TMS320C6474 DSP
Chip Interrupt Controller (CIC)

User's Guide

TEXAS INSTRUMENTS

Literature Number: SPRUFK6
October 2008

Contents

List of Figures

List of Tables

Read This First

About This Manual

This document describes system event routing using the chip interrupt controller (CIC) for the TMS320C6474 devices. In the TMS320C6474 devices, the number of system events is greater than each core's (C64x+ Megamodule) interrupt handler logic supported occurrences. To facilitate the robustness of event routing, CIC logic is incorporated to multiplex the system events and feed each core and third-party channel controller (TPCC), accordingly. There is dedicated CIC hardware logic for each C64x+ core and TPCC. Event selection is done by software programming of their resources.

The TMS320C6474 also has the capability to communicate with different cores by programming the inter-processor (core) communication (IPC) registers to facilitate handshaking between each core.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the C6000™ devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

SPRU189 — *TMS320C6000 DSP CPU and Instruction Set Reference Guide.* Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C6000 digital signal processors (DSPs).

SPRU198 — *TMS320C6000 Programmer's Guide.* Describes ways to optimize C and assembly code for the TMS320C6000™ DSPs and includes application program examples.

SPRU301 — *TMS320C6000 Code Composer Studio Tutorial.* Introduces the Code Composer Studio™ integrated development environment and software tools.

SPRU321 — *Code Composer Studio Application Programming Interface Reference Guide.* Describes the Code Composer Studio™ application programming interface (API), which allows you to program custom plug-ins for Code Composer.

SPRU871 — *TMS320C64x+ Megamodule Reference Guide.* Describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

SPRS552 — *TMS320C6474 Multicore Digital Signal Processor* data manual.

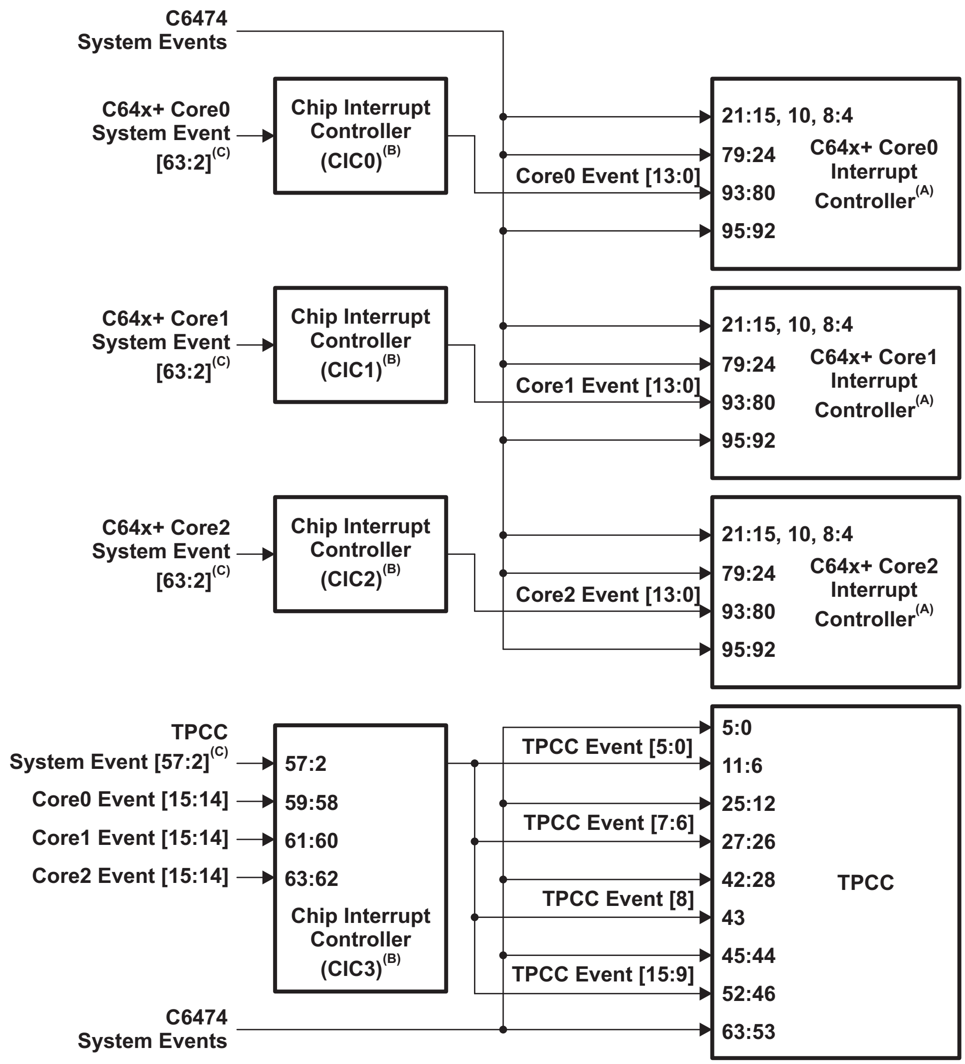
Trademarks

TMS320C6474 Chip Interrupt Controller (CIC)

1 Overview

This section describes the details of how the system events are propagated to each core and TPCC. In the C6474 devices, there is dedicated CIC logic to route system events corresponding to each core and TPCC. CIC0 is dedicated to the C64x+ Core0 system event routing, CIC1 is dedicated to C64x+ Core1 system event routing, CIC2 is dedicated to C64x+ Core2 system event routing, and CIC3 is dedicated to TPCC system event routing.

