Table 1-1 defines the important acronyms used in this document.

Table 1-1 Terminology

Acronym	Definition
AEG	Advanced event trigger
CIC	Chip interrupt controller
ISR	Interrupt service routine
MMR	Memory mapped register
NMI	Non-maskable interrupt
QM	Hardware queue manager
QMSS	Queue manager subsystem
EDMA3CC	EDMA3 channel controller
End of Table 1-1	

1.3 Chip-level Interrupt Controller Operation

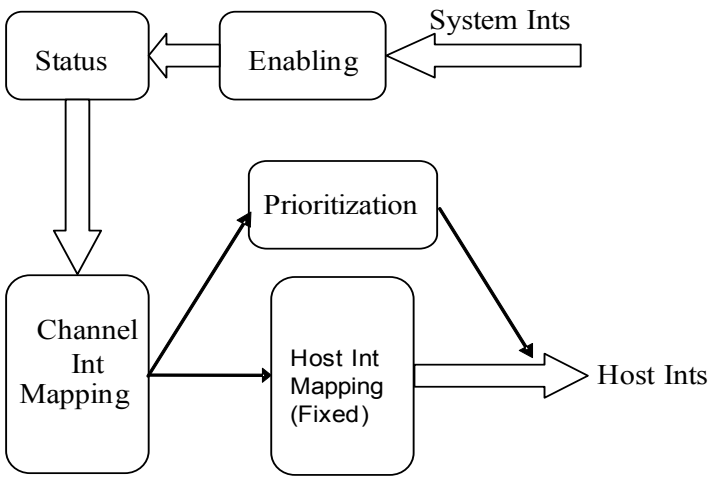
The CIC module controls the system interrupt (interrupts from peripheral) mapping to the host (CorePac, ARM core, etc. inside KeyStone Architecture devices) interrupt interface. System interrupts are generated by the peripherals. The CIC receives the system interrupts and maps them to internal channels. The channels are used to group interrupts together. These channels are then mapped onto the interface that is typically a smaller number of host interrupts. The configuration of the CIC in KeyStone Architecture devices has a fixed one-to-one mapping between channels and host interrupts.

Some terminology needs to be clarified with all the types of interrupts used in a system. First, interrupts generated by an peripheral are called peripheral interrupts. Second, peripheral interrupts need to be manipulated such as stretching or synchronizing to be ready for system usage and are called system interrupts. The system interrupts are those that are usable by interrupt controllers and are always pulse interrupts (active for a single cycle upon becoming pending) in KeyStone devices. System interrupts can be individual for a single core or can be shared by multiple cores.

The CIC encompasses several functions to process the system interrupts and prepare them for the host interface. These functions are:

- Enabling
- Status
- Channel mapping (programmable)
- Host interrupt mapping (fixed)
- Prioritization

Figure 1-1 CIC Block Diagram



1.3.1 Enabler

This stage of the CIC is to enable the system interrupts based on programmed settings. For each system interrupt, the SoC uses the enable register programmable by software to enable or disable system interrupts. Those that are disabled will not generate any host interrupts.

1.3.2 Interrupt Status

The next stage of the CIC is to capture which system interrupts are pending. This status is captured into a status register that is readable by the system to identify the system interrupts that are pending and/or enabled. The pending status reflects whether the system interrupt occurred since the last time the status register bit was cleared. Each bit in the status register is individually clearable. Status is also cleared when the specific status is cleared. Software can also set interrupts without hardware triggering.

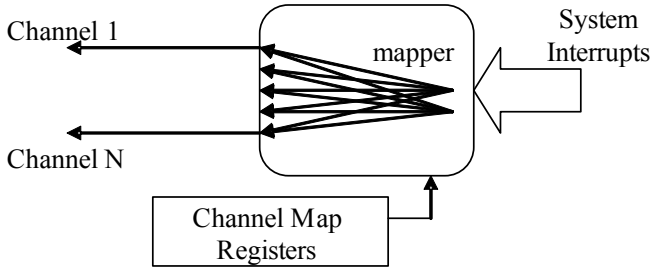
There are two kinds of status: raw status and enabled status. Raw status is the pending status of the system interrupt without regard to the enable bit for the system interrupt. Enabled status is the pending status of the system interrupts with the enable bits active. When the enable bit is inactive the enabled status will always be inactive. Only the enabled status will be used in later stages in the CIC, while raw status is provided for software or debug usage.

1.3.3 Channel Mapping

The next stage of the CIC is to map the system interrupts that have been enabled to internal channels. Channels are used to group the system interrupts into a smaller number of interrupt inputs that can be given to a host interface. When multiple system interrupts are mapped to the same channel their interrupts are OR'd together so that when either is active the output is active. Figure 1-2 shows the channel mapping functions (please note that the interrupt cannot be mapped to multiple channels but the connection is selected by the channel map register value).

The mapping of the system interrupts to channels is programmable. Each system interrupt has a register to define its mapping.

Figure 1-2 Channel Mapping Block Diagram



1.3.4 Host Interrupt Mapping

The next stage of the CIC is host mapping. This stage maps the defined number of channels to the host interrupts.

The mapping of the channels to host interrupts is fixed (one-to-one mapping). Each channel has a register to define its host interrupt and the register is read only.

1.3.5 Prioritization

The next stage of the CIC is prioritization. Prioritization may not be implemented in all of the CICs in KeyStone devices. Please refer to the device-specific data manual for details.

