Interrupt from PRU to R5F

Processors Platform Software

Exported on 07/08/2025

Table of Contents

Initializing and Handling Interrupt on R5F:

First you will have to configure the interrupt using SysConfig (More information at: AM64x MCU+ SDK: PRUICSS (ti.com)[1])

In SysConfig: Under PRUICSS module select ICSSG Instance and then

Select an interrupt Event number from 16 to 31 under PRUICSS > ICSSG0 INTC Internal Signals Mapping. Can proceed with default mapping for Channel and host interrupt. (TRM 6.4.7.1)

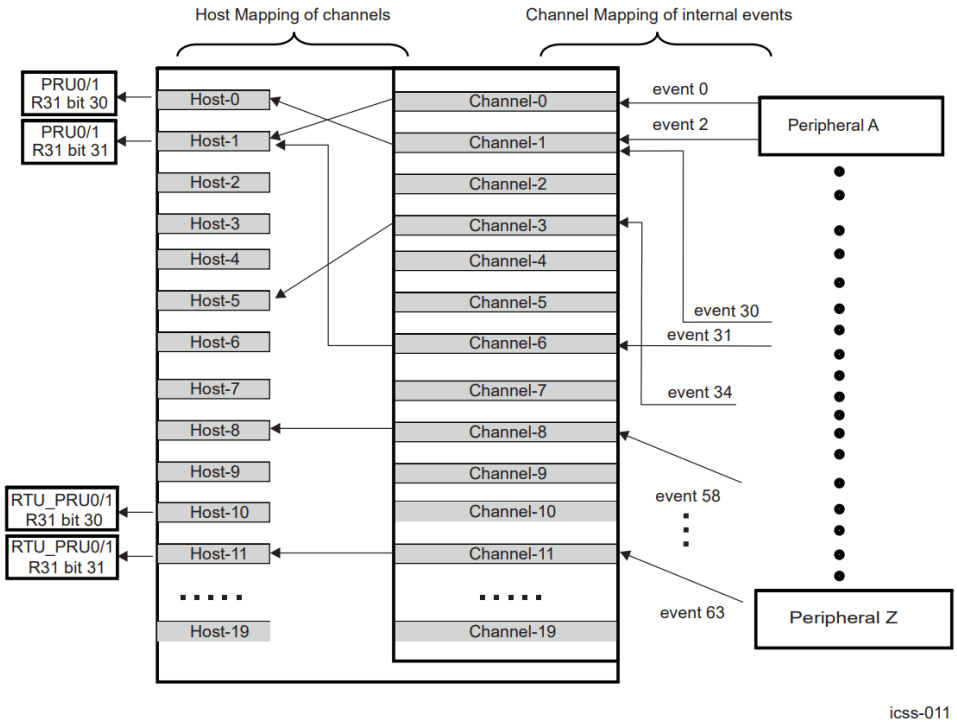

Figure 6-190. PRU_ICSSG Interrupt Controller Block Diagram

From TRM 9.4.1.3 R5FSS0_CORE1 Interrupt Map:

R5FSS0_CORE0_INTR_IN_120	120	PRU_ICSSG0_PR1_HOST_INTR_PEND_0
R5FSS0_CORE0_INTR_IN_121	121	PRU_ICSSG0_PR1_HOST_INTR_PEND_1
R5FSS0_CORE0_INTR_IN_122	122	PRU_ICSSG0_PR1_HOST_INTR_PEND_2
R5FSS0_CORE0_INTR_IN_123	123	PRU_ICSSG0_PR1_HOST_INTR_PEND_3
R5FSS0_CORE0_INTR_IN_124	124	PRU_ICSSG0_PR1_HOST_INTR_PEND_4
R5FSS0_CORE0_INTR_IN_125	125	PRU_ICSSG0_PR1_HOST_INTR_PEND_5
R5FSS0_CORE0_INTR_IN_126	126	PRU_ICSSG0_PR1_HOST_INTR_PEND_6
R5FSS0_CORE0_INTR_IN_127	127	PRU_ICSSG0_PR1_HOST_INTR_PEND_7

1 https://software-dl.ti.com/mcu-plus-sdk/esd/AM64X/08_02_00_31/exports/docs/api_guide_am64x/DRIVERS_PRUICSS_PAGE.html#autotoc_md612

Host interrupts 2-9 (from 0 to 19 as in Fig 6-190 above) are exported to system interrupts and correspond to PRU_ICSSG0_PR1_HOST_INTR_PEND_0-PRU_ICSSG0_PR1_HOST_INTR_PEND_7 and create interrupt on R5F0-0 on input interrupt lines 120-127.

