OMAP5912 Multimedia Processor Interrupts Reference Guide

Literature Number: SPRU757B
October 2004

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
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
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
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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

Preface

Read This First

About This Manual

This document describes the interrupts of the OMAP5912 multimedia processor.

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.

Related Documentation From Texas Instruments

Documentation that describes the OMAP5912 device, related peripherals, and other technical collateral, is available in the OMAP5912 Product Folder on TI's website: www.ti.com/omap5912.

Trademarks

Contents

Figures

Tables

Interrupts

This document describes the interrupts of the OMAP5912 multimedia processor.

1 Interrupt Overview

Three level 2 interrupt controllers are used in OMAP5912:

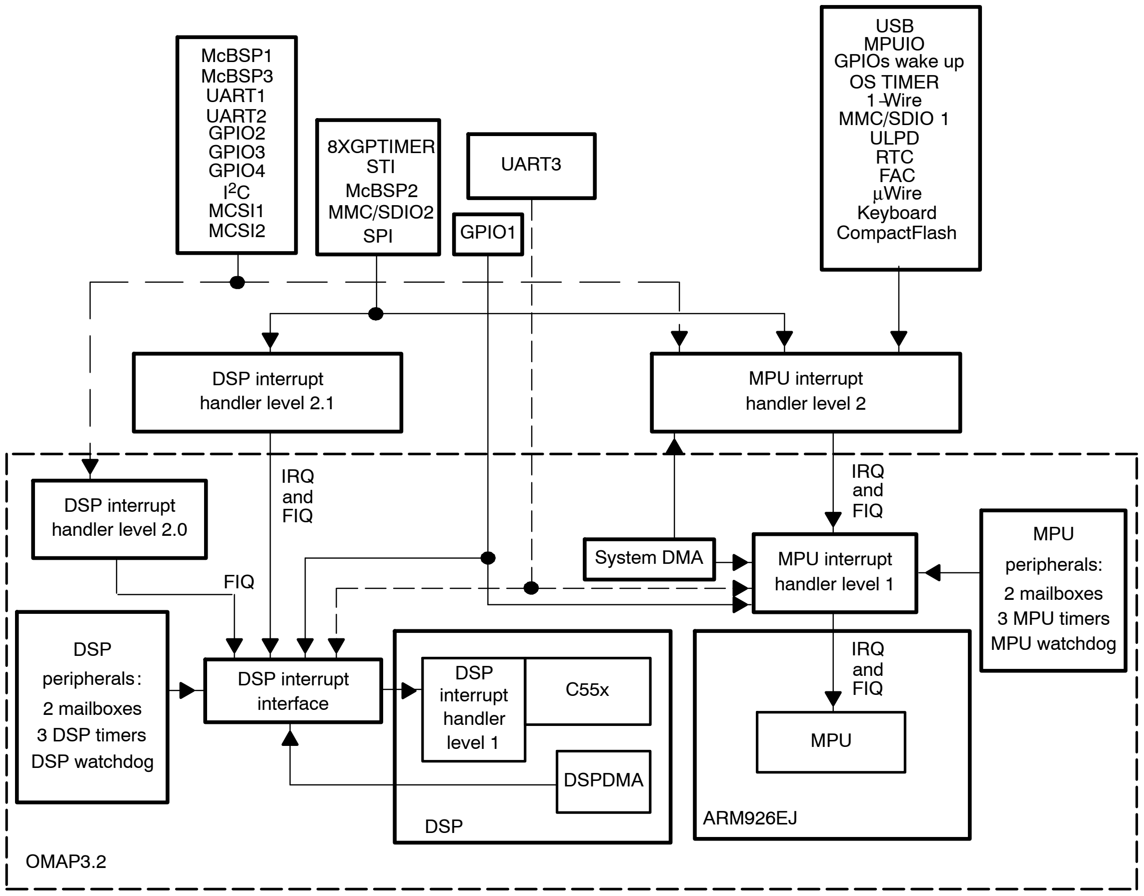
- ❑ One MPU level 2 interrupt handler (also referred to as MPU interrupt level 2) is implemented outside of OMAP3.2 and can handle 128 interrupts.
- ❑ One DSP level 2 interrupt handler (also referred to as DSP interrupt level 2.1) is instantiated outside of OMAP3.2 and can handle 64 interrupts.
- ❑ One OMAP3.2 DSP level 2 interrupt handler (referenced as DSP interrupt level 2.0) can handle 16 interrupts.

There are two level 1 interrupt controllers in OMAP3.2 and the DSP:

- ❑ One MPU level 1 interrupt handler that can handle 31 interrupts (OMAP3.2).
- ❑ One DSP level 1 interrupt handler that can handle 16 interrupts (DSP).

Figure 1 shows which peripheral module can trigger an interrupt in each processor. Some peripherals can generate interrupts to both processors. There are 160 interrupts on the MPU and 98 interrupts on the DSP side.

Figure 1. Interrupt Interconnect


McBSP1
McBSP3
UART1
UART2
GPIO2
GPIO3
GPIO4
I2C
MCSI1
MCSI2
8XGPTIMER
STI
McBSP2
MMC/SDIO2
SPI
UART3
GPIO1
USB
MPUIO
GPIOs wake up
OS TIMER
1-Wire
MMC/SDIO 1
ULPD
RTC
FAC
µWire
Keyboard
CompactFlash
DSP interrupt handler level 2.1
MPU interrupt handler level 2
DSP interrupt handler level 2.0
IRQ and FIQ
FIQ
System DMA
MPU interrupt handler level 1
MPU peripherals: 2 mailboxes 3 MPU timers MPU watchdog
DSP peripherals: 2 mailboxes 3 DSP timers DSP watchdog
DSP interrupt interface
DSP interrupt handler level 1
C55x
DSPDMA
DSP
MPU
ARM926EJ
OMAP3.2

Two DSP level 2 interrupt handlers connect directly to the DSP interrupt interface and then to the DSP level 1 interrupt handler. One MPU level 2 interrupt handler connects in parallel to the MPU level 1 interrupt handler. For more detail on the architecture and programming model of the level 2 interrupt handler, see Section 2, *Interrupt Controller*.

There are four groups of shared peripherals. Most of the shared peripherals connect to DSP/MPU level 2 interrupt handlers. Two exceptions are the GPIO1 and UART3, which connect to the MPU level 1 handler and the DSP level 1 handler.

Table 1 through Table 5 include the default priority and the required sensitivity to be programmed by software per interrupt line for each interrupt handler (L1 and L2). When the sensitivity is not precise, it must be considered as edge. The sensitivity depends on the peripheral type.

1.1 DSP Interrupt Mapping

On the DSP side are 98 interrupt input lines, 18 on the first level, and 80 on the second level.

Table 1. DSP Level 1 Interrupt Mapping

Priority	DSP Soft Interrupt	DSP Hard Interrupt	Location Vectors (Hex/Byte)	Function
1	SINT0	INT0	0	Reset
2	SINT1	INT1	8	Nonmaskable interrupt
3	SINT2	INT2	10	Emulator/test interrupt
5	SINT3	INT3	18	Level 2.0 interrupt handler
6	SINT4	INT4	20	TC_ABORT interrupt
7	SINT5	INT5	28	MAILBOX 1
9	SINT6	INT6	30	Level 2.1 interrupt handler FIQ
10	SINT7	INT7	38	IRQ2_GPIO1
11	SINT8	INT8	40	DSP timer 3 interrupt
13	SINT9	INT9	48	DMA_channel_1 interrupt
14	SINT10	INT10	50	MPUI interrupt
15	SINT11	INT11	58	Reserved
17	SINT12	INT12	60	UART3
18	SINT13	INT13	68	DSP watchdog interrupt
21	SINT14	INT14	70	DMA_channel_4 interrupt
22	SINT15	INT15	78	DMA_channel_5 interrupt
4	SINT16	INT16	80	STIO interrupt
8	SINT17	INT17	88	Level 2.1 interrupt handler IRQ
12	SINT18	INT18	90	DMA_channel_0 interrupt
16	SINT19	INT19	98	MAILBOX 2
19	SINT20	INT20	A0	DMA_channel_2 interrupt
20	SINT21	INT21	A8	DMA_channel_3 interrupt
23	SINT22	INT22	B0	DSP timer 2 interrupt