If multiple interrupts feed into a single channel/host interrupt (since the one-to-one mapping of channel to host interrupt), it is necessary to prioritize between all the system interrupts to decide on a single system interrupt to handle. The CIC provides hardware to perform this prioritization with a given scheme so that software does not have to do this. The system interrupt with index number 0 has the highest priority and system interrupt with highest index has the lowest priority. So the single host interrupt level prioritization picks the active system interrupt with lowest index. This prioritized system interrupt for the host interrupt is stored in the Host Interrupt Prioritized Index Register that the software can read to quickly get the system interrupt to handle.

The prioritization is done for each host interrupt as well as across all the host interrupts and can be read in separate registers. The Global Prioritized Index Register is for the prioritization across all the host interrupts. This is only useful when multiple host interrupts are not really used, or there is no priority between the host interrupts, or the interrupt service routine (ISR) can only service one interrupt. In these cases the global priority works across the entire CIC and all host interrupts and system interrupts. This prioritization is based on the channel index. Channel 0 has the highest priority and the channel with highest index number has the lowest priority. So the highest priority channel with a pending interrupt is chosen and then the highest priority system interrupt is chosen (as chosen for the Host Interrupt Priority Index Register by system interrupt index). In this case the mapping of system interrupts to channel affects the prioritization so the software can setup up system interrupts into priorities (or channels) and then the priority of the interrupts inside the channel is less important.

There is a prioritization hold mode in the Control Register that can be enabled to lock the prioritized index register to hold the value once a read of the register is performed. This allows software to lock the value after the initial read so that it can be read again without being updated if new, higher priority system interrupts are pending. The lock is per host interrupt and will not affect other host interrupt registers. The lock is ended when software does one of the following: writes to the Host Interrupt Prioritized Index Register (it is normally read only), writes to the Host Interrupt Enable Indexed Set Register for this host interrupt, writes to the Host Interrupt Enable Indexed Clear Register for this host interrupt, or writes to the Host Interrupt Enable Register and sets this host interrupt's enable bit. When the prioritization hold mode is not enabled then the Prioritized Indexed Register will continually update whenever new system interrupts are pending.

1.3.6 Interrupt Service Sequence

One simple example of the interrupt service sequence is shown in Table 1-2. The interrupt service routine (ISR) executes when an interrupt is taken and services the pending interrupts before exiting.

Table 1-2 Interrupt Service Sequence

Step	Description	Example actions for C66x
1	Receive the interrupt	Corresponding bit set in IFR register (by hardware) GIE bit cleared (by hardware) Execution from interrupt servicing table (IST) begins, entering to the ISR (by hardware) Corresponding bit cleared in IFR register (by hardware)
2	Disable the host interrupt	Disable the corresponding host interrupt output in the CIC Host Interrupt Enable Register
3	Determine the exact interrupt and clear the status of the system interrupt (one of the two methods)	Read the Prioritized Index Register (if prioritization is implemented and decided to be used) Clear the system interrupt flag in CIC System Interrupt Status Clear Register
		Read the System Interrupt Status Enabled/Clear Register (for bit flags) Clear the system interrupt flag in CIC System Interrupt Status Clear Register
4	Service the interrupt	Execute the user's code for servicing the interrupt
5	Re-enable the host interrupt	Enable the corresponding host interrupt output in the CIC Host Interrupt Enable Register
6	Return from interrupt	GIE bit set (by hardware) Return from ISR to the user's application

In Step 3, if prioritization is used, either the Host Interrupt Prioritized Index Register or the Global Prioritized Index Register could be checked. There is no need to check both registers. The ISR setup will determine which to use and once one is read it will give the system interrupt index to service. It can either read the Host Interrupt Prioritized Index Register if it knows which host interrupt was triggered and it uses that priority in the host, or it can use the Global Prioritized Index Register if it is not using any host priority.

If the host can accept other host interrupts while processing the ISR then disabling all the host interrupts (in Step 2) would be suggested, assuming the ISR only wants to process one interrupt at a time (the reason it is using the Global Prioritized Index Register). There is a global control bit (bit[0] in Global Enable Register) to enable/disable all the host interrupts to make the process easier (the individual host interrupt enables are maintained as well).

Note—When the CIC system interrupt flag is being cleared in ISR, if the same system interrupt happens again at the same cycle, the CIC system interrupt flag will not be cleared. By adding Step 2 (disabling the host interrupt) and Step 5 (re-enabling the host interrupt) in ISR will make sure to generate another interrupt to the host after exiting from the current ISR. So the new interrupt received during the ISR will not be missed. Please refer to the CSL example in section 2.3 "Interrupt Servicing" for the sequence as well.

Chapter 2

Registers

2.1 CIC Register Offsets

This section lists offsets for the CIC registers. Some registers may not exist based on the number of enabled events. See the device-specific data manual for more specific offsets.