Initialize the interrupts as: (Refer: AM64x MCU+ SDK: PRUICSS (ti.com)[2])

For ICSS internal event number 16 mapped to channel 2 → host interrupt 2, sample code snippet:

```
    gPruIcss0Handle = PRUICSS_open(CONFIG_PRU_ICSS0);
    status = PRUICSS_intcInit(gPruIcss0Handle, &icss0_intc_initdata);
    DebugP_assert(SystemP_SUCCESS == status);

    status = PRUICSS_registerIrqHandler(gPruIcss0Handle,
                                        0,      /*  pruEvtoutNum
PRU_ICSSG0_PR1_HOST_INTR_PEND_0 = 0, .... PRU_ICSSG0_PR1_HOST_INTR_PEND_7 = 7 */
                                        120,    /*  r5fIntrNum   corresponding r5f input
interrupt line */
                                        1,      /*  eventNum     */
                                        0,      /*  wait_enable  */
                                        &ISR
                                        );
    DebugP_assert(SystemP_SUCCESS == status);

void ISR(void *args)
{
    // ISR code here
    PRUICSS_clearEvent(gPruIcss0Handle, sysEventNum); /* sysEventNum = PRU internal
event number from 16 to 31 */
}
```

2 https://software-dl.ti.com/mcu-plus-sdk/esd/AM64X/08_02_00_31/exports/docs/api_guide_am64x/DRIVERS_PRUICSS_PAGE.html#autotoc_md612

Generating interrupt from PRU:

From TRM 6.4.5.2.2.2:

Simultaneously writing a ‘1’ to pru_r31_vec_valid (R31 bit 5) and a channel number from 0 to 15 to pru_r31_vec[3:0] (R31 bits 3-0) creates a pulse on the output of the corresponding pr_pru_mst_intr[x]_intr_req INTC system event. For example, writing ‘100000’ will generate a pulse on prk_pru_mst_intr[0]_intr_req, writing ‘100001’ will generate a pulse on prk_pru_mst_intr[1]_intr_req, and so on to where writing ‘101111’ will generate a pulse on prk_pru_mst_intr[15]_intr_req and writing ‘0xxxxx’ will not generate any system event pulses. The output values from both PRU cores in a subsystem are ORed together. The output channels 0-15 are connected to the INTC system events 16-31, respectively. This allows the PRU to assert one of the system events 16-31 by writing to its own R31 register. The system event is used to either post a completion event to one of the host CPUs (Arm) or to signal the other PRU. The host to be signaled is determined by the system interrupt to interrupt channel mapping (programmable). The 16 events are named as prk_pru_mst_intr<15:0>_intr_req. See the , PRU_ICSSG Interrupt Requests Mapping, in the section, PRU_ICSSG Local Interrupt Controller, for more details.

So:

```
    ldi     r31.b0, 0x20 + 0x0                  ; creates event 16 to interrupt r5f
core
```

This is all we need to do to trigger interrupt if the interrupt settings have already been configured by the r5f core.