Table 1. DSP Level 1 Interrupt Mapping (Continued)

Priority	DSP Soft Interrupt	DSP Hard Interrupt	Location Vectors (Hex/Byte)	Function
24	SINT23	INT23	B8	DSP timer 1 interrupt
25	SINT24	INT24	C0	Bus error interrupt # 25 BERR
26	SINT25	INT25	C8	Emulator interrupt # 26 DLOG
27	SINT26	INT26	D0	Emulator interrupt # 27 RTOS
28	SINT27	INT27	D8	Software interrupt #28
29	SINT28	INT28	E0	Software interrupt #29
30	SINT29	INT29	E8	Software interrupt #30
31	SINT30	INT30	F0	Software interrupt #31
32	SINT31	INT31	F8	Software interrupt #32

1.1.1 DSP Level 2 Interrupt Handler

The system has two DSP level 2 interrupt handlers. One is in the OMAP3.2 gigacell (the DSP level 2.0 interrupt handler), and the other is outside the OMAP3.2 gigacell (the DSP level 2.1 interrupt handler):

- The DSP level 2.0 interrupt handler handles 16 interrupt lines.
- The DSP level 2.1 interrupt handler handles 64 interrupt lines.

For more detail on the architecture and programming model of the level 2 interrupt handler, see Section 2, *Interrupt Controller*.

1.1.2 DSP Level 2 Interrupt Mapping

The default interrupt priority for the DSP interrupt handler level 1 can be remapped by MPU software only when DSP is held in reset. All level 2 interrupt lines have the same priority by default, which DSP software can modify through the configuration register in the level 2 interrupt controller.

Table 2. DSP Level 2 Interrupt Mapping

Level 2.0 Interrupt Line	Mapping	Default Sensitivity Configuration
IRQ_0	McBSP3 TX	Level
IRQ_1	McBSP3 RX	Level
IRQ_2	McBSP1 TX	Level
IRQ_3	McBSP1 RX	Level
IRQ_4	UART2	Level
IRQ_5	UART1	Level
IRQ_6	MCSI1 TX	Level
IRQ_7	MCSI1 RX	Level
IRQ_8	MCSI2 TX	Level
IRQ_9	MCSI2 RX	Level
IRQ_10	MCSI1 frame error	Level
IRQ_11	MCSI2 frame error	Level
IRQ_12	IRQ2_GPIO2	level
IRQ_13	IRQ2_GPIO3	level
IRQ_14	IRQ2_GPIO4	level
IRQ_15	I^2C	Level

Table 3. DSP Level 2.1 Interrupt Mapping

Level 2.1 Interrupt Line	Mapping	Default Sensitivity Configuration
IRQ_0	NAND flash interrupt	Level
IRQ_1	GPTIMER1	Level
IRQ_2	GPTIMER2	Level
IRQ_3	GPTIMER3	Level
IRQ_4	GPTIMER4	Level
IRQ_5	GPTIMER5	Level
IRQ_6	GPTIMER6	Level

Table 3. DSP Level 2.1 Interrupt Mapping (Continued)

Level 2.1 Interrupt Line	Mapping	Default Sensitivity Configuration
IRQ_7	GPTIMER7	Level
IRQ_8	GPTIMER8	Level
IRQ_9	Reserved	–––––
IRQ_10	McBSP2 TX	Level
IRQ_11	McBSP2 RX	Level
IRQ_12	MCSI1_RST_INT	Level
IRQ_13	MCSI2_RST_INT	Level
IRQ_14	MMC/SDIO2	Level
IRQ_15	SPI	Level
IRQ_16– IRQ_47	Reserved	–––––
IRQ_48 … IRQ_63	Free	–––––

For each interrupt, the user must configure the SENS_nEDGE bit in the corresponding interrupt level register (ILR) according to whether the interrupt is edge- or level-sensitive. See Table 13 for more details.

1.1.3 MPU Interrupt Mapping

The interrupt priority is not hard-coded on the MPU side. Thus, the software must define all interrupt priorities for both level interrupt handlers. (See Table 4.)

Table 4. MPU Level 1 Interrupt Mapping

Level 1 Interrupt Line	OMAP 5912 Mapping	Sensitivity
IRQ_0	Level 2 interrupt handler IRQ	Level
IRQ_1	Camera IF	Level
IRQ_2	Level 2 interrupt handler FIQ	Level
IRQ_3	External FIQ	User-defined

† These IRQs are available only when the DMA is in OMAP3.2 mode (i.e. not in OMAP3.1 compatibility mode). See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.
‡ These IRQs are available only when the DMA is in OMAP3.1 compatibility mode. See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.

Table 4. MPU Level 1 Interrupt Mapping (Continued)

Level 1 Interrupt Line	OMAP 5912 Mapping	Sensitivity
IRQ_4	McBSP2 TX	Edge
IRQ_5	McBSP2 RX	Edge
IRQ_6	IRQ_RTDX (emulation event)	Edge
IRQ_7	IRQ_DSP_MMU_ABORT	Level
IRQ_8	IRQ_HOST_INT	Edge
IRQ_9	IRQ_ABORT	Level
IRQ_10	IRQ_DSP_MAILBOX1	Level
IRQ_11	IRQ_DSP_MAILBOX2	Level
IRQ_12	IRQ_LCD_LINE	Level
IRQ_13	Reserved	–––––
IRQ_14	IRQ1_GPIO1	Level
IRQ_15	UART3	Level
IRQ_16	IRQ_TIMER3	Edge
IRQ_17	GPTIMER1	Level
IRQ_18	GPTIMER2	Level
IRQ_19	IRQ_DMA_CH0†/IRQ_DMA_CH0_CH6‡	Level
IRQ_20	IRQ_DMA_CH1†/IRQ_DMA_CH1_CH7‡	Level
IRQ_21	IRQ_DMA_CH2†/IRQ_DMA_CH2_CH8‡	Level
IRQ_22	IRQ_DMA_CH3	Level
IRQ_23	IRQ_DMA_CH4	Level
IRQ_24	IRQ_DMA_CH5	Level
IRQ_25	IRQ_DMA_CH_LCD	Level
IRQ_26	IRQ_TIMER1	Edge
IRQ_27	IRQ_WD_TIMER	Edge

† These IRQs are available only when the DMA is in OMAP3.2 mode (i.e. not in OMAP3.1 compatibility mode). See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.
‡ These IRQs are available only when the DMA is in OMAP3.1 compatibility mode. See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.

Table 4. MPU Level 1 Interrupt Mapping (Continued)

Level 1 Interrupt Line	OMAP 5912 Mapping	Sensitivity
IRQ_28	Public TIPB abort	Level
IRQ_29	Reserved	–––––
IRQ_30	IRQ_TIMER2	Edge
IRQ_31	IRQ_LCD_CTRL	Level

† These IRQs are available only when the DMA is in OMAP3.2 mode (i.e. not in OMAP3.1 compatibility mode). See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.
‡ These IRQs are available only when the DMA is in OMAP3.1 compatibility mode. See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.

1.1.4 ARM926EJS Level 2 Interrupt Mapping

DSP and MPU share most of the peripherals. Therefore, almost all of the DSP level 2 interrupts also go to the MPU level 2 interrupt handlers. The MPU level 2 interrupt handlers are enabled to process 128 more interrupt lines outside the OMAP gigacell, for a total of 160 interrupt lines. (See Table 5.)

Table 5. MPU Level 2 Interrupt Mapping

Level 2 Interrupt Line	Mapping	Sensitivity
IRQ_0	FAC	Level
IRQ_1	Keyboard	Edge
IRQ_2	μWIRE TX	Level
IRQ_3	μWIRE RX	Level
IRQ_4	I^{2}C	Level
IRQ_5	MPUIO	Level
IRQ_6	USB HHC 1	Level
IRQ_7	USB HHC 2	Level
IRQ_8	USB_OTG	Level
IRQ_9	Reserved	–––––
IRQ_10	McBSP3 TX	Edge
IRQ_11	McBSP3 RX	Edge

† These IRQs are available only when the DMA is in OMAP3.2 mode (i.e. not in OMAP3.1 compatibility mode). See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.