Apart from the CIC, each core has its own interrupt controller through which a particular event/interrupt is propagated to the CPU and AEG logic. The C64x+ core interrupt controller includes an event combiner, an interrupt selector, an exception combiner, and an advanced event generator (AEG) selector. The event combiner allows a large number of system events to be greatly reduced. The interrupt selector allows any of the system events to be routed to up to 12 maskable interrupts. The exception combiner lets any of the system events be grouped together for a single exception input to the core. The AEG event selector allows any system event to be an AEG trigger. The TPCC supports up to 64 events and there is event detection logic that recognizes the TPCC system events. Figure 1 shows the basic architecture of the C6474 event routing system.



A Input events 0-3, 9, 11-14 and 96-127 are the C64x+ core's internal events and are not available at device level.

B The CIC input events [1:0] are internal to the C64x+ core and are not available at device level.

C The C64x+ Corex system events [63:2] and TPCC system events [57:2] are a subset of system events.

Figure 1. Overview of Event Routing in the TMS320C6474 Device

1.1 Terms and Abbreviations

Table 1. Terms and Abbreviations

Term	Description
AEG	Advanced Event Generation
AIF	Antenna Interface
DMA	Direct Memory Access
CIC	Chip Interrupt Controller
CSL	Chip Support Library
EDMA	Enhanced DMA Controller; also referred to as TPDMA
ESL	Event Selection Logic
FSYNC	Frame Synchronization

Table 1. Terms and Abbreviations (continued)

Term	Description
GPIO	General-Purpose Input/Output
GPSC	Global Power/Sleep Controller; manages the LPSCs
I2C	Inter-Integrated Circuit Control Bus
IPC	Inter-Processor (Core) Communication
LPSC	Local Power/Sleep Controller; one per module
McBSP	Multichannel Buffered Serial Port
MDIO	Management Data Input/Output
TCP	Turbo Coprocessor
TPCC	TPDMA Channel Controller
TPDMA	Third-Party DMA Engine; also referred to as EDMA
TPTC	TPDMA Transfer Controller
VCP	Viterbi Coprocessor
CSL	Chip Support Library

2 Chip Interrupt Controller Overview

The Chip Interrupt Controller is comprised of the event combiner, event selection logic (ESL), and the event flag register. ESL supports any of the 64 input system events and 16 output events. Figure 2 describes the basic building block of the CIC.

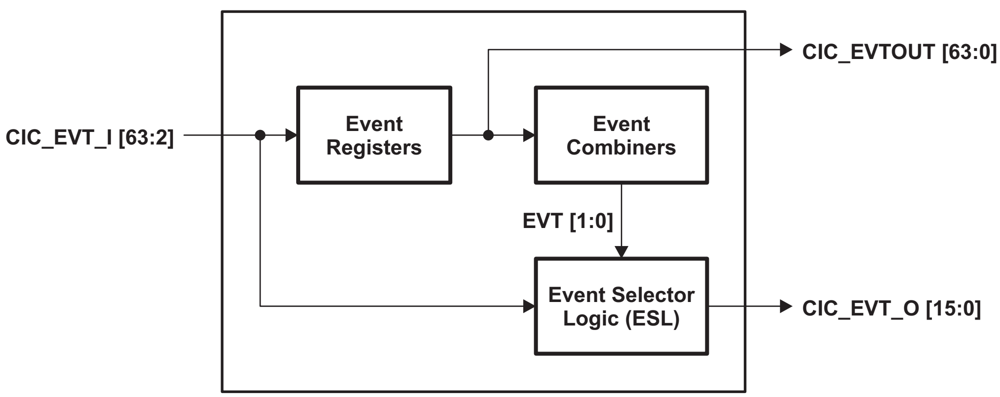


Figure 2. Chip Interrupt Controller Block Diagram

The different system events are connected to the CIC input event pins (CIC_EVT_I[x], where x = 2 to 63); the other two events are generated from CIC logic (EVT [1:0]). EVT0 is generated by the combination of CIC_EVT_I [31:2] and EVT1 is generated by the combination of CIC_EVT_I [63:32]. The entire 64 events are available along with 16 selection-logic events (CIC_EVT_O [15:0]). For more detailed information regarding the different system events connected to the CIC input events, see Section 8 and Section 9.

3 Event Selection Logic

There are four instances of the CIC module in the C6474 device, each of these modules has an ESL block which can accommodate 64 input events and generate 16 output events. Figure 3 shows the event selection logic block diagram.