Table 2-1 CIC Register Offsets

Address Offset	Register	Section
0x000	Revision Register	Section 2.2.1
0x004	Control Register	Section 2.2.2
0x008 – 0x00C	Reserved	
0x010	Global Enable Register	Section 2.2.3
0x014 – 0x01C	Reserved	
0x020	System Interrupt Status Indexed Set Register	Section 2.2.4
0x024	System Interrupt Status Indexed Clear Register	Section 2.2.5
0x028	System Interrupt Enable Indexed Set Register	Section 2.2.6
0x02C	System Interrupt Enable Indexed Clear Register	Section 2.2.7
0x030	Reserved	
0x034	Host Interrupt Enable Indexed Set Register	Section 2.2.8
0x038	Host Interrupt Enable Indexed Clear Register	Section 2.2.9
0x03C – 0x07C	Reserved	
0x080	Global Prioritized index Register	Section 2.2.10
0x084 – 0x1FC	Reserved	
0x200 – 0x27C	System Interrupt Status Raw/Set Registers	Section 2.2.11
0x280 – 0x2FC	System Interrupt Status Enabled/Clear Registers	Section 2.2.12
0x300 – 0x37C	System Interrupt Status Enable Set Registers	Section 2.2.13
0x380 – 0x3FC	System Interrupt Status Enable Clear Registers	Section 2.2.14
0x400 – 0x7FC	Channel Map Registers	Section 2.2.15
0x800 – 0x8FC	Host Map Registers	Section 2.2.16
0x900 – 0xCFC	Host Interrupt Prioritized Index Registers	Section 2.2.17
0xD00 – 0x14FC	Reserved	
0x1500 – 0x151C	Host Interrupt Enable Registers	Section 2.2.18
0x1520 – 0x2FFC	Reserved	
End of Table 2-1		

2.2 CIC Registers

This section provides detailed information on the peripheral constant registers.

2.2.1 Revision Register

The Revision Register (Figure 2-1) contains the ID and revision information for the device. The offset is 0x0.

Figure 2-1 Revision Register

31 30	29 28	27 16	15 11	10 8	7 6	5 0
SCHEME	Reserved	FUNCTION	REVRTL	REVMAJ	REVCUSTOM	REVMIN
R-1	R-0	R-0xe82	R-Device Specific	R-1	R-0	R-0

Legend: R = Read only; -*n* = value after reset

Table 2-2 Revision Register Field Descriptions

Bits	Field	Description
31-30	SCHEME	Scheme that this register is compliant with.
29-28	Reserved	Reads return 0 and writes have no effect.
27-16	FUNCTION	Function
15-11	REVRTL	RTL revision
10-8	REVMAJ	Major revision
7-6	REVCUSTOM	Custom revision
5-0	REVMIN	Minor revision
End of Table 2-2		

2.2.2 Control Register

The Control Register (Figure 2-2) holds global control parameters. The offset is 0x4. *This register is only available only when prioritization is implemented*

Figure 2-2 Control Register

31 5	4	3 0
Reserved	PRIORITY_HOLD	Reserved
R-0	R/W-1	R-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-3 Control Register Field Descriptions

Bits	Field	Description
31-5	Reserved	Reads return 0 and writes have no effect.
4	PRIORITY_HOLD	Enable priority holding mode. 0 = Priority holding disabled, Prioritized Index Registers will continually update 1 = Priority holding enabled, Prioritized Index Registers will hold their value after the first read and release the hold after either is written or Host Interrupt is set or cleared as described in Section 1.3.5
3-0	Reserved	Reads return 0 and writes have no effect.
End of Table 2-3		

2.2.3 Global Enable Register

The Global Enable Register (Figure 2-3) enables all the host interrupts at once. Individual host interrupts are still enabled or disabled from their individual enables and are not overridden by the global enable. The offset is 0x010.

Figure 2-3 Global Enable Register (GLOBAL_ENABLE_HINT_REG)

31	1	0
Reserved		ENABLE
R-0		R/W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-4 Global Enable Register Field Descriptions

Bits	Field	Description
31-1	Reserved	Reads return 0 and writes have no effect.
0	ENABLE	This field enables or disables all the host interrupts at once. Reads return the current global enable value. When written: 0 = Disable all the host interrupts 1 = Enable all the host interrupts

2.2.4 System Interrupt Status Indexed Set Register

The System Interrupt Status Indexed Set Register (Figure 2-4) allows setting the status of an interrupt. The interrupt to set is the index value written. This sets the System Interrupt Status Raw/Set Register bit of the given index. The offset is 0x020.

Figure 2-4 System Interrupt Status Indexed Set Register (STATUS_SET_INDEX_REG)

31	10	9	0
Reserved		INDEX	
R-0		R/W-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-5 System Interrupt Status Indexed Set Register Field Descriptions

Bits	Field	Description
31-10	Reserved	Reads return 0 and writes have no effect.
9-0	INDEX	This field allows setting the status of an interrupt. Reads return 0. Writes set the status of the interrupt given in the index value. 0 = Set the status of the system interrupt 0 1h = Set the status of the system interrupt 1 ... 3FFh = Set the status of the system interrupt 1023

2.2.5 System Interrupt Status Indexed Clear Register

The System Interrupt Status Indexed Clear Register (Figure 2-5) allows clearing the status of an interrupt. The interrupt to clear is the index value written. This clears the System Interrupt Status Raw/Set Register bit of the given index. The offset is 0x024.

Figure 2-5 System Interrupt Status Indexed Clear Register (STATUS_CLR_INDEX_REG)

31	10	9	0
Reserved		INDEX	
R-0		R/W-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-6 System Interrupt Status Indexed Clear Register Field Descriptions

Bits	Field	Description
31-10	Reserved	Reads return 0 and writes have no effect.
9-0	INDEX	This field allows clearing the status of an interrupt. Reads return 0. Writes clear the status of the interrupt given in the index value. 0 = Clear the status of the system interrupt 0 1h = Clear the status of the system interrupt 1 ... 3FFh = Clear the status of the system interrupt 1023

2.2.6 System Interrupt Enable Indexed Set Register

The System Interrupt Enable Indexed Set Register (Figure 2-6) allows enabling an interrupt. The interrupt to enable is the index value written. This sets the System Interrupt Enable Set Register bit of the given index. The offset is 0x028.