Table 5. MPU Level 2 Interrupt Mapping (Continued)

Level 2 Interrupt Line	Mapping	Sensitivity
IRQ_12	McBSP1 TX	Edge
IRQ_13	McBSP1 RX	Edge
IRQ_14	UART1	Level
IRQ_15	UART2	Level
IRQ_16	MCSI1 combined TX/RX/Frame error/RST	Level
IRQ_17	MCSI2 combined TX/RX/Frame error/RST	Level
IRQ_18	Free	–––––
IRQ_19	Reserved	–––––
IRQ_20	USB W2FC Geni it	Level
IRQ_21	1-Wire	Level
IRQ_22	OS timer	Edge
IRQ_23	MMC/SDIO1	Level
IRQ_24	32-kHz gauging IRQ/USB client wakeup IRQ	Level/Edge
IRQ_25	RTC periodical timer	Edge
IRQ_26	RTC alarm	Level
IRQ_27	Reserved	–––––
IRQ_28	DSP_MMU_IRQ	Level
IRQ_29	USB W2FC IRQ_ISO_ON	Level
IRQ_30	USB W2FC IRQ_NON_ISO_ON	Level
IRQ_31	McBSP2 RX OVERFLOW	Level
IRQ_32–IRQ_33	Reserved	–––––
IRQ_34	GPTIMER3	Level
IRQ_35	GPTIMER4	Level
IRQ_36	GPTIMER5	Level

† These IRQs are available only when the DMA is in OMAP3.2 mode (i.e. not in OMAP3.1 compatibility mode). See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.

Table 5. MPU Level 2 Interrupt Mapping (Continued)

Level 2 Interrupt Line	Mapping	Sensitivity
IRQ_37	GPTIMER6	Level
IRQ_38	GPTIMER7	Level
IRQ_39	GPTIMER8	Level
IRQ_40	IRQ1_GPIO2	Level
IRQ_41	IRQ1_GPIO3	Level
IRQ_42	MMC/SDIO2	Level
IRQ_43	CompactFlash	Edge
IRQ_44	COMMRX (emulation event)	Level
IRQ_45	COMMTX (emulation event)	Level
IRQ_46	Peripheral wake up	Level
IRQ_47	Free	Level
IRQ_48	IRQ1_GPIO4	Level
IRQ_49	SPI	Level
IRQ_50– IRQ_52	Reserved	–––––
IRQ_53	IRQ_DMA_CH6†	Level
IRQ_54	IRQ_DMA_CH7†	Level
IRQ_55	IRQ_DMA_CH8†	Level
IRQ_56	IRQ_DMA_CH9†	Level
IRQ_57	IRQ_DMA_CH10†	Level
IRQ_58	IRQ_DMA_CH11†	Level
IRQ_59	IRQ_DMA_CH12†	Level
IRQ_60	IRQ_DMA_CH13†	Level
IRQ_61	IRQ_DMA_CH14†	Level
IRQ_62	IRQ_DMA_CH15†	Level

† These IRQs are available only when the DMA is in OMAP3.2 mode (i.e. not in OMAP3.1 compatibility mode). See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.

Table 5. MPU Level 2 Interrupt Mapping (Continued)

Level 2 Interrupt Line	Mapping	Sensitivity
IRQ_63	Reserved	–––––
IRQ_64	Reserved (USB HHC2 suspend)	–––––
IRQ_65	Reserved	–––––
IRQ_66	Free	––––
IRQ_67–IRQ_90	Reserved	–––––
IRQ_91	SHA-1/MD5	Level
IRQ_92	RNG	Level
IRQ_93	RNGIDLE	Level
IRQ_94–IRQ_102	Reserved	–––––
IRQ_103 … IRQ_127	Free	–––––

† These IRQs are available only when the DMA is in OMAP3.2 mode (i.e. not in OMAP3.1 compatibility mode). See the Multimedia Processor Direct Memory Access (DMA) Support Reference Guide (literature number SPRU755) for more information.

For each interrupt, the user must configure the SENS_nEDGE bit in the corresponding interrupt level register (ILR) according to whether the interrupt is edge or level sensitive. See Table 13 for more details.

2 Interrupt Controllers (MPU Level 2 and DSP Level 2.1)

The MPU level 2 and DSP level 2.1 interrupt controllers have the same programming model. The only difference between the two is the number of interrupts each can handle (128 for MPU level 2 and 64 for DSP level 2.1). Throughout this section, both interrupt controllers are referred to collectively as the interrupt controller.

The MPU level 2 interrupt controller functional clock source is ARM_CK or ARM_CK/2, according to the ARM_CKCTL.ARM_INTHCK_SEL bit.

The DSP level 2.1 interrupt controller functional clock source is the DSPPER_CK.

The interrupt controller is able to handle interrupts coming from different functional blocks, prioritize them, and route them to a host. It generates one IRQ and one FIQ signal to the host. Both signals are active low-level interrupts, synchronous with the interrupt controller functional clock.

The interrupt controller can be programmed to assign different priorities and mask each interrupt independently. Each interrupt line can be programmed to be either level sensitive or edge triggered. The interrupt handler also provides an asynchronous signal to the host in order to have a way to wake up the system, in case an interrupt occurs when the clocks are turned off (system in idle state). See Figure 2 for the interrupt controller block diagram.

2.1 Functional Description

The interrupt controller provides prioritized and maskable interrupts to the host.

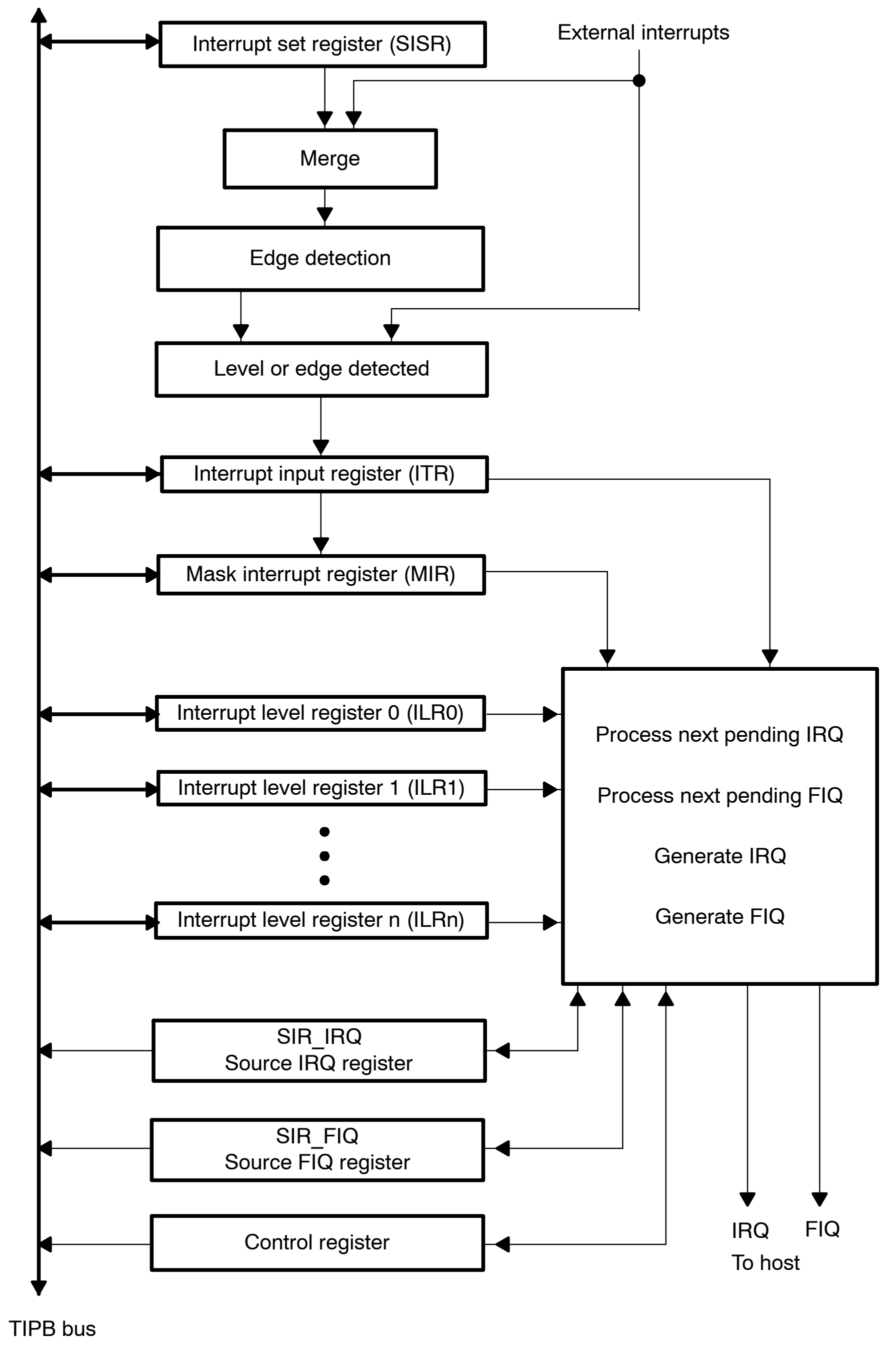
One interrupt level register (ILR) is associated with each incoming interrupt. It assigns a priority to the corresponding interrupt, determines whether it is to be level or edge sensitive, and selects which interrupt (FIQ or IRQ) it is to generate. If several interrupts have the same priority level, they are serviced in a predefined order (see Table 13).