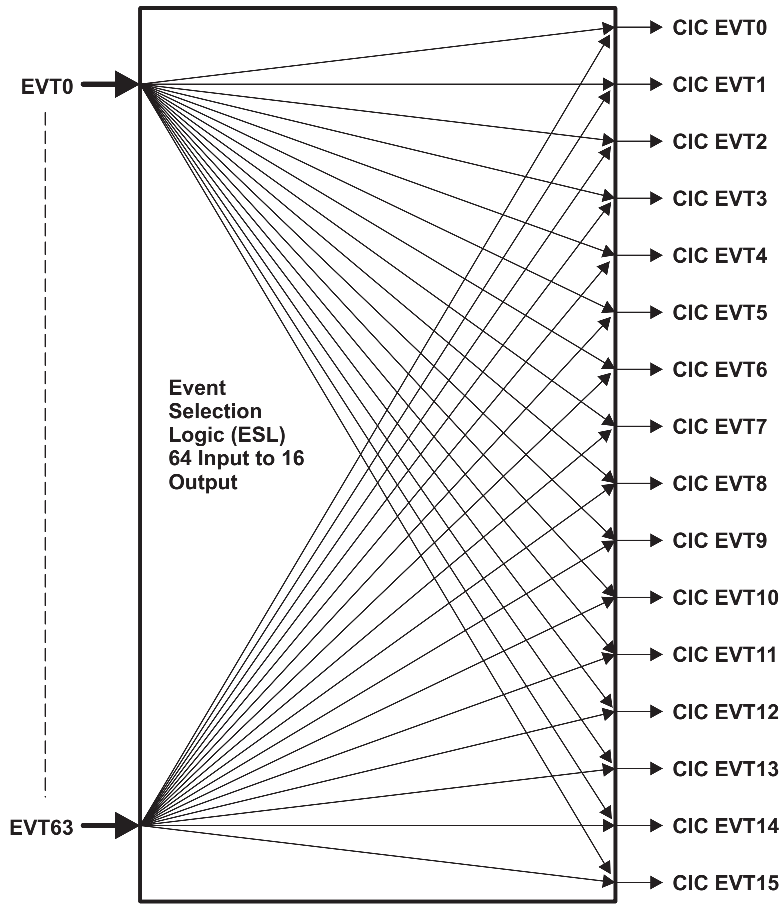


Figure 3. Event Selection Logic Block Diagram

Event selection logic can take 64 input events and route each of these to any of the distinct 16 output pins by programming the event mux registers (EVTMUX) output event selection fields (EVTMUXx[CIC_EVTy], where x = 0,1,2,3 and y = 0,1,….15). By doing this, you have a choice of assigning a hardware system event/interrupt to any of the 16 output pins that go to the core interrupt handler and the TPCC. For more detailed information regarding the different CIC_EVT and their mapping in respect to the core interrupt handler and the TPCC, see Section 8 and Section 9, respectively.

4 CIC Registers

There are four instances of CIC logic; each one supports the following set of registers. Table 2 lists the base addresses for the CIC0-CIC3 registers and Table 3 is a summary of the CIC registers.

Table 2. Chip Interrupt Controller Registers

Base Address	Acronym	Description
0x02880000	CIC0	Chip Interrupt Controller 0 Registers
0x02880100	CIC1	Chip Interrupt Controller 1 Registers
0x02880200	CIC2	Chip Interrupt Controller 2 Registers
0x02880300	CIC3	Chip Interrupt Controller 3 Registers

Table 3. CIC Register Summary

Offset	Acronym	Description	See
0x00 - 0x03	EVTFLAG0	Event Flag Register 0	Section 4.1
0x04 - 0x07	EVTFLAG1	Event Flag Register 1	Section 4.1
0x10 - 0x13	EVTSET0	Event Set Register 0	Section 4.2
0x14 - 0x17	EVTSET1	Event Set Register 1	Section 4.2
0x20 - 0x23	EVTCLR0	Event Clear Register 0	Section 4.3
0x24 - 0x27	EVTCLR1	Event Clear Register 1	Section 4.3
0x30 - 0x33	EVTMASK0	Event Mask Register 0	Section 4.4
0x34 - 0x37	EVTMASK1	Event Mask Register 1	Section 4.4
0x40 - 0x43	MEVTFLAG0	Masked Event Flag Register 0	Section 4.5
0x44 - 0x47	MEVTFLAG1	Masked Event Flag Register 1	Section 4.5
0x50 - 0x53	EVTMUX0	Event Mux Register 0	Section 4.6
0x54 - 0x57	EVTMUX1	Event Mux Register 1	Section 4.6
0x58 - 0x5B	EVTMUX2	Event Mux Register 2	Section 4.6
0x5C - 0x5F	EVTMUX3	Event Mux Register 3	Section 4.6

4.1 Event Flag Register

The event flag register (EVTFLAG*n*) captures all 64 input events that are received by the CIC. This is a read-only register. Each field is set to 1 once a particular event is received until it is cleared by writing a 1 to the respective event clear register (EVTCLR*n*) file.

Event flag register 0 (EVTFLAG0) and event flag register 1 (EVTFLAG1) are shown in Figure 4 and Figure 5 and described in Table 4.

Figure 4. Event Flag Register 0 (EVTFLAG0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EF31	EF30	EF29	EF28	EF27	EF26	EF25	EF24	EF23	EF22	EF21	EF20	EF19	EF18	EF17	EF16

R-0000000000000000000000000000000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EF15	EF14	EF13	EF12	EF11	EF10	EF9	EF8	EF7	EF6	EF5	EF4	EF3	EF2	EF1	EF0

R-0000000000000000000000000000000

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

Figure 5. Event Flag Register 1 (EVTFLAG1)

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
EF63	EF62	EF61	EF60	EF59	EF58	EF57	EF56	EF55	EF54	EF53	EF52	EF51	EF50	EF49	EF48

R-0000000000000000000000000000000

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
EF47	EF46	EF45	EF44	EF43	EF42	EF41	EF40	EF39	EF38	EF37	EF36	EF35	EF34	EF33	EF32

R-0000000000000000000000000000000

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

Table 4. Event Flag Registers (EVTFLAG*n*) Field Descriptions

Bit	Field	Value	Description
63-0	EF*n*		Event Flag *n*
		0	There is no input event to be captured
		1	Input event is being captured

4.2 Event Set Register

The event set register (EVTSET*n*) is useful for debugging purposes. By writing a 1 to this bit file, you can manually get a dummy event and the particular EVTFLAG*n* register fields will be set.

Event set register 0 (EVTSET0) and event set register 1 (EVTSET1) are shown in Figure 6 and Figure 7 and described in Table 5.