Figure 2-6 System Interrupt Enable Indexed Set Register (ENABLE_SET_INDEX_REG)

31	10	9	0
Reserved		INDEX	
R-0		R/W-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-7 System Interrupt Enable Indexed Set Register Field Descriptions

Bits	Field	Description
31-10	Reserved	Reads return 0 and writes have no effect.
9-0	INDEX	This field allows enabling an interrupt. Reads return 0. Writes set the enable of the interrupt given in the index value. 0 = Enable the system interrupt 0 1h = Enable the system interrupt 1 ... 3FFh = Enable the system interrupt 1023

2.2.7 System Interrupt Enable Indexed Clear Register

The System Interrupt Enable Indexed Clear Register (Figure 2-7) allows disabling an interrupt. The interrupt to disable is the index value written. This clears the Enable Register bit of the given index. The offset is 0x02C.

Figure 2-7 System Interrupt Enable Indexed Clear Register (ENABLE_CLR_INDEX_REG)

31	10	9	0
Reserved		INDEX	
R-0		R/W-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-8 System Interrupt Enable Indexed Clear Register Field Descriptions

Bits	Field	Description
31-10	Reserved	Reads return 0 and writes have no effect.
9-0	INDEX	This field allows disabling an interrupt. Reads return 0. Writes clear the enable of the interrupt given in the index value. 0 = Disable the system interrupt 0 1h = Disable the system interrupt 1 ... 3FFh = Disable the system interrupt 1023

2.2.8 Host Interrupt Enable Indexed Set Register

The Host Interrupt Enable Indexed Set Register (Figure 2-8) allows enabling a host interrupt output. The host interrupt to enable is the index value written. This enables the host interrupt output or triggers the output again if already enabled. The offset is 34h.

Figure 2-8 Host Interrupt Enable Indexed Set Register (HINT_ENABLE_SET_INDEX_REG)

31	10	9	0
Reserved		INDEX	
R-0		R/W-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-9 Host Interrupt Enable Indexed Set Register Field Descriptions

Bits	Field	Description
31-10	Reserved	Reads return 0 and writes have no effect.
9-0	INDEX	This field allows enabling a host interrupt output. Reads return 0. Writes set the enable of the host interrupt given in the index value. 0 = Enable or trigger the host interrupt output 0 1h = Enable or trigger the host interrupt output 1 ... FFh = Disable the host interrupt output 255 Others = Reserved.

2.2.9 Host Interrupt Enable Indexed Clear Register

The Host Interrupt Enable Indexed Clear Register (Figure 2-9) allows disabling a host interrupt output. The host interrupt to disable is the index value written. This disables the host interrupt output. The offset is 0x038.

Figure 2-9 Host Interrupt Enable Indexed Clear Register (HINT_ENABLE_CLR_INDEX_REG)

31	10	9	0
Reserved		INDEX	
R-0		R/W-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-10 Host Interrupt Enable Indexed Clear Register Field Descriptions

Bits	Field	Description
31-10	Reserved	Reads return 0 and writes have no effect.
9-0	INDEX	This field allows disabling a host interrupt output. Reads return 0. Writes clear the enable of the host interrupt given in the index value. 0 = Disable the host interrupt output 0 1h = Disable the host interrupt output 1 ..... FFh = Disable the host interrupt output 255 Others = Reserved.

2.2.10 Global Prioritized Index Register

The Global Prioritized Index Register (Figure 2-10) shows the interrupt number of the highest priority interrupt pending across all the host interrupts. The offset is 0x080. *This register is available only when prioritization is implemented.*

Figure 2-10 Global Prioritized Index Register

31	30 10	9 0
NONE	Reserved	PRI_INDEX
R-1	R-0	R-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-11 Global Prioritized Index Register Field Descriptions

Bits	Field	Description
31	NONE	No interrupts are pending. Can be used by host to test if no interrupts are pending. 0 = Interrupts are pending 1 = No interrupts are pending
30-10	Reserved	Reads return 0 and writes have no effect.
9-0	PRI_INDEX	The currently highest priority interrupt index pending across all the host interrupts.

2.2.11 System Interrupt Status Raw/Set Registers

The System Interrupt Status Raw/Set Registers (Figure 2-11) show the pending enabled status of the system interrupts. Software can write to the Status Set Registers to set a system interrupt without a hardware trigger. There is one bit per system interrupt. If the number of system interrupts is greater than 32, then multiple registers are used for each 32 system interrupts. The system interrupts number is device specific and the maximum is 1024. The maximum registers number is 32 (=1024/32). The offset is from 0x200 to 0x27C.

Figure 2-11 System Interrupt Status Raw/Set Registers (RAW_STATUS_REGx)

31 N	N-1 0
Reserved	RAW_STATUS
R-0	R/W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-12 System Interrupt Status Raw/Set Registers Field Descriptions

Bits	Field	Description
31-N	Reserved	Reads return 0 and writes have no effect.
N-1:0	RAW_STATUS	System interrupt raw status and setting. Reads return the raw status. When written (per bit position): 0 = No effect 1 = Set the status of the system interrupt

2.2.12 System Interrupt Status Enabled/Clear Registers

The System Interrupt Status Enabled/Clear Registers (Figure 2-12) show the pending enabled status of the system interrupts. Software can write to the Status Clear Registers to clear a system interrupt after it has been serviced. If a system interrupt status is not cleared, then another host interrupt may not be triggered or another host interrupt may be triggered incorrectly. There is one bit per system interrupt. If the number of system interrupts is greater than 32, then multiple registers are used for each 32 system interrupts. The system interrupts number is device specific and the maximum is 1024. The maximum registers number is 32 (=1024/32). The offset is from 0x280 to 0x2FC.