For test purposes, the interrupt controller provides a set of software interrupt set registers (SISR). Each bit of these registers corresponds to an incoming interrupt line. By writing a1 to the targeted bit, an interrupt is generated if the corresponding ILR is set to edge sensitive. External interrupt requests and internal software requests are merged before being sent to the interrupt controller state machine.

Each incoming interrupt is routed either to the FIQ or the IRQ interrupt. The IRQ or FIQ outputs from the interrupt controller are reset by writing a 1 to the corresponding bit of the control register. Both the IRQ and FIQ are synchronous to the interrupt controller functional clock.

The interrupt handler can wake up the host asynchronously even if the functional clocks are turned off (host in idle state).

Figure 2. Interrupt Controller



2.1.1 Interrupt Processing Sequence

Although only the IRQ is discussed here, the sequence is the same for FIQ. The sequence depends, however, on the interrupt sensitivity (level or edge).

2.1.2 Edge-Triggered Interrupts

1) The interrupt controller module receives incoming interrupts and registers them in the ITR register.

2) The interrupt controller asserts the IRQ signal and begins priority calculation. When the highest priority interrupt is known, the source IRQ register (SIR_IRQ) is updated with the current interrupt number.

   The only way to deassert the IRQ that has been sent to the host is to set the NEW_IRQ_AGR bit in the control register.

3) The host, when it recognizes the interrupt, jumps to the interrupt routine.

4) Within the interrupt routine, the host reads the SIR_IRQ register to determine which interrupt line caused the interrupt. Reading the SIR_IRQ resets the interrupt bit in the ITR register. The IRQ/FIQ line, however, stays asserted. Based on the content of SIR_IRQ, the host executes specific code.

5) Before jumping out of the interrupt routine, the host must set the NEW_IRQ_AGR bit of the control register. Setting this bit deasserts the IRQ line and enables the interrupt controller to process any other pending interrupts.

2.1.3 Level-Sensitive Interrupts

1) The interrupt controller receives incoming interrupts. Level-sensitive interrupts are not registered. The interrupt controller assumes that a peripheral asserting a level-sensitive interrupt does not deassert it until the software directs it to do so.

2) The interrupt controller asserts the IRQ signal and begins priority calculation. When the highest priority interrupt is known, the SIR_IRQ register is updated to the current interrupt number.

   The only way to deassert the IRQ that has been sent to the host is to set the NEW_IRQ_AGR bit in the control register.

3) The host, when it recognizes the interrupt, jumps to the interrupt routine.

4) Within the interrupt routine, the host reads the SIR_IRQ register in the interrupt controller to determine which interrupt line caused the interrupt. Reading the SIR_IRQ has no effect on the ITR register for the level-sensitive interrupt, because the interrupt is still asserted at this point. Based on the content of SIR_IRQ, the host executes specific code.

5) Before jumping out of the interrupt routine, the software must:

   - Ensure that the peripheral that asserted the interrupt deasserts it.
   - Set the NEW_IRQ_AGR bit of the control register. Setting this bit deasserts the IRQ/FIQ line and enables the interrupt controller to process any pending interrupts.

2.1.4 Interrupt Latency

To minimize interrupt latency, an IRQ (or FIQ) is generated whenever an incoming interrupt is detected. Owing to internal resynchronization, the IRQ/FIQ generation requires three interrupt controller functional clock cycles to be generated.

The interrupt source calculation (priority handling and SIR register updating) is made in the background. The interrupt handler stalls any access to the SIR register until the interrupt source calculation is done.

To avoid unnecessary stalling, the interrupt source calculation does not depend on the number of incoming interrupts active at the same time. It depends instead on the total number of incoming interrupts. For the MPU interrupt handler, the interrupt source calculation requires 10 cycles, regardless of the number of active incoming interrupts. For the DSP side, the interrupt source calculation requires 6 cycles.

2.1.5 Interrupt Handler Sleep Mode

The MPU can shut off the system clock to save power. The system is awakened by an asynchronous event, which can be an interrupt. For proper system operation, the interrupt controller must ensure that no interrupt is generated when the system is going to power-saving (big or deep sleep) mode.

Before going to idle, the host sends an idle request (IDLE_REQ) to the interrupt controller. When the interrupt controller is ready to go to sleep, it sends back an idle acknowledge (IDLE_ACK) to signal that its functional clock can be safely shut down. IDLE_REQ/IDLE_ACK handshake is used to prevent IRQ/FIQ from occurring when the system goes into idle mode. Two different protocols manage the handshake: FORCE_WAKEUP and SMART_IDLE. The protocol used is defined by the value of the IDLE_MODE field in the OCP_CFG register (see Table 18).

2.1.6 Going to Sleep

The procedure used for going to sleep depends on the IDLE_MODE value.

Smart Idle Mode

In SMART_IDLE mode, when the IDLE_REQ signal is asserted (high level), the interrupt handler goes into an idle state on the next functional clock cycle, if no IRQ/FIQ is currently pending. If incoming interrupts are pending, they are processed before going into idle state. As long as the interrupt controller is in idle state, all incoming interrupts are ignored but still stored into the ITR register.

As soon as it is ensured that no IRQ/FIQ will be generated, the interrupt controller asserts the IDLE_ACK signal.

Upon receiving the IDLE_ACK signal, the power management is aware that no more interrupts are currently being processed and that clocks can safely be turned off.

Force Wake-up Mode

In force wake-up mode, when the IDLE_REQ signal is asserted, the interrupt controller asserts IDLE_ACK on the next OCP clock cycle regardless of the state of the current incoming interrupt and of any interrupt possibly being processed. It immediately masks all incoming interrupts and deasserts the IRQ/FIQ signal.

Waking Up

When the system wakes up, power management turns the clocks on again, and then deasserts the IDLE_REQ signal. At that moment, the interrupt controller becomes aware that its clocks are turned on. It deasserts IDLE_ACK and comes out of idle state. From this point on, IRQ/FIQ generation becomes possible again.

2.1.7 External Interrupt Asynchronous Path

When the system is asleep, the interrupt controller functional clock is shut off, the interrupt controller is in idle state, and IRQ/FIQ generation is no longer possible. However, in this case, a dedicated asynchronous path is activated in the interrupt controller to propagate interrupts to the clock manager.

Whenever the interrupt controller is idle, and an unmasked interrupt occurs, the interrupt controller asynchronously asserts a wakeup signal. Basically, this signal is an OR of all unmasked ITR bits. This signal notifies the clock manager that an external wake-up event has occurred.

When the interrupt controller functional clock is eventually turned on, the interrupt controller goes out of idle state and generation of IRQ/FIQ is enabled again. The interrupt waking up the system is *not* lost (although reading the SIR register does not necessarily give the waking interrupt number, if another higher priority interrupt was pending when the system was awakened).

2.1.8 Interrupt Global Masking

To avoid interrupting the software during the execution of critical routines, a global masking mechanism is implemented, controlled by the control register GLOBAL_MASK bit (see Figure 3) and the interrupt controller output signal IRQ_SECURE_MASK_N.