Figure 6. Event Set Register 0 (EVTSET0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ES31	ES30	ES29	ES28	ES27	ES26	ES25	ES24	ES23	ES22	ES21	ES20	ES19	ES18	ES17	ES16

W-00000000000000000000000000000000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ES15	ES14	ES13	ES12	ES11	ES10	ES9	ES8	ES7	ES6	ES5	ES4	ES3	ES2	ES1	ES0

W-00000000000000000000000000000000

LEGEND: W = Write only; -*n* = value after reset

Figure 7. Event Set Register 1 (EVTSET1)

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
ES63	ES62	ES61	ES60	ES59	ES58	ES57	ES56	ES55	ES54	ES53	ES52	ES51	ES50	ES49	ES48

W-00000000000000000000000000000000

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
ES47	ES46	ES45	ES44	ES43	ES42	ES41	ES40	ES39	ES38	ES37	ES36	ES35	ES34	ES33	ES32

W-00000000000000000000000000000000

LEGEND: W = Write only; -*n* = value after reset

Table 5. Event Set Registers (EVTSET*n*) Field Descriptions

Bit	Field	Value	Description
63-0	ES*n*		Event Set *n*
		0	Do nothing
		1	Manually set event in the EVTFLAG*n* [ES*n*] field

4.3 Event Clear Register

The event clear register (EVTCLR*n*) clears a particular field of the EVTFLAG*n* register. After servicing that particular event/interrupt, the CPU should write a 1 to the respective (EVTCLR*n*) bit field to clear the event so that the next event can be serviced.

Event clear register 0 (EVTCLR0) and event clear register 1 (EVTCLR1) are shown in Figure 8 and Figure 9 and described in Table 6.

Figure 8. Event Clear Register 0 (EVTCLR0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EC31	EC30	EC29	EC28	EC27	EC26	EC25	EC24	EC23	EC22	EC21	EC20	EC19	EC18	EC17	EC16

W-0000000000000000000000000000000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EC15	EC14	EC13	EC12	EC11	EC10	EC9	EC8	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0

W-0000000000000000000000000000000

LEGEND: W = Write only; -*n* = value after reset

Figure 9. Event Clear Register 1 (EVTCLR1)

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
EC63	EC62	EC61	EC60	EC59	EC58	EC57	EC56	EC55	EC54	EC53	EC52	EC51	EC50	EC49	EC48

W-0000000000000000000000000000000

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
EC47	EC46	EC45	EC44	EC43	EC42	EC41	EC40	EC39	EC38	EC37	EC36	EC35	EC34	EC33	EC32

W-0000000000000000000000000000000

LEGEND: W = Write only; -*n* = value after reset

Table 6. Event Clear Registers (EVTCLR*n*) Field Descriptions

Bit	Field	Value	Description
63-0	EC*n*		Event Clear *n*
		0	No effect
		1	Clears the event

4.4 Event Mask Register

In the CIC, there is an event combiner sub-module that can combine 32 input events into a single event and be used as a single CPU interrupt. EVT0 is generated from 30 input events, CIC_EVT_I [31:2]; EVT1 is generated from 32 input events, CIC_EVT_I [63:32].

The event mask register (EVTMASK) is used to select which input events are used to generate a combined event output, (EVT*n*). By default, all the events are unmasked except EVTMASK0 [1:0], which is unused and masked. If you want to mask a particular input event, you need to write a 1 to the field of that particular EVTMASK*n* register.

Event mask register 0 (EVTMASK0) and event mask register 1 (EVTMASK1) are shown in Figure 10 and Figure 11 and described in Table 7.

Figure 10. Event Mask Register 0 (EVTMASK0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EM31	EM30	EM29	EM28	EM27	EM26	EM25	EM24	EM23	EM22	EM21	EM20	EM19	EM18	EM17	EM16

R/W-0000000000000000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM15	EM14	EM13	EM12	EM11	EM10	EM9	EM8	EM7	EM6	EM5	EM4	EM3	EM2	EM1	EM0

R/W-0000000000000011

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

Figure 11. Event Mask Register 1 (EVTMASK1)

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
EM63	EM62	EM61	EM60	EM59	EM58	EM57	EM56	EM55	EM54	EM53	EM52	EM51	EM50	EM49	EM48

R/W-0000000000000000

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
EM47	EM46	EM45	EM44	EM43	EM42	EM41	EM40	EM39	EM38	EM37	EM36	EM35	EM34	EM33	EM32

R/W-0000000000000000

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

Table 7. Event Mask Registers (EVTMASK*n*) Field Descriptions

Bit	Field	Value	Description
63-0	EM*n*		Event Mask *n*
		0	Input event is unmasked (enabled)
		1	Input event is masked (disabled)

4.5 Masked Event Flag Register

In addition to the EVTFLAG register, the CIC incorporates another level of viewing masked events in a register, called the masked event flag register (MEVTFLAG*n*). By reading this status register, the CPU can find out which particular unmasked events need service. Masked events are generated by logical ANDing of particular fields of the EVTFLAG register and the EVTMASK register.

Masked event flag register 0 (MEVTFLAG0) and masked event flag register 1 (MEVTFLAG1) are shown in Figure 12 and Figure 13 and described in Table 8.