Table 6-425. PRU_ICSSG Interrupt Requests

Interrupt Requests					
Module Instance	Module Interrupt Signal	Destination Interrupt Input	Destination	Type	Description
PRU_ICSSG0	PRU_ICSSG0_PR1_HOST_INTR_PEND_0	GICSS0_SPI_IN_120	COMPUTE_CLUSTER0	Level	Host interrupt 0
		R5FSS0_CORE0_INTR_IN_120	R5FSS0_CORE0		
		R5FSS0_CORE1_INTR_IN_120	R5FSS1_CORE1		
		R5FSS1_CORE0_INTR_IN_120	R5FSS1_CORE0		
		R5FSS1_CORE1_INTR_IN_120	R5FSS1_CORE1		
	PRU_ICSSG0_PR1_HOST_INTR_PEND_1	GICSS0_SPI_IN_121	COMPUTE_CLUSTER0	Level	Host interrupt 1
		R5FSS0_CORE0_INTR_IN_121	R5FSS0_CORE0		
		R5FSS0_CORE1_INTR_IN_121	R5FSS0_CORE1		
		R5FSS1_CORE0_INTR_IN_121	R5FSS1_CORE0		
		R5FSS1_CORE1_INTR_IN_121	R5FSS1_CORE1		
	PRU_ICSSG0_PR1_HOST_INTR_PEND_2	GICSS0_SPI_IN_122	COMPUTE_CLUSTER0	Level	Host interrupt 2
		R5FSS0_CORE0_INTR_IN_122	R5FSS0_CORE0		
		R5FSS0_CORE1_INTR_IN_122	R5FSS0_CORE1		
		R5FSS1_CORE0_INTR_IN_122	R5FSS1_CORE0		
		R5FSS1_CORE1_INTR_IN_122	R5FSS1_CORE1		
	PRU_ICSSG0_PR1_HOST_INTR_PEND_3	GICSS0_SPI_IN_123	COMPUTE_CLUSTER0	Level	Host interrupt 3
		R5FSS0_CORE0_INTR_IN_123	R5FSS0_CORE0		
		R5FSS0_CORE1_INTR_IN_123	R5FSS0_CORE1		
		R5FSS1_CORE0_INTR_IN_123	R5FSS1_CORE0		
		R5FSS1_CORE1_INTR_IN_123	R5FSS1_CORE1		
	PRU_ICSSG0_PR1_HOST_INTR_PEND_4	GICSS0_SPI_IN_124	COMPUTE_CLUSTER0	Level	Host interrupt 4
		R5FSS0_CORE0_INTR_IN_124	R5FSS0_CORE0		
		R5FSS0_CORE1_INTR_IN_124	R5FSS0_CORE1		
		R5FSS1_CORE0_INTR_IN_124	R5FSS1_CORE0		
		R5FSS1_CORE1_INTR_IN_124	R5FSS1_CORE1		

6.4.7.1 PRU_ICSSG Interrupt Controller Functional Description

The PRU_ICSSG INTC supports up to 160 interrupts from different peripherals (including 64 internal interrupts from PRU_ICSSG located interrupt sources). The INTC maps these events to 20 channels inside the INTC (see Figure 6-194). Interrupts from these 20 channels are further mapped to 20 Host Interrupts.

- Any of the 160 internal interrupts can be mapped to any of the 20 channels.
- Multiple interrupts can be mapped to a single channel.
- An interrupt should not be mapped to more than one channel.
- Any of the 20 channels can be mapped to any of the 20 host interrupts. It is recommended to map channel "x" to host interrupt "x", where x is from 0 to 19.
- A channel should not be mapped to more than one host interrupt
- For channels mapping to the same host interrupt, lower number channels have higher priority.
- For interrupts on same channel, priority is determined by the hardware interrupt number. The lower the interrupt number, the higher the priority.
- Host Interrupt 0 is connected to bit 30 in register 31 (R31) of PRU0 and PRU1 in parallel.
- Host Interrupt 1 is connected to bit 31 in register 31 (R31) for PRU0 and PRU1 in parallel.
- Host Interrupts 2 through 9 exported from PRU_ICSSG and mapped to device level interrupt controllers.
- Host Interrupt 10 is connected to bit 30 in register 31 (R31) to both RTU_PRU0 and RTU_PRU1 in parallel.
- Host Interrupt 11 is connected to bit 31 in register 31 (R31) to both RTU_PRU0 and RTU_PRU1 in parallel.
- Host Interrupts 12 through 19 are connected to each of the 6 Task Managers.