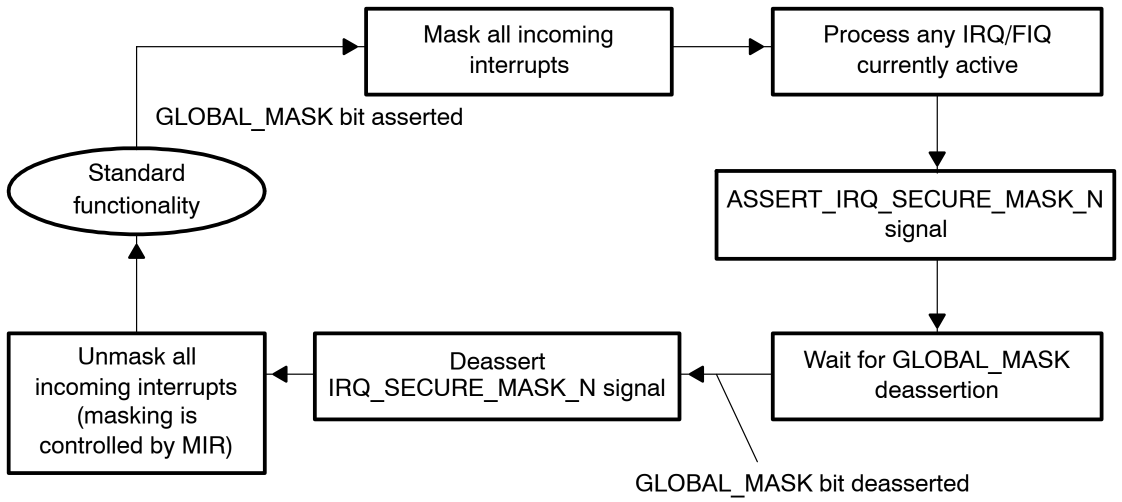
As long as the GLOBAL_MASK bit is set, all incoming interrupts are registered into ITR, but not processed. If IRQ or FIQ is asserted, the interrupt controller waits for the current IRQ/FIQ routine to complete.

When both IRQ and FIQ are deasserted, the interrupt controller asserts (sets to 0) the IRQ_SECURE_MASK_N signal. As long as this signal is asserted the interrupt controller must not assert IRQ or FIQ. Incoming interrupts must still be stored in the ITR register.

Upon deassertion of the GLOBAL_MASK bit, the interrupt controller first deasserts the IRQ_SECURE_MASK_N signal, then releases global masking (masking goes back under the control of the MIR registers), and returns to normal operation mode.

The IRQ_SECURE_MASK_N signal is synchronous of the interrupt controller functional clock.

Figure 3. Global Mask Bit Effect


Mask all incoming interrupts
Process any IRQ/FIQ currently active
GLOBAL_MASK bit asserted
Standard functionality
ASSERT_IRQ_SECURE_MASK_N signal
Unmask all incoming interrupts (masking is controlled by MIR)
Deassert IRQ_SECURE_MASK_N signal
Wait for GLOBAL_MASK deassertion
GLOBAL_MASK bit deasserted

2.1.9 Interrupt Spying

The immediate value of the ITR register, whose number is binary coded on the SPY_ITR_SEL input port asynchronously, is put on the SPY_ITR_OUT output port, regardless of the interrupt controller state.

2.1.10 Interrupt Controller Registers

All addresses in this section are byte addresses.

Regardless of register width, all addresses are aligned on 32-bit boundaries. This means that the address mapping does not depend on register width and that there are holes in the mapping for widths smaller than 32.

The mapping below describes only the basic set of registers. This mapping is duplicated according to the host width and number of incoming interrupts.

To ease software development and hardware implementation, duplication occurs at offset 0x100. This means that 64 locations stay unused at the end of each register section. Given that there are 11 address bits, this allows for a maximum of 8 sections, or 256 interrupts, if the data path is 32 bits wide.

The registers SIR_IRQ, SIR_FIQ, CONTROL, STATUS, OCP_CFG and INTH_REV are not duplicated, and are only available at offset 0x10, 0x14, 0x18, 0xA0, 0xA4, and 0xA8 (in the first section). These register addresses are reserved in the other sections. Writing at these locations has no effect, and reading them returns 0.

All register accesses are little endian.

All unused register bits are read as 0.

The OMAP programming model ensures that no posted write occurs in any writeable register.

Table 6 lists the interrupt controller registers. Table 7 through Table 20 describe the register bits.

Table 6. Interrupt Controller Registers

Register	Description	R/W	Offset
ITR	Interrupt register	R/W	0x00
MIR	Interrupt mask register	R/W	0x04
RESERVED	Reserved	R	0x08
RESERVED	Reserved	R	0x0C
SIR_IRQ	Interrupt encoded source register (IRQ)	R	0x10
SIR_FIQ	Interrupt encoded source register (FIQ)	R	0x14
CONTROL	Interrupt control register	R/W	0x18
	ILR Registers		
ILR0	Interrupt priority level register bit 0	R/W	0x1C
ILR1	Interrupt priority level register bit 1	R/W	0x20
ILR2	Interrupt priority level register bit 2	R/W	0x24
ILR3	Interrupt priority level register bit 3	R/W	0x28
ILR4	Interrupt priority level register bit 4	R/W	0x2C
ILR5	Interrupt priority level register bit 5	R/W	0x30
ILR6	Interrupt priority level register bit 6	R/W	0x34
ILR7	Interrupt priority level register bit 7	R/W	0x38
ILR8	Interrupt priority level register bit 8	R/W	0x3C
ILR9	Interrupt priority level register bit 9	R/W	0x40

Table 6. Interrupt Controller Registers (Continued)

Register	Description	R/W	Offset
	ILR Registers (Continued)		
ILR10	Interrupt priority level register bit 10	R/W	0x44
ILR11	Interrupt priority level register bit 11	R/W	0x48
ILR12	Interrupt priority level register bit 12	R/W	0x4C
ILR13	Interrupt priority level register bit 13	R/W	0x50
ILR14	Interrupt priority level register bit 14	R/W	0x54
ILR15	Interrupt priority level register bit 15	R/W	0x58
ILR16	Interrupt priority level register bit 16	R/W	0x5C
ILR17	Interrupt priority level register bit 17	R/W	0x60
ILR18	Interrupt priority level register bit 18	R/W	0x64
ILR19	Interrupt priority level register bit 19	R/W	0x68
ILR20	Interrupt priority level register bit 20	R/W	0x6C
ILR21	Interrupt priority level register bit 21	R/W	0x70
ILR22	Interrupt priority level register bit 22	R/W	0x74
ILR23	Interrupt priority level register bit 23	R/W	0x78
ILR24	Interrupt priority level register bit 24	R/W	0x7C
ILR25	Interrupt priority level register bit 25	R/W	0x80
ILR26	Interrupt priority level register bit 26	R/W	0x84
ILR27	Interrupt priority level register bit 27	R/W	0x88
ILR28	Interrupt priority level register bit 28	R/W	0x8C
ILR29	Interrupt priority level register bit 29	R/W	0x90
ILR30	Interrupt priority level register bit 30	R/W	0x94

Table 6. Interrupt Controller Registers (Continued)

Register	**Description**	**R/W**	**Offset**
	ILR Registers (Continued)		
ILR31	Interrupt priority level register bit 31	R/W	0x98
	Status and ID Register		
SISR	Software interrupt set register	W	0x9C
STATUS	Status register	R	0xA0
OCP_CFG	OCP configuration register	R/W	0xA4
INTH_REV	Interrupt controller revision ID	R	0xA8

2.1.11 Implementation

The MPU level 2 interrupt controller can handle 128 interrupts and has 32 bits wide registers. The DSP level 2.1 interrupt controller can handle 64 interrupts, and has 16 bits registers.

Both interrupt controllers have four full sets of registers.

In each set, the MPU interrupt controller has 32 ILR, mapped in the ranges 0x1C−0x98, 0x11C−0x198, 0x21C−0x298, and 0x31C–0x398. Each of these registers has nine relevant bits (read on the 9 LSB). In ILR, register 0 (@0x1C) controls interrupt 0, and register 127 (@0x398) controls interrupt 127.