Figure 12. Masked Event Flag Register 0 (MEVTFLAG0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEF31	MEF30	MEF29	MEF28	MEF27	MEF26	MEF25	MEF24	MEF23	MEF22	MEF21	MEF20	MEF19	MEF18	MEF17	MEF16

R-0000000000000000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEF15	MEF14	MEF13	MEF12	MEF11	MEF10	MEF9	MEF8	MEF7	MEF6	MEF5	MEF4	MEF3	MEF2	MEF1	MEF0

R-0000000000000000

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

Figure 13. Masked Event Flag 1 (MEVTFLAG1)

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
MEF63	MEF62	MEF61	MEF60	MEF59	MEF58	MEF57	MEF56	MEF55	MEF54	MEF53	MEF52	MEF51	MEF50	MEF49	MEF48

R-0000000000000000

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
MEF47	MEF46	MEF45	MEF44	MEF43	MEF42	MEF41	MEF40	MEF39	MEF38	MEF37	MEF36	MEF35	MEF34	MEF33	MEF32

R-0000000000000000

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

Table 8. Masked Event Flag Registers (MEVTFLAG*n*) Field Descriptions

Bit	Field	Value	Description
63-0	MEF*n*		Masked Event Flag *n*
		0	CPU does not need to do anything for this event
		1	This enabled event has occurred

4.6 Event Mux Registers

In the CIC, there are four event mux registers that allow the event combiner to select any of the 64 input events to be configured to connect to any 16 distinct output events.

Event mux register 0 (EVTMUX0), event mux register 1 (EVTMUX1), event mux register 2 (EVTMUX2), and event mux register 3 (EVTMUX3) are shown in Figure 14 through Figure 17 and described in Table 9.

Figure 14. Event Mux Register 0 (EVTMUX0)

31–30	29–24	23–22	21–16
Reserved	CIC_EVT3	Reserved	CIC_EVT2
R-00	R/W-000011	R-00	R/W-000010

15–14	13–8	7–6	5–0
Reserved	CIC_EVT1	Reserved	CIC_EVT0
R-00	R/W-000001	R-00	R/W-000000

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

Figure 15. Event Mux Register 1 (EVTMUX1)

31–30	29–24	23–22	21–16
Reserved	CIC_EVT7	Reserved	CIC_EVT6
R-00	R/W-000111	R-00	R/W-000110

15–14	13–8	7–6	5–0
Reserved	CIC_EVT5	Reserved	CIC_EVT4
R-00	R/W-000101	R-00	R/W-000100

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

Figure 16. Event Mux Register 2 (EVTMUX2)

31–30	29–24	23–22	21–16
Reserved	CIC_EVT11	Reserved	CIC_EVT10
R-00	R/W-001011	R-00	R/W-001010

15–14	13–8	7–6	5–0
Reserved	CIC_EVT9	Reserved	CIC_EVT8
R-00	R/W-001001	R-00	R/W-001000

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

Figure 17. Event Mux Register 3 (EVTMUX3)

31–30	29–24	23–22	21–16
Reserved	CIC_EVT15	Reserved	CIC_EVT14
R-00	R/W-001111	R-00	R/W-001110

15–14	13–8	7–6	5–0
Reserved	CIC_EVT13	Reserved	CIC_EVT12
R-00	R/W-001101	R-00	R/W-001100

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

Table 9. Event Mux Registers (EVTMUX*n*) Field Descriptions

Bit	Field	Value	Description
29-24 21-16 13-8 5-0	CIC_EVT*n*		CIC_EVT*n* (where *n* = 0,1,2 …15) For example, for CIC_EVT0:
		000000	CIC_EVT_I [0] is connected to CIC_EVT_O [0]. In TMS320C6474 this is reserved.
		000001	CIC_EVT_I [1] is connected to CIC_EVT_O [0]. In TMS320C6474 this is reserved.
		000010	CIC_EVT_I [2] is connected to CIC_EVT_O[0]
		000011	CIC_EVT_I [3] is connected to CIC_EVT_O[0]
		111110	CIC_EVT_I [62] is connected to CIC_EVT_O[0]
		111111	CIC_EVT_I [63] is connected to CIC_EVT_O[0]

5 CIC Usage

One peripheral interrupt that goes to the C64x+ core via the CIC is used to illustrate CIC usage. The C6474 device has a semaphore module that gives a semaphore error interrupt when there are atomic access violations by different cores using the semaphore. The semaphore error interrupt (event number 58) goes to CIC0, CIC1, and CIC2 for respective core-specific CICs, then routes this input event to distinct CIC_OUT events according to your programmed CIC logic.