Figure 2-12 System Interrupt Status Enabled/Clear Registers (ENA_STATUS_REGx)

31 ... N	N-1 ... 0
Reserved	ENABLED_STATUS
R-0	R/W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-13 System Interrupt Status Enabled/Clear Registers Field Descriptions

Bits	Field	Description
31-N	Reserved	Reads return 0 and writes have no effect.
N-1:0	ENABLED_STATUS	System interrupt enabled status and clearing. Reads return the enabled status (before enabling with the System Interrupt Enable Set Registers). When written (per bit position): 0 = No effect 1 = Clear the status of the system interrupt.

2.2.13 System Interrupt Enable Set Registers

The System Interrupt Enable Set Registers (Figure 2-13) enable system interrupts to trigger outputs. System interrupts that are not enabled do not interrupt the host. There is a bit per system interrupt. If the number of system interrupts is greater than 32, then multiple registers are used for each 32 system interrupts. The system interrupts number is device specific and the maximum is 1024. The maximum registers number is 32 (=1024/32). The offset is from 0x300 to 0x37C.

Figure 2-13 System Interrupt Enable Set Register (ENABLE_REGx)

31 N	N-1 0
Reserved	ENABLE
R-0	R/W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-14 System Interrupt Enable Set Register Field Descriptions

Bits	Field	Description
31-N	Reserved	Reads return 0 and writes have no effect.
N-1:0	ENABLE	System interrupt enables. Read returns the enable value: 0 = Interrupt disabled 1 = Interrupt enabled When written (per bit position): 0 = No effect 1 = Enable the interrupt

2.2.14 System Interrupt Enable Clear Registers

The System Interrupt Enable Clear Registers (Figure 2-14) disable system interrupts to map to channels. System interrupts that are not enabled do not interrupt the host. There is one bit per system interrupt. If the number of system interrupts is greater than 32, then multiple registers are used for each 32 system interrupts. The system interrupts number is device specific and the maximum is 1024. The maximum registers number is 32 (=1024/32). The offset is from 0x380 to 0x3FC.

Figure 2-14 System Interrupt Enable Clear Register (ENABLE_CLR_REGx)

31 N	N-1 0
Reserved	ENABLE
R-0	R/W-0

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-15 System Interrupt Enable Clear Register Field Descriptions

Bits	Field	Description
31-N	Reserved	Reads return 0 and writes have no effect.
N-1:0	ENABLE	System interrupt disables. Read returns the enable value: 0 = Interrupt disabled 1 = Interrupt enabled When written (per bit position): 0 = No effect 1 = Disable the interrupt

2.2.15 Channel Map Registers

The Channel Map Registers (Figure 2-15) define the channel for each system interrupt. There is one register per four system interrupts. The system interrupts number is device specific and the maximum is 1024. The maximum registers number is 256 (=1024/4). The offset is from 0x400 to 0x7FC.

Figure 2-15 Channel Interrupt Map Register (CH_MAP_REGx)

31 24	23 16	15 8	7 0
CH3_MAP	CH2_MAP	CH1_MAP	CH0_MAP
R/W-0	R/W-0	R/W-0	R/W-0

Legend: R = Read only; -*n* = value after reset

Table 2-16 Channel Interrupt Map Register Field Descriptions

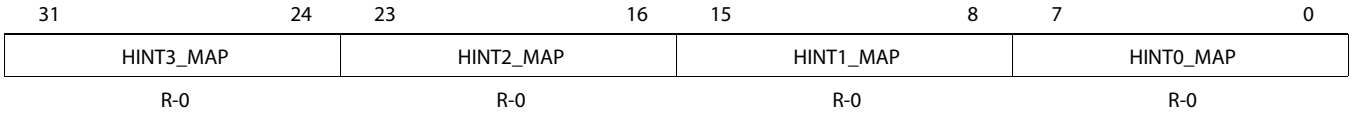
Bits	Field	Description
31-24	CH3_MAP	Set the channel for the system interrupt N + 3[1]
23-16	CH2_MAP	Set the channel for the system interrupt N + 2
15-8	CH1_MAP	Set the channel for the system interrupt N + 1
7-0	CH0_MAP	Set the channel for the system interrupt N

1. The 8 bits are used to represent the index of the mapped output events. It supports a maximum 256 events. Some bits may not be used because of the number of available output events. Unused bits are 0 and can not be set.

2.2.16 Host Interrupt Map Registers

The Host Interrupt Map Registers (Figure 2-16) define the host interrupt for each channel. There is one register per four channels. It is read-only because the channel-to-host mapping is fixed. The channel number is device specific and the maximum is 256. The maximum registers number is 64 (=256/4). The offset is from 0x800 to 0x8FC.