Interrupt Input Line	Interrupt ID	Source Interrupt
ICSSG0 Interrupts		
R5FSS0_CORE0_INTR_IN_120	120	PRU_ICSSG0_PR1_HOST_INTR_PEND_0
R5FSS0_CORE0_INTR_IN_121	121	PRU_ICSSG0_PR1_HOST_INTR_PEND_1
R5FSS0_CORE0_INTR_IN_122	122	PRU_ICSSG0_PR1_HOST_INTR_PEND_2
R5FSS0_CORE0_INTR_IN_123	123	PRU_ICSSG0_PR1_HOST_INTR_PEND_3
R5FSS0_CORE0_INTR_IN_124	124	PRU_ICSSG0_PR1_HOST_INTR_PEND_4
R5FSS0_CORE0_INTR_IN_125	125	PRU_ICSSG0_PR1_HOST_INTR_PEND_5
R5FSS0_CORE0_INTR_IN_126	126	PRU_ICSSG0_PR1_HOST_INTR_PEND_6
R5FSS0_CORE0_INTR_IN_127	127	PRU_ICSSG0_PR1_HOST_INTR_PEND_7
ICSSG1 Interrupts		
R5FSS0_CORE0_INTR_IN_248	248	PRU_ICSSG1_PR1_HOST_INTR_PEND_0
R5FSS0_CORE0_INTR_IN_249	249	PRU_ICSSG1_PR1_HOST_INTR_PEND_1
R5FSS0_CORE0_INTR_IN_250	250	PRU_ICSSG1_PR1_HOST_INTR_PEND_2
R5FSS0_CORE0_INTR_IN_251	251	PRU_ICSSG1_PR1_HOST_INTR_PEND_3
R5FSS0_CORE0_INTR_IN_252	252	PRU_ICSSG1_PR1_HOST_INTR_PEND_4
R5FSS0_CORE0_INTR_IN_253	253	PRU_ICSSG1_PR1_HOST_INTR_PEND_5
R5FSS0_CORE0_INTR_IN_254	254	PRU_ICSSG1_PR1_HOST_INTR_PEND_6
R5FSS0_CORE0_INTR_IN_255	255	PRU_ICSSG1_PR1_HOST_INTR_PEND_7

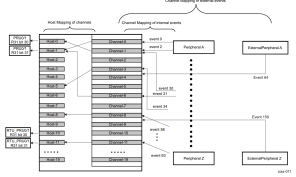

Figure 6-194. PRU_ICSSG Interrupt Controller Block Diagram

Table 6-481. PRU_ICSSG0 and PRU_ICSSG1 Internal Interrupts

Event Number	PRU_ICSSG Internal Interrupt Signal	Source
31	pr1_pru_mst_intr[15]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
30	pr1_pru_mst_intr[14]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
29	pr1_pru_mst_intr[13]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
28	pr1_pru_mst_intr[12]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
27	pr1_pru_mst_intr[11]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
26	pr1_pru_mst_intr[10]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
25	pr1_pru_mst_intr[9]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
24	pr1_pru_mst_intr[8]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
23	pr1_pru_mst_intr[7]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
22	pr1_pru_mst_intr[6]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
21	pr1_pru_mst_intr[5]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
20	pr1_pru_mst_intr[4]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
19	pr1_pru_mst_intr[3]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
18	pr1_pru_mst_intr[2]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
17	pr1_pru_mst_intr[1]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1
16	pr1_pru_mst_intr[0]_intr_req	PRU_ICSSG1 PRU0, PRU_ICSSG1 PRU1

Table 6-435. Event Interface Mapping (R31) Field Descriptions

Bit	Field	Description
31-6	Reserved	
5	pru<n>_r31_vec_valid	Valid strobe for vector output
4	Reserved	
3-0	pru<n>_r31_vec[3:0]	Vector output

Simultaneously writing a ‘1’ to pru<n>_r31_vec_valid (R31 bit 5) and a channel number from 0 to 15 to pru<n>_r31_vec[3:0] (R31 bits 3-0) creates a pulse on the output of the corresponding pr<k>_pru_mst_intr[x]_intr_req INTC system event. For example, writing ‘100000’ will generate a pulse on prk_pru_mst_intr[0]_intr_req, writing ‘100001’ will generate a pulse on prk_pru_mst_intr[1]_intr_req, and so on to where writing ‘101111’ will generate a pulse on prk_pru_mst_intr[15]_intr_req and writing ‘0xxxxx’ will not generate any system event pulses. The output values from both PRU cores in a subsystem are ORed together.

The output channels 0-15 are connected to the INTC system events 16-31, respectively. This allows the PRU to assert one of the system events 16-31 by writing to its own R31 register. The system event is used to either post a completion event to one of the host CPUs (Arm) or to signal the other PRU. The host to be signaled is determined by the system interrupt to interrupt channel mapping (programmable). The 16 events are named as prk_pru_mst_intr<15:0>_intr_req. See the , *PRU_ICSSG Interrupt Requests Mapping*, in the section, *PRU_ICSSG Local Interrupt Controller*, for more details.