In this example for CIC0 (of Core0), CIC_EVENT_IN (input event ID = 58) is routed to CIC_EVT_O (output event ID = 0). Once the CIC propagates the event to CIC_EVT_O (ID = 0), it is the C64x+ Core0 interrupt handler's responsibility to service the interrupt. Using CSL, the following example code routes system event 58 via CIC0 to Core0. The semaphore error event is generated by software programming using the Core0 semaphore error register.

```
// Structure for the CIC CSL-object allocated by you
CSL_CicObj        hCicObj;
// Structure for the CIC specific parameter
CSL_CicParam      CicParam;
// Structure for returning status of the function call
CSL_Status        hCicStatus;
// This is returned by the CSL_cicOpen (...) API. The handle is used
// to identify the event of interest in all CIC calls
CSL_CicHandle     hCicHandle;
// Status returned for test pass signature. 0-> Pass 1-> Fail
int status;
// Status returned for the CIC pending event status
Uint32 intrStat;
// This is returned by the CSL_semOpen (...) API.
CSL_SemHandle   hSemHandle;
// Structure for the object that holds reference to the
// instance of SEM requested after the call
CSL_SemObj        hSemObj;
// Structure for returning status of the function call
CSL_Status      SemStat;
// Structure for module specific parameters, (Number of SEM resource)
CSL_SemParam    Param;
// Instance of SEM to which a handle is requested
CSL_InstNum     SemInst =0;
// Object for Semaphore error settings
CSL_SemErrSet_Arg SemErr;



// Flag used for handshaking with ISR routine
Bool interruptFlag = FALSE;

void main (void)
{
   Uint32 result;
   VAL_TRACE (0x01010101);
   // Chip Specific Initialization
   VAL_chipInit (NULL);

   // Environment Initialization
   VAL_envInit (NULL);

   // Function call to perform the actual test
   result = CIC_DoTest ();

   // This function does the result reporting and termination
   VAL_testExit (result);
}

// This function will do CIC route test for Event ID = 58. i.e.: Semaphore Err Int.
CIC_DoTest () {

    // Step 1: CICInit
   CSL_cicInit(NULL);

    CicParam.ectlEvtId = CSL_CIC_ECTL_EVT0;       // CIC Out = 0 ;
    CicParam.eventId   = CSL_CIC_EVENTID_SEMERR ; // Semaphore Err Event = 58
```

```
    // Step 2: CICOpen for CIC0 (CSL_CIC_0)
    hCicHandle = CSL_cicOpen(&hCicObj,CSL_CIC_0,&CicParam, &hCicStatus);

    if (hCicHandle != NULL ) {
     CSL_cicHwControl(hCicHandle, CSL_CIC_CMD_EVTENABLE, NULL);
    }

    // Generate a Sem4 Error Interrupt (Event ID 58) which goes to CIC
    Gen_Sem_Err();

    // Check the CIC Masked Event Flag Register.
    CSL_cicGetHwStatus(hCicHandle,CSL_CIC_QUERY_PENDSTATUS,(void*)&intrStat);

    // Now wait and see whether ISR is serviced using CIC event route
    while (interruptFlag == FALSE)  {
     status = 1;
    };
    status  = 0; // return pass if interrupt check passed.
    // Close CIC handle
    CSL_cicClose(hCicHandle);
    return status;
}

// This part of the code will generate Sem Err (Evt ID =58)
Gen_Sem_Err () {

    // Step 1: SemInit
    CSL_semInit(NULL);

    Param.flags = 4; // SEM 4 is used.

    // Step 2: semOpen
    hSemHandle = CSL_semOpen(&hSemObj, SemInst,&Param, &SemStat);

    Setup_Interrupt();

    // Set Semaphore Error
    SemErr.errCode    = 1; // setting semaphore error forcefully.
    SemErr.mstId      = 0; // Core 0
    SemErr.semNum     = 4; // Sem 4 is used

    if(hSemHandle!=NULL){
    // Setting Semaphore Error which intern generate SemErr (ID 58)
       CSL_semHwControl(hSemHandle,CSL_SEM_CMD_ERR_SET,&SemErr);
    }
    // Close Semaphore handle
    CSL_semClose(hSemHandle);

}

void Setup_Interrupt() {

    CSL_IntcParam vectId1;
    // Setup the global Interrupt
    context.numEvtEntries = 3;
    context.eventhandlerRecord = Record;
    CSL_intcInit(&context);

    // Enable NMIs
    CSL_intcGlobalNmiEnable();

    // Enable Global Interrupts
    CSL_intcGlobalEnable(NULL);

    // For Semaphore error interrupt
    vectId1 = CSL_INTC_VECTID_5;          // CIC_OUT_0 = 80 for Core0
    hIntcSemErr = CSL_intcOpen (&intcSemErr, 80, &vectId1 , NULL);
    EventRecord.handler = (CSL_IntcEventHandler)SemaphoreErrorHandler;
    EventRecord.arg = (void*)hSemHandle;
    CSL_intcPlugEventHandler(hIntcSemErr,&EventRecord);
    /* Clear the Interrupt */
    CSL_intcHwControl(hIntcSemErr,CSL_INTC_CMD_EVTCLEAR,NULL);

    /* Enable the Interrupt */
```

```
    CSL_intcHwControl(hIntcSemErr,CSL_INTC_CMD_EVTENABLE,NULL);

}

void SemaphoreErrorHandler(CSL_SemHandle   hSemHandle)
{
    interruptFlag = TRUE;
    VAL_TRACE(0xC001BABE);
}
```

6 Inter-Processor (Core) Interrupts

In multi-CPU devices, it is necessary to communicate between each core. In the C6474 device, two sets of registers are available for programming that facilitate inter-core communication. IPC generation registers (IPCGR0, IPCGR1, IPCGR2) and IPC acknowledgment registers (IPCAR0, IPCAR1, IPCAR2) are dedicated for inter-core interrupts. These registers can be utilized by external hosts or the C64x+ core to communicate with other C64x+ cores. Writing a 1 to the IPCG field of the IPCGR*n* register generates an interrupt pulse to the Core*n* (*n* = 0, 1, 2). These registers also provide *source ID* flags to indicate which master/host has caused the interrupt pulse. Up to 28 different sources of interrupt can be generated and it is entirely software programmable for allocating *source ID* for any particular interrupt.

7 IPC Registers

Table 3 shows the summary for the inter-core communication registers.