Figure 2-16 Host Interrupt Map Register (HINT_MAP_REGx)



Legend: R = Read only; -*n* = value after reset

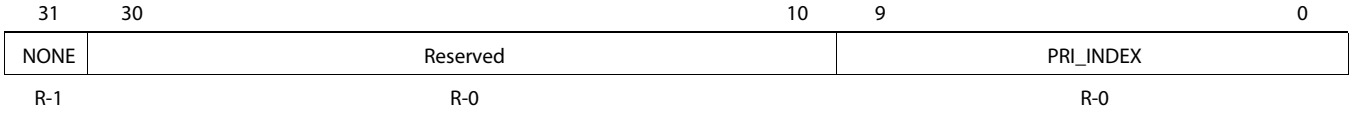
Table 2-17 Host Interrupt Map Register Field Descriptions

Bits	Field	Description
31-24	HINT3_MAP	Set the host interrupt for channel N + 3
23-16	HINT2_MAP	Set the host interrupt for channel N + 2
15-8	HINT1_MAP	Set the host interrupt for channel N + 1
7-0	HINT0_MAP	Set the host interrupt for channel N

2.2.17 Host Interrupt Prioritized Index Registers

The Host Interrupt Prioritized Index Registers (Figure 2-17) show the highest priority current pending interrupt for the host interrupt. There is one register per host interrupt. The host interrupt number is device specific and the maximum is 256. The maximum registers number is 256. The offset is from 0x900 to 0xCFC. *These registers are available only when prioritization is implemented.*

Figure 2-17 Host Interrupt Prioritized Index Register



Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-18 Host Interrupt Prioritized Index Register Field Descriptions

Bits	Field	Description
31	NONE	No interrupts are pending. Can be used by host to test if no interrupts are pending. 0 = Interrupts are pending 1 = No interrupts are pending
30-10	Reserved	Reads return 0 and writes have no effect.
9-0	PRI_INDEX	The interrupt number of the highest priority pending interrupt for this host interrupt.

2.2.18 Host Interrupt Enable Registers

The Host Interrupt Enable Registers (Figure 2-18) enable or disable individual host interrupts. These work separately from the global enables. There is one bit per host interrupt. These bits are updated when writing to the Host Interrupt Enable Index Set and Host Interrupt Enable Index Clear registers. The host interrupt number is device specific and the maximum is 256. The maximum registers number is 8 (=256/32). The offset is from 0x1500 to 0x151C.

Figure 2-18 Host Interrupt Enable Register (ENABLE_HINT_REGx)

31	N	N-1	0
Reserved		ENABLE	
R-0		R/W-0	

Legend: R = Read only; R/W = Read/Write; -*n* = value after reset

Table 2-19 Host Interrupt Enable Register Field Descriptions

Bits	Field	Description
31-N	Reserved	Reads return 0 and writes have no effect.
N-1:0	ENABLES	This field enables the host interrupts. When written (per bit position): 0 = Disable the host interrupt 1 = Enable the host interrupt

2.3 Interrupt Servicing

The following code is an example sequence for receiving and handling an interrupt using CSL. Note CSL_intcxxx is for the C66x CorePac interrupt controller and CSL_cpintcxxx is for the chip-level interrupt controller. The example shows how to enable queue pending signal from Queue Manager Subsystem (QMSS) in the Multicore Navigator and map them to core interrupt.

Please note that, in the following example, all the queue numbers and event numbers are based on the C6670 device. Please see the device-specific data manual for the desired values.

Also note that, in C6670, the queue pending signals are the secondary events, which will be routed to the chip-level interrupt controller (CIC0) first and then be mapped to the CorePac interrupt controller. Then the event will be mapped to the core interrupt from the CorePac interrupt controller.