In each set, the DSP interrupt controller has 16 ILR, mapped in the ranges 0x1C−0x58, 011C−0x158, 0x21C−0x258, and 0x31C−0x358. Each of these registers has eight relevant bits (read on the 8 LSB).

Table 7. Interrupt Input Register (ITR)

	DW-1†	**DW-2**	**DW-3**	**DW-4**	**DW-5**	**….**	**2**	**1**	**0**
@0x00	IRQ-DW-1						IRQ_2	IRQ_1	IRQ_0
Access	RW	RW	RW	RW	RW	RWs	RW	RW	RW
Default	0	0	0	0	0	0s	0	0	0

† DW = 32 for the MPU interrupt handler; DW =16 for the DSP interrupt handler.

If the incoming interrupt line number *n* is detected, the corresponding bit is set to 1. If the number of incoming interrupts is lower than DW, the MSB of this register is set to 0.

In case of an edge-sensitive interrupt, when the host accesses the SIR_IRQ or SIR_FIQ register, the bit corresponding to the currently active interrupt is reset. For level-sensitive interrupts, this bit is simply the interrupt line current state (after resynchronization).

The host can also clear each bit individually. To do this, it must write a 0 to the corresponding bit. Write access to this register is stalled as long as the actual register bit (on the functional clock domain) is not 0. Writing a 1 to any ITR bit has no effect.

This register is a status register that gives the state of every incoming interrupt regardless of which interrupt is currently being processed. Coherency between this register and the SIR_IRQ/SIR_FIQ registers is not ensured.

Table 8. Mask Interrupt Register (MIR)

	DW-1†	**DW-2**	**DW-3**	**DW-4**	**DW-5**	**….**	**2**	**1**	**0**
@0x04	MIR-DW-1						MIR_2	MIR_1	MIR_0
Access	RW	RW	RW	RW	RW	RWs	RW	RW	RW
Default	1	1	1	1	1	1s	1	1	1

† DW = 32 for the MPU interrupt handler; DW = 16 for the DSP interrupt handler.

This register masks each incoming interrupt by setting the corresponding bit to 1.

MIR operates after ITR. This means that occurrences of incoming interrupts are always stored in ITR.

Masking a particular interrupt does not deassert either IRQ or FIQ if it is active owing to this interrupt (providing that no other interrupts are active at the same time), regardless of whether the interrupt is level- or edge-sensitive. It only prevents subsequent incoming interrupts from being taken into account.

The write into this register must be done while there are no pending interrupts.

Table 9. Source IRQ Register (SIR_IRQ)

	CODE_NB_IT−1 … **0**†
@0x10	Active IRQ number (hexadecimal)
Access	R
Default	0

† CODE_NB_IT = 7 for the MPU interrupt handler; CODE_NB_IT = 6 for the DSP interrupt handler.

This register stores the currently active IRQ interrupt number (in hexadecimal). Reading this register clears the corresponding bit in the ITR register if the interrupt is set as edge sensitive. Unused bits are read as 0.

In case a priority calculation is ongoing for IRQ, read to this register is stalled until priority calculation completion.

Table 10. Source FIQ Register (SIR_FIQ)

	CODE_NB_IT−1 … 0
@0x14	Active FIQ number (hexadecimal)
Access	R
Default	0

This register stores the currently active FIQ interrupt number (in hexadecimal). Reading this register clears the corresponding bit in the ITR register if the interrupt is set as edge sensitive. Unused bits are read as 0.

In case a priority calculation is ongoing for FIQ, read to this register is stalled until priority calculation completion.

Table 11. Control Register

	DW-1…3	**2**	**1**	**0**
@0x18	Reserved	GLOBAL_MASK	NEW_FIQ_AGR	NEW_IRQ_AGR
Access	R	RW	W	W
Default	0	0	0	0

Table 12. Control Register Bit Descriptions

Name	Function
GLOBAL_MASK	Setting this bit to 1 has the following effect (in this order): All incoming or software generated (through SISR) interrupts are still stored in ITR, but neither FIQ nor IRQ are. When this bit is set *and* IRQ and FIQ are inactive *and* no IRQ/FIQ are to occur, the IRQ_SECURE_MASK_N signal is asserted. Resetting this bit has the following effect (in this order): IRQ_SECURE_MASK_N signal is deasserted. FIQ and IRQ generation become possible again (and all pending interrupts are processed normally). This behavior is shown in Figure 3.
NEW_FIQ_AGR	New FIQ agreement Writing a 1 resets FIQ output and enables a new FIQ generation. The FIQ output is reset if the corresponding bit of the ITR of the treated interrupt has been cleared or masked. Write access to this register bit is stalled until FIQ output is cleared. This bit is write only. Reading it always returns 0.
NEW_IRQ_AGR	New IRQ agreement Writing a 1 resets IRQ output and enables a new IRQ generation. The IRQ output is reset if the corresponding bit of ITR of the treated interrupt has been cleared or masked. Write access to this register bit is stalled until IRQ output is cleared. This bit is write only. Reading it always returns 0.

To allow for future evolution, when the software writes into this register all reserved bits must be written as 0.

Table 13. Interrupt Level Register (ILR)

	CODE_NB_IT+1 … 2[†]	**1**	**0**
@0x1C–0x98	PRIORITY	SENS_nEDGE	FIQ_nIRQ
Access	RW	RW	RW
Default	0	0	0

[†] CODE_NB_IT = 7 for the MPU interrupt handler; CODE_NB_IT = 6 for the DSP interrupt handler.

Table 14 describes the interrupt level register bits.

Table 14. Interrupt Level Register Bit Descriptions

Name	**Function**
PRIORITY	Defines the priority level when the corresponding interrupt is routed to IRQ or FIQ. 0 is the highest priority level. All bits set to 1 is the lowest priority level.
SENS_nEDGE	0: The corresponding interrupt is falling-edge sensitive. 1: The corresponding interrupt is low-level sensitive.
FIQ_nIRQ	0: The corresponding interrupt is routed to IRQ. 1: The corresponding interrupt is routed to FIQ.

There is one ILR per incoming interrupt.

Assuming that all interrupts have the same priority level and are active at the same time, the default order is IRQ_N, IRQ_N−1 , IRQ_N−2, … , IRQ_0.

Writes into this register must be done while there are no pending interrupts.

Table 15. Software Interrupt Set Register (SISR)

	DW-1†	**DW-2**	**DW-3**	**DW-4**	**DW-5**	**….**	**2**	**1**	**0**
@0x9C	$SISR_{DW-1}$						$SISR_2$	$SISR_1$	$SISR_0$
Access	W	W	W	W	W	Ws	W	W	W
Default	0	0	0	0	0	0s	0	0	0

† DW = 32 for the MPU interrupt handler; DW = 16 for the DSP interrupt handler.

Writing a 1 to any bit generates an interrupt to the host if the corresponding ILR register is set as edge trigger. Otherwise, no interrupt is generated.

A zero is always returned from a read to this register. External interrupts are merged (ORed) with the software interrupts before masking occurs.

Table 16. Status Register

	DW-1… 2†	**0**
@0xA0	Reserved	RESET_DONE
Access	R	R
Default	0	1

† DW = 32 for the MPU interrupt handler; DW = 16 for the DSP interrupt handler.

Table 17. Status Register Bit Description

Name	Function
RESET_DONE	0: Reset has not occurred (functional logic is currently being reset). 1: Reset has occurred. This bit concerns only the functional and OCP clock domains. When OCP clock domain is being reset, OCP accesses are stalled and this bit cannot be read.

Table 18. OCP_CFG Register

	DW-1… 5[†]	4	3	2	1	0
@0xA4	RESERVED	IdleMode		RE-SERVED	SOFT RESET	AUTOIDLE
Access		RW		R	W	R/W
Default	0	0	0	0	0	0

[†] DW = 32 for the MPU interrupt handler; DW = 16 for the DSP interrupt handler.

Table 19. OCP_CFG Register Bit Descriptions

Name	Function
IDLEMODE	Power management REQ/ACK control (see Section 2.1.5) 00: Force wake-up. An idle request is acknowledged unconditionally. 01: Reserved. Do not use. 10: Smart idle. Acknowledgement of the idle request occurs only if no incoming interrupts are pending. 11: Reserved. Do not use.
SOFTRESET	Writing 1 to this bit generates a software reset of the module. Both OCP and functional clock domain are reset. This bit is write only. Reading it always returns 0. To check reset completion, use the RESET_DONE bit in the status register.
AUTOIDLE	Internal OCP clock gating strategy. 0: OCP clock is free-running. 1: Automatic OCP clock gating is applied based on OCP interface activity.

To allow for future evolution, when the software writes into this register all reserved bits must be written as 0. These bits are currently read-only and are always read as 0, no matter what the value written. Future evolution, however, may have one or more of them as read/write.

Table 20. Interrupt Revision Register (INTH_REV)

	DW-1… 8[†]	7	6	5	4	3	2	1	0
@0xA8	Reserved	MAJOR_REV				MINOR_REV			
Access	R	R				R			
Default	0	Revision dependent				Revision dependent			

† DW = 32 for the MPU interrupt handler; DW = 16 for the DSP interrupt handler.

This register provides the revision number of the interrupt controller block.

3 Level 1 MPU Interrupt Handler

3.1 Description

The MPU interrupt handler allows up to 32 hosts that generate interrupts to connect to the MPU, which can accept only two interrupts: fast interrupt request (FIQ) and low-priority interrupt request (IRQ). You can also program the interrupt handler to assign different priorities and mask each interrupt, and you can program each interrupt line to be either edge-triggered or level-sensitive.

For power management, the clock manager can turn off the interrupt handler functional clock. A handshaking protocol is defined for the clock and reset module to idle or wake up the interrupt handler.

A key difference between the level 1 and level 2 MPU interrupt handler is that there is only one set of registers, some of which differ from the level 2 handlers.

3.1.1 Interrupt Control and Configuration

If an interrupt occurs, the ITR register stores the incoming interrupt in the corresponding bit. When there are several incoming interrupts, the MPU interrupt handler compares the priority level of the interrupts before sending an IRQ or FIQ to the MPU core. The selected interrupt's number is stored in SIR_IRQ or SIR_FIQ for the MPU to determine which interrupt service routine to execute. Reading either of these registers by the MPU resets the corresponding bit in ITR. The MPU can also clear each bit individually in ITR by writing a 0 to the corresponding bits. Writing a 1 keeps its previous value.

Each incoming interrupt can be masked individually by setting the corresponding bit in MIR to 1.

One interrupt level register (ILR) is associated with each incoming interrupt. ILR determines whether the interrupt is to be edge-triggered or level-sensitive and assigns it a priority level: 0 (the highest priority), 1, ... 30, 31 (the lowest priority). If several interrupts have the same priority level assigned, they are serviced in a predefined order: IRQ_31, IRQ_30, ..., IRQ_1, IRQ_0. ILR also allows routing each of the 32 interrupts to either FIQ or IRQ.

The IRQ or FIQ outputs can be reset by writing a 1 to the corresponding bit of the CONTROL_REG to enable new IRQ or FIQ generation. The writing also clears the SIR_IRQ or SIR_FIQ register. The corresponding bit in the ITR must be cleared before writing to the CONTROL_REG.

3.1.2 Software Interrupt

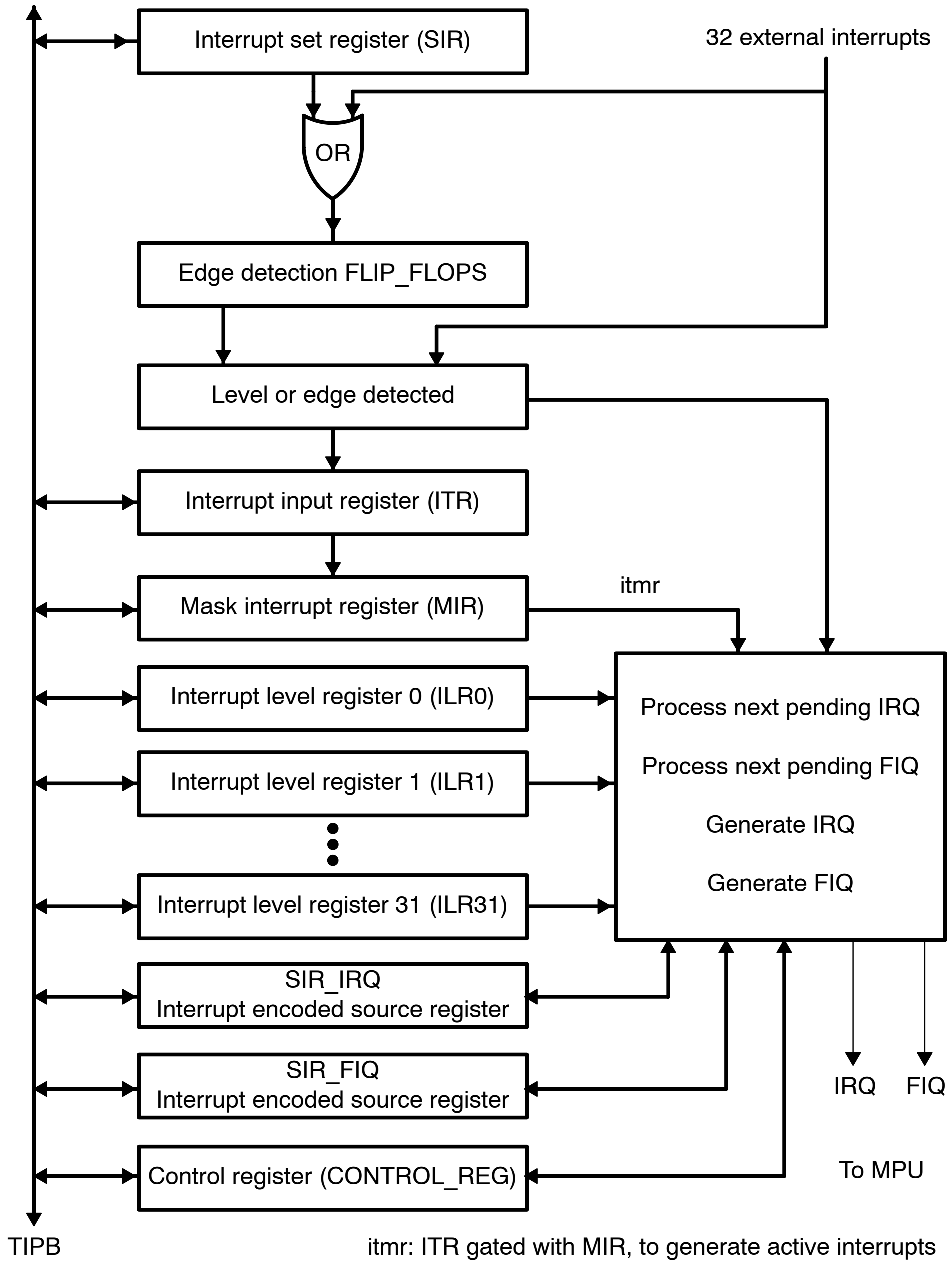
The interrupt handler also provides a 32-bit software interrupt register (SIR), which corresponds to the same 32-bit external interrupt lines. Writing a 0 followed by a 1 to the targeted bit generates an interrupt if the corresponding ILR is set to edge-sensitive; otherwise, no interrupt is generated.

An external interrupt request and an internal software request are merged before being sent to the interrupt handler to be serviced. The software interrupt register is always read as 0. You can use this software interrupt mechanism to simulate an external interrupt and test the corresponding interrupt driver as long as the interrupt line is programmed as edge-sensitive.

All internal interrupts are brought to the OMAP 3.2 subchip level to provide maximum flexibility for system integration. You can reorganize, regroup, add, or delete the interrupt inputs for your applications.

Figure 4 shows the MPU interrupt handler.

Figure 4. MPU Interrupt Handler


Interrupt set register (SIR)
32 external interrupts
OR
Edge detection FLIP_FLOPS
Level or edge detected
Interrupt input register (ITR)
itmr
Mask interrupt register (MIR)
Interrupt level register 0 (ILR0)
Interrupt level register 1 (ILR1)
Interrupt level register 31 (ILR31)
Process next pending IRQ
Process next pending FIQ
Generate IRQ
Generate FIQ
SIR_IRQ
Interrupt encoded source register
SIR_FIQ
Interrupt encoded source register
Control register (CONTROL_REG)
IRQ
FIQ
To MPU
TIPB
itmr: ITR gated with MIR, to generate active interrupts

3.1.3 Interrupt Sequence

Table 21 shows the MPU interrupt sequence for an IRQ interrupt only. The FIQ interrupt sequence is identical.

Table 21. MPU Interrupt Sequence

Step	Interrupt Handler Action	MPU Action
One or several interrupts occur that set the corresponding bits in ITR	If one active interrupt occurs and the IRQ is not already active, the interrupt handler sends an IRQ. If several active interrupts occur, the interrupt handler must locate the interrupt with the highest priority. If an IRQ is not already active, the interrupt handler sends an IRQ.	
Processing the interrupt	When the IRQ is sent, SIR_IRQ is updated and the priority resolver is reset.	The MPU must read SIR_IRQ to determine the interrupt line being serviced. Then the MPU runs the corresponding subroutine.
Finish the interrupt		1. The MPU must first clear the interrupt bit in ITR (by writing a 0 in the corresponding bit or by reading SIR_IRQ. 2. For a level-sensitive interrupt, the level must be removed from the source that keeps the interrupt request low (active) for the next interrupt to occur. 3. Sets CONTROL_REG.NEW_IRQ_AGR to reset IRQ output and SIR_IRQ, thus allowing a new IRQ generation.

3.1.4 Interrupt Handler Software

To process edge-triggered and level-sensitive interrupts correctly, the following sequences must occur in the system. Only the IRQ treatment is described here. The FIQ treatment is exactly identical.

3.1.5 Edge-Triggered Interrupts

1) The interrupt handler module receives incoming one or more interrupts from outside OMAP and registers it in the interrupt register (ITR).

2) If there are several active incoming interrupts, the interrupt handler determines the highest priority interrupt and puts it in the N_IRQ register, which is invisible to the software programmer.

3) If the IRQ (interrupt from interrupt handler to the MPU) is not active, the interrupt handler sends the interrupt in N_IRQ register to the MPU as an IRQ signal. Then the source IRQ register (SIR_IRQ) is updated with contents of the N_IRQ register (the SIR_IRQ contains encoded information that conveys the interrupt line number of the IRQ).
4) The MPU recognizes the interrupt and jumps to the interrupt service routine (ISR) code.
5) Within the ISR code, the MPU reads the SIR_IRQ in the interrupt handler to determine which interrupt line caused the interrupt. The MPU executes specific code appropriately.
6) When MPU reads the SIR_IRQ, the corresponding bit is reset in ITR of interrupt handler module. The IRQ is still active.
7) The MPU, when it is about to exit the ISR routine, writes a 1 to CONTROL_REG.NEW_IRQ_AGR to deassert the IRQ going to MPU and to enable a new IRQ generation.
8) The MPU exits the ISR and continues its normal code execution.
9) When CONTROL_REG.NEW_IRQ_AGR is written, the process jumps to Step 2.

3.1.6 Level-Sensitive Interrupts

1) The interrupt handler module receives one or more incoming interrupts from outside OMAP. Level-sensitive interrupts are not registered, but are used in the logic as is. The interrupt handler assumes that the peripheral will not deassert the level-sensitive incoming interrupts until it is told to do so by the MPU.
2) The interrupt handler determines the highest priority interrupt and puts it in the N_IRQ register.
3) If the IRQ (interrupt from interrupt handler to the MPU) is not active, the interrupt handler sends the highest priority interrupt (interrupt in N_IRQ register) to the MPU as IRQ signal. Then the SIR_IRQ is updated with contents of N_IRQ register (the SIR_IRQ register contains encoded information that conveys the interrupt line number of IRQ). If the IRQ is active, which means CONTROL_REG.NEW_IRQ_AGR has not been set by MPU, the incoming IRQ has to wait until IRQ is not active.
4) The MPU recognizes the interrupt and jumps to the ISR code.
5) Within the ISR code, the MPU reads the SIR_IRQ in the interrupt handler to determine which interrupt line caused the interrupt.

6) The ISR code must be capable of doing one of the following things:
   - Letting the peripheral know that the interrupt generated by it has been serviced so the peripheral can deassert the interrupt request
   - Writing to interrupt handler mask interrupt register (MIR) to mask the level-sensitive interrupt

     Here the peripheral has to deassert the interrupt before the mask to the interrupt can be removed, so that the next interrupt can be recognized.

     If the peripheral deasserts the interrupt before the code in ISR tells it to, then the behavior is unpredictable and the interrupt may be lost.
7) The MPU must write a 1 to CONTROL_REG.NEW_IRQ_AGR when it is about to exit the ISR routine to deassert the IRQ going to MPU and to enable a new IRQ generation.
8) The MPU exits the ISR and continues its normal code execution.
9) When CONTROL_REG.NEW_IRQ_AGR is written into by MPU, the process jumps to Step 2.

3.1.7 Registers

Table 22 lists the registers available to handle interrupts. Table 23 through Table 30 provide register bit descriptions. All these registers are 32 bits wide and are controlled directly by the private TIBP bus. To determine the base address of these registers, see the Applications Processor Data Manual (SPRS231).

Table 22. Interrupt Registers

Name	Description	R/W	Offset
ITR	Interrupt register	R/W	0x00
MIR	Interrupt mask register	R/W	0x04
SIR_IRQ	Interrupt encoded source register for IRQ	R	0x10
SIR_FIQ	Interrupt encoded source register for FIQ	R	0x14
CONTROL_REG	Interrupt control register	R/W	0x18
ILRx	Interrupt level register	R/W	0x1C + 0x4 * x
SIR	Software interrupt set register	R/W	0x9C
GMR	Global mask interrupt register	R/W	0xA0

Table 23. Interrupt Register (ITR)

Offset: 0x00				
Bit	**Name**	**Function**	**R/W**	**Reset**
31:0	ACT_IRQ	Sets corresponding bit in ITR for edge-sensitive and level-sensitive interrupts.	R/W	0x0000 0000

When the MPU accesses SIR_IRQ or SIR_FIQ, the ITR bit corresponding to the interrupt that has requested MPU action is reset. The MPU can also clear each bit individually by writing a 0 to the corresponding bits at the ITR address. Writing a 1 to a bit keeps its previous value. You can use the individual clearing just before the MPU unmasks some interrupts and thus ignore some interrupt occurrences.

Table 24. Mask Interrupt Register (MIR)

Offset: 0x04				
Bit	**Name**	**Function**	**R/W**	**Reset**
31:0	IRQ_MSK	Masks each incoming interrupt individually	R/W	0xFFFF FFFF

You can mask each incoming interrupt individually with this register by setting the corresponding bit to 1. This register operates after the interrupt register, which means that occurrences of incoming interrupts are always stored in the interrupt register.

Table 25. Interrupt Encoded Source Register for IRQ (SIR_IRQ)

Offset: 0x10				
Bit	**Name**	**Function**	**R/W**	**Reset**
31:5	Reserved			
4:0	IRQ_NUM	Indicates encoded interrupt number that has an IRQ request. Reading this register clears the corresponding bit in the ITR register if the interrupt is set as edge-sensitive.	R	00000

Table 26. Interrupt Encoded Source Register for FIQ (SIR_FIQ)

Offset: 0x14				
Bit	**Name**	**Function**	**R/W**	**Reset**
31:5	Reserved			
4:0	FIQ_NUM	Indicates the encoded interrupt number that has an FIQ request. Reading this register clears the corresponding bit in the ITR register if the interrupt is set as edge-sensitive.	R	00000

Table 27. Interrupt Control Register (CONTROL_REG)

Offset: 0x18				
Bit	**Name**	**Function**	**R/W**	**Reset**
31:2	Reserved			
1	NEW_FIQ_AGR	New FIQ agreement. Writing a 1 resets the FIQ output, clears the SIR_FIQ, and enables a new FIQ generation. The corresponding bit of the ITR must be cleared first. Writing 0 has no effect.	R/W	0
0	NEW_IRQ_AGR	New IRQ agreement. Writing a 1 resets the IRQ output, clears the SIR_IRQ, and enables a new IRQ generation. The corresponding bit of the ITR must be cleared first. Writing 0 has no effect.	