Table 10. IPC Register Summary

Offset	Acronym	Description	See
0x02880900 - 0x02880903	IPCGR0	IPC Generation Register 0 (for C64x+ Core0)	Section 7.1
0x02880904 - 0x02880907	IPCGR1	IPC Generation Register 1 (for C64x+ Core1)	Section 7.1
0x02880908 - 0x0288090B	IPCGR2	IPC Generation Register 2 (for C64x+ Core2)	Section 7.1
0x02880940 - 0x02880943	IPCAR0	IPC Acknowledgment Register 0 (for C64x+ Core0)	Section 7.2
0x02880944 - 0x02880947	IPCAR1	IPC Acknowledgment Register 1 (for C64x+ Core1)	Section 7.2
0x02880948 - 0x0288094B	IPCAR2	IPC Acknowledgment Register 2 (for C64x+ Core2)	Section 7.2

7.1 IPC Generation Registers

The C6474 device has three IPC generation registers; each one is dedicated to each core. The IPCGR0 is dedicated to the C64x+ Core0, the IPCGR1 is dedicated to the C64x+ Core1, and the IPCGR2 is dedicated to the C64x+ Core2. Writing a 1 to the IPCG field of these registers generates an interrupt pulse to the respective core.

The IPC generation register (IPCGR0 - IPCGR2) is shown in Figure 18 and described in Table 11.

Figure 18. IPC Generation Registers (IPCGR0 - IPCGR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRCS27	SRCS26	SRCS25	SRCS24	SRCS23	SRCS22	SRCS21	SRCS20	SRCS19	SRCS18	SRCS17	SRCS16	SRCS15	SRCS14	SRCS13	SRCS12
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14	13	12	11	10	9	8	7	6	5	4	3		1	0
SRCS11	SRCS10	SRCS9	SRCS8	SRCS7	SRCS6	SRCS5	SRCS4	SRCS3	SRCS2	SRCS1	SRCS0	Reserved			IPCG
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-000			R/W-0

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

Table 11. IPC Generation Registers Field Descriptions

Bit	Field	Value	Description
31-4	SRCS*n*		Write
		0	No Effect
		1	Set register bit. The source or host/master has generated an inter-core interrupt pulse for the respective core. You can assign any of the available 28 different interrupt generator source IDs to any of these fields (31:4).
			Read
			Returns the current value.
3-1	Reserved	0	A value written to this field has no effect.
0	IPCG		Write
		0	No Effect
		1	Generate an inter-core interrupt pulse for the respective core. IPCGR0 is for Core0.
			Read
			Returns 0; no effect.

7.2 IPC Acknowledgment Registers

In the C6474 device there are three IPC acknowledgment registers; each one is dedicated to each core. IPCAR0 is dedicated to the C64x+ Core0, IPCAR1 is dedicated to the C64x+ Core1, and IPCAR2 is dedicated to the C64x+ Core2.

The IPC acknowledgment register (IPCAR0 - IPCAR2) is shown in Figure 19 and described in Table 12.

Figure 19. IPC Acknowledgment Registers (IPCAR0 - IPCAR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SRCC27	SRCC26	SRCC25	SRCC24	SRCC23	SRCC22	SRCC21	SRCC20	SRCC19	SRCC18	SRCC17	SRCC16	SRCC15	SRCC14	SRCC13	SRCC12
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14	13	12	11	10	9	8	7	6	5	4	3			0
SRCC11	SRCC10	SRCC9	SRCC8	SRCC7	SRCC6	SRCC5	SRCC4	SRCC3	SRCC2	SRCC1	SRCC0	Reserved			
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0000			

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

Table 12. IPC Acknowledgment Registers Field Descriptions

Bit	Field	Value	Description
31-4	SRCC*n*		Write
		0	No Effect
		1	Clears the corresponding bit field in the associated IPCAR*n* register.
			Read
			Returns the current value of the internal register.
3-0	Reserved	0	A value written to this field has no effect.

8 CIC Event Lists for the C64x+ Megamodule

Table 13 lists the 64 event inputs to the CIC [2:0]. There are some C64x+ core-specific events that go to a specified core from that core's CIC. Note that *n* implies the event number that matches the specific core number to which it is routed.

Table 13. Input Events to CIC[2:0]

Number	Name	Description
0	EVT0	Output of Event Combiner 0 for Events [31:2]
1	EVT1	Output of Event Combiner 1 for Events [63:32]
2	Unused	Reserved
3	Unused	Reserved
4	I2CINT	I2C Error Interrupt
5	FSERR	FSYNC Error Interrupt
6	RIOINT7	RapidIO Error Interrupt
7	PSC_ERRINT	GPSC Error Interrupt
8	VCPINT	VCP Error Interrupt
9	TCPINT	TCP Error Interrupt
10	RINT0	McBSP0 Receive Interrupt
11	XINT0	McBSP0 Transmit Interrupt
12	RINT1	McBSP1 Receive Interrupt
13	XINT1	McBSP1Transmit Interrupt
14	REVT0	McBSP0 Receive EDMA Event
15	XEVT0	McBSP0 Transmit EDMA Event
16	REVT1	McBSP1 Receive EDMA Event
17	XEVT1	McBSP1 Transmit EDMA Event
18	IREVT	I2C Receive EDMA Event
19	IXEVT	I2C Transmit EDMA Event
20	FSEVT18	FSYNC Event
21	FSEVT19	FSYNC Event
22	FSEVT20	FSYNC Event
23	FSEVT21	FSYNC Event
24	FSEVT22	FSYNC Event
25	FSEVT23	FSYNC Event
26	FSEVT24	FSYNC Event
27	FSEVT25	FSYNC Event
28	FSEVT26	FSYNC Event
29	FSEVT27	FSYNC Event
30	FSEVT28	FSYNC Event
31	FSEVT29	FSYNC Event
32	VCPREVT	VCP Receive Event
33	VCPXEVT	VCP Transmit Event
34	TCPREVT	TCP Receive Event
35	TCPXEVT	TCP Transmit Event
36	TPCC_ERRINT	TPCC Error Interrupt
37	TPCC_MPINT	TPCC Memory Protection Interrupt
38	TPTC_ERRINT0	TPTC0 Error Interrupt
39	TPTC_ERRINT1	TPTC1 Error Interrupt
40	TPTC_ERRINT2	TPTC2 Error Interrupt
41	TPTC_ERRINT3	TPTC3 Error Interrupt