Example 2-1

```
- - - - - - - - - - - - - - - - - - - - - - - - - -
#include <stdio.h>
#include <ti/csl/csl.h>
#include <ti/csl/tistdtypes.h>
#include <ti/csl/csl_error.h>
#include <ti/csl/csl_intc.h>
#include <ti/csl/csl_cpIntcAux.h>
#include <ti/csl/csl_chip.h>
#include <ti/csl/soc.h>


#define QPEND_IDX                       5
#define FIRST_QPEND_QUEUE               662
#define FIRST_QPEND_CIC0_OUT_EVENT      56


/* Function declarations */
void init_interrupt_controllers();
void setup_qpend_interrupt(Uint16 qnum,Uint16 CorePac_event);


/* Global variable declarations */
CSL_IntcContext Intcontext;             //For CorePac INTC
CSL_IntcObj intcQmss;                   //For CorePac INTC
CSL_IntcHandle hIntcQmss[6];            //For CorePac INTC
CSL_IntcEventHandlerRecord Record[6];   //For CorePac INTC
CSL_CPINTC_Handle cphnd;                //For chip-level CIC


Int16 sys_event;
Int16 host_event;

void main(void)
{
    ......

    /* Init Interrupt Controllers including chip-level CIC and CorePac INTC.*/
    init_interrupt_controllers();

    /* Setup ISR for QMSS que pend queue */
    /* Queue pending signals are from 662 to 671 to CIC0 */
    setup_qpend_interrupt(FIRST_QPEND_QUEUE, FIRST_QPEND_CIC0_OUT_EVENT);

    ......
}


/* Perform one-time initialization of chip-level and CorePac
 * interrupt controllers.
 */
void init_interrupt_controllers()
{
    CSL_IntcGlobalEnableState state;

    /* Setup the global Interrupt */
```

```
    Intcontext.numEvtEntries = 6;
    Intcontext.eventhandlerRecord = Record;
    CSL_intcInit(&Intcontext);

    /* Enable NMIs  */
    CSL_intcGlobalNmiEnable();

    /* Enable Global Interrupts  */
    CSL_intcGlobalEnable(&state);

    /* Initialize the chip level CIC CSL handle. */
    cphnd = CSL_CPINTC_open(0);
    if (cphnd == 0)
    {
        printf("Cannot initialize CPINTC\n");
        return;
    }
}


/* This function connects a QMSS Queue Pend interrupt to a core event*/
void setup_qpend_interrupt(Uint16 qnum, Uint16 CorePac_event)
{
    Int16 chan;
    CSL_IntcParam vectId1;

    /* These are event input index of CIC0 for Que_pend events from Que Manager */
    Uint8 events[10] = {134, 135, 136, 137, 138, 139, 140, 141, 142, 175};

    /* Step 1: Translate the queue number into the CIC input event. */
    /* qnum is expected to be 662..671 */
    chan = (qnum - FIRST_QPEND_QUEUE);
    if ((chan < 0) || (chan > 10))
    {
        printf("Invalid Queue Pend queue %d\n", qnum);
        return;
    }
    sys_event = events[chan];

    /* Step 2: Map the CIC input event to the CorePac input event
     * (which are 56 to 63). */
    CSL_CPINTC_disableAllHostInterrupt(cphnd);

    CSL_CPINTC_setNestingMode(cphnd, CPINTC_NO_NESTING);

    /* Map the input system event to a channel.  Note, the
     * mapping from channel to Host event is fixed.*/
    CSL_CPINTC_mapSystemIntrToChannel(cphnd, sys_event, 0);

    /* Enable the system interrupt */
    CSL_CPINTC_enableSysInterrupt(cphnd, sys_event);

    host_event = CorePac_event - FIRST_QPEND_CIC0_OUT_EVENT;

    /* Enable the channel (output). */
    CSL_CPINTC_enableHostInterrupt(cphnd, host_event);

    /* Enable all host interrupts. */
    CSL_CPINTC_enableAllHostInterrupt(cphnd);

    /* Step 3: Hook an ISR to the CorePac input event. */
    vectId1 = CSL_INTC_VECTID_12; //4 through 15 are available
    hIntcQmss[QPEND_IDX] = CSL_intcOpen(&intcQmss,
                                        CorePac_event, // selected event ID
                                       &vectId1,
                                        NULL);

    /* Hook the ISR */
    Record[QPEND_IDX].handler = (CSL_IntcEventHandler)&QPEND_USER_DEFINED_ISR;
    Record[QPEND_IDX].arg = (void *)CorePac_event;
    CSL_intcPlugEventHandler(hIntcQmss[QPEND_IDX],&(Record[QPEND_IDX]));

    /* Clear the Interrupt */
    CSL_intcHwControl(hIntcQmss[QPEND_IDX],CSL_INTC_CMD_EVTCLEAR,NULL);

    /* Enable the Event & the interrupt */
    CSL_intcHwControl(hIntcQmss[QPEND_IDX],CSL_INTC_CMD_EVTENABLE,NULL);

    return;
}
```

```
/* This function ISR for QPEND interrupts. Note the interrupt is
routed through chip-level CIC */
void QPEND_USER_DEFINED_ISR (Uint32 eventId)
{

    /* Disable the CIC0 host interrupt output */
    CSL_CPINTC_disableHostInterrupt(cphnd, host_event);

    /* Clear the CIC0 system interrupt */
    CSL_CPINTC_clearSysInterrupt(cphnd, sys_event);

    ......
    (service the interrupt at source)
    (clear the source interrupt flag is typically the first step in the service)

    /* Clear the CorePac interrupt */
    CSL_intcHwControl(hIntcQmss[QPEND_IDX],CSL_INTC_CMD_EVTCLEAR,NULL);

    /* Enable the CIC0 host interrupt output */
    CSL_CPINTC_enableHostInterrupt(cphnd, host_event);

    /* Optional: Close the handle if we are remapping with each call. */
    CSL_intcClose(&intcQmss);
}
```

End of Example 2-1

2.3.1 Interrupts Require End of Interrupt Handshaking

Some peripherals have an interrupt that requires handshaking. The interrupt can be triggered by multiple conditions. After the CorePac services the interrupt, the CorePac needs to write a value into the EOI_VECTOR field in the End of Interrupt register (there is one such register for each peripheral and it is inside the peripheral's register space) to acknowledge that it has cleared the interrupt condition before a new interrupt can be generated. See the peripheral user guide for details.

2.4 Inter-Processor Communication

Each of the cores can communicate with one another through inter-processor interrupts for core synchronization, allowing for direct notification from one core to another. After one has completed a particular task it can notify the other core(s) by asserting an event through software through the control register(s), which will result in an event flag and/or core interrupt to the other core(s).

2.4.1 Inter-Processor Interrupt Registers (IPCGR0~IPCGRx and IPCAR0~IPCARx)

IPCGRx are the IPC interrupt generation registers and IPCARx are the IPC interrupt acknowledgement registers to facilitate inter core interrupts.

KeyStone Architecture has x IPCGRx registers (x=number of cores) and x IPCARx registers (x=number of cores). This can be used by external hosts or cores to generate interrupts to other cores. A write of 1 to IPCG field of IPCGRx register will generate an interrupt pulse to core x (0 <= x <= number of cores) for KeyStone Architecture.

These registers also provide a *Source ID* facility by which up to 28 different sources of interrupts can be identified. Allocation of source bits to source processor and meaning is based entirely on software convention. The registers are shown in Figure 2-19 and Figure 2-20. Virtually anything can be a source for these registers as this is completely controlled by software. Any master that has access to BOOTCFG space can write to these registers.

Figure 2-19 IPC Generation Register (IPCGRx)

31	30	29	28	27 8	7	6	5	4	3 1	0
SRCS27	SRCS26	SRCS25	SRCS24	SRCS23–SRCS4	SRCS3	SRCS2	SRCS1	SRCS0	Reserved	IPCG
R/W-0	R/W-0	R/W-0	R/W-0	RW-0 (per bitfield)	R/W-0	R/W-0	R/W-0	R/W-0	R-000	R/W-0

Legend: R = Read only; W = Write only; -*n* = value after reset

Table 2-20 IPC Generation Register Field Descriptions

Bit	Field	Description
31-4	SRCSx	This field sets both SRCSx and the corresponding SRCCx bits. Reads return current value of internal register bit. When written (per bit position): 0 = No effect 1 = Sets both SRCSx and the corresponding SRCCx bits
3-1	Reserved	Reads return 0 and writes have no effect.
0	IPCG	This field creates an inter-DSP interrupt. Reads return 0. When written: 0 = No effect 1 = Create an Inter-DSP Interrupt pulse to the corresponding core (core 0 for IPCGR0, etc.)

Figure 2-20 IPC Acknowledgement Register (IPCARx)

31	30	29	28	27 8	7	6	5	4	3 0
SRCC27	SRCC26	SRCC25	SRCC24	SRCC23–SRCC4	SRCC3	SRCC2	SRCC1	SRCC0	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	RW-0 (per bitfield)	R/W-0	R/W-0	R/W-0	R/W-0	R-0

Legend: R = Read only; W = Write only; -*n* = value after reset

Table 2-21 IPC Acknowledgement Register Field Descriptions

Bit	Field	Description
31-4	SRCCx	This field clears both SRCCx and corresponding SRCSx bits. Reads return current value of internal register bit. When written (per bit position): 0 = No effect 1 = Clears both SRCCx and the corresponding SRCSx bits
3-0	Reserved	Reads return 0 and writes have no effect.

2.4.2 Inter-Processor Host Interrupt Registers (IPCGRH and IPCARH)

The CIC provides an interrupt output capability to internal hosts such as a CorePac or a RISC core inside KeyStone Architecture devices. Similarly, there is a need to provide an interrupt output capability to an external host (another processor such as a KeyStone Architecture device). To facilitate external host core interrupt, IPCGRH and IPCARH registers are provided. Operation and use of IPCGRH/IPCARH is the same as other IPCGR/IPCAR registers. The register fields are shown in Figure 2-21 and Figure 2-22. The interrupt output pulse created by IPCGRH is driven on a device pin, external host interrupt/event output (HOUT).

The external host interrupt output pulse is asserted for 4 core/6 clock cycles followed by a deassertion of 4 core/6 clock cycles.

Figure 2-21 IPC Host Generation Register (IPCGRH)

31	30	29	28	27 8	7	6	5	4	3 1	0
SRCS27	SRCS26	SRCS25	SRCS24	SRCS23–SRCS4	SRCS3	SRCS2	SRCS1	SRCS0	Reserved	IPCG
R/W-0	R/W-0	R/W-0	R/W-0	RW-0 (per bitfield)	R/W-0	R/W-0	R/W-0	R/W-0	R-000	R/W-0

Legend: R = Read only; W = Write only; -*n* = value after reset

Table 2-22 IPC Host Generation Register Field Descriptions

Bits	Field	Description
31-4	SRCSx	This field sets both SRCSx and the corresponding SRCCx bits. Reads return current value of internal register bit. When written (per bit position): 0 = No effect 1 = Sets both SRCSx and the corresponding SRCCx bits
3-1	Reserved	Reads return 0. Writes has no effect.
0	IPCG	This field creates an inter-DSP interrupt. Reads return 0. When written: 0 = No effect 1 = Create an Interrupt pulse on device pin (external host interrupt/event output in HOUT pin)

Figure 2-22 IPC Host Acknowledgement Register (IPCARH)

31	30	29	28	27 8	7	6	5	4	3 0
SRCC27	SRCC26	SRCC25	SRCC24	SRCC23–SRCC4	SRCC3	SRCC2	SRCC1	SRCC0	Reserved
R/W-0	R/W-0	R/W-0	R/W-0	RW-0 (per bitfield)	R/W-0	R/W-0	R/W-0	R/W-0	R-0

Legend: R = Read only; W = Write only; -*n* = value after reset

Table 2-23 IPC Host Acknowledgement Register Field Descriptions

Bits	Field	Description
31-4	SRCCx	This field clears both SRCCx and corresponding SRCSx bits. Reads return current value of internal register bit. When written (per bit position): 0 = No effect 1 = Clears both SRCCx and the corresponding SRCSx bits
3-0	Reserved	Reads return 0 and writes have no effect.

2.4.3 NMI Event Generation to Core

NMIGRx registers are used for generating an NMI event to the corresponding core. KeyStone Architecture has x NMIGRx registers (NMIGR0 through NMIGRx, x = number of cores). The NMIGR0 register generates an NMI event to core0, the NMIGR1 register generates an NMI event to core1, and so on. Writing a 1 to the NMIG field generates an NMI pulse. Writing a 0 has no effect and reads return 0 and have no other effect. The register is shown in Figure 2-23.

Figure 2-23 NMI Generation Register (NMIGRx)

31 1	0
Reserved	NMIG
R-0	R-0

Legend: R = Read only; -*n* = value after reset

Table 2-24 NMI Generation Register Field Descriptions

Bits	Field	Description
31-1	Reserved	Reads return 0 and writes have no effect.
0	NMIG	This field creates NMI pulse to the corresponding core. Reads return 0. When written: 0 = No effect 1 = Create NMI pulse to the corresponding core (core 0 for NMIGR0, etc.)

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**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Mobile Processors	www.ti.com/omap		
Wireless Connectivity	www.ti.com/wirelessconnectivity		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265