Table 13. Input Events to CIC[2:0] (continued)

Number	Name	Description
42	TPTC_ERRINT4	TPTC4 Error Interrupt
43	TPTC_ERRINT5	TPTC5 Error Interrupt
44	TPCC_AETEVT	TPCC AET Event
45	AIF_EVT2	AIF CPU Interrupt 2
46	AIF_EVT3	AIF CPU Interrupt 3
47	AIF_PSEVT0	AIF Packet Switched Transfer Event
48	AIF_PSEVT1	AIF Packet Switched Transfer Event
49	AIF_PSEVT2	AIF Packet Switched Transfer Event
50	AIF_PSEVT3	AIF Packet Switched Transfer Event
51	AIF_PSEVT4	AIF Packet Switched Transfer Event
52	AIF_PSEVT5	AIF Packet Switched Transfer Event
53	AIF_PSEVT6	AIF Packet Switched Transfer Event
54	AIF_TEVT0	AIF Logic Analyzer Trace Event
55	AIF_TEVT1	AIF Logic Analyzer Trace Event
56:57	Unused	Reserved
58	SEMERR*n* [(1)]	Semaphore Error Event for C64x+ CORE*n*
63:59	Unused	Reserved

(1) The C64x+ Core0 receives SEMERR0 interrupts, the C64x+ Core1 receives SEMERR1 interrupts, and the C64x+ Core2 receives SEMERR2 interrupts.

9 CIC Event Lists for TPCC

The CIC3 is responsible for routing the system events to TPCC. Several events that routed through the event controller may be required to be used as trigger events for a DMA transaction. Table 14 lists the 64 event inputs to the CIC3.

Table 14. Input Events to CIC3

Number	Name	Description
0	EVT0	Output of Event Combiner 0 for Events [31:2]
1	EVT1	Output of Event Combiner 1 for Events [63:32]
2	FSEVT0	Frame Synchronization Event 0
3	FSEVT1	Frame Synchronization Event 1
4	FSEVT2	Frame Synchronization Event 2
5	FSEVT3	Frame Synchronization Event 3
6	FSEVT14	Frame Synchronization Event 14
7	FSEVT15	Frame Synchronization Event 15
8	FSEVT16	Frame Synchronization Event 16
9	FSEVT17	Frame Synchronization Event 17
10	FSEVT18	Frame Synchronization Event 18
11	FSEVT19	Frame Synchronization Event 19
12	FSEVT20	Frame Synchronization Event 20
13	FSEVT21	Frame Synchronization Event 21
14	FSEVT22	Frame Synchronization Event 22
15	FSEVT23	Frame Synchronization Event 23
16	FSEVT24	Frame Synchronization Event 24
17	FSEVT25	Frame Synchronization Event 25
18	FSEVT26	Frame Synchronization Event 26
19	FSEVT27	Frame Synchronization Event 27
20	FSEVT28	Frame Synchronization Event 28
21	RIOINT0	RapidIO Interrupt 0
22	RIOINT1	RapidIO Interrupt 1
23	RIOINT2	RapidIO Interrupt 2
24	RIOINT3	RapidIO Interrupt 3
25	RIOINT4	RapidIO Interrupt 4
26	RIOINT5	RapidIO Interrupt 5
27	RIOINT7	RapidIO Interrupt 7
28	MACINT0	Ethernet MAC Interrupt
29	MACRXINT0	Ethernet MAC Interrupt
30	MACTXINT0	Ethernet MAC Interrupt
31	MACINT1	Ethernet MAC Interrupt
32	MACRXINT1	Ethernet MAC Interrupt
33	MACTXINT1	Ethernet MAC Interrupt
34	MACINT2	Ethernet MAC Interrupt
35	MACRXINT2	Ethernet MAC Interrupt
36	MACTXINT2	Ethernet MAC Interrupt
37	SEMERR0	Semaphore Error Interrupt
38	SEMERR1	Semaphore Error Interrupt
39	SEMERR2	Semaphore Error Interrupt
40:42	Unused	Reserved
43	TINT3L	Timer3 Interrupt Low

Table 14. Input Events to CIC3 (continued)

Number	Name	Description
44	TINT3H	Timer3 Interrupt High
45	TINT4L	Timer4 Interrupt Low
46	TINT4H	Timer4 Interrupt High
47	TINT5L	Timer5 Interrupt Low
48	TINT5H	Timer5 Interrupt High
49	AIF_TEVT0	AIF Logic Analyzer Trace Event
50	AIF_TEVT1	AIF Logic Analyzer Trace Event
51:52	Unused	Reserved
53	GPINT0	GPIO Event
54	GPINT1	GPIO Event
55	GPINT2	GPIO Event
56	GPINT3	GPIO Event
57	GPINT4	GPIO Event
58	CIC0_EVT14	CIC_EVT_o[14] from Chip Interrupt Controller[0]
59	CIC0_EVT15	CIC_EVT_o[15] from Chip Interrupt Controller[0]
60	CIC1_EVT14	CIC_EVT_o[14] from Chip Interrupt Controller[1]
61	CIC1_EVT15	CIC_EVT_o[15] from Chip Interrupt Controller[1]
62	CIC2_EVT14	CIC_EVT_o[14] from Chip Interrupt Controller[2]
63	CIC2_EVT15	CIC_EVT_o[15] from Chip Interrupt Controller[2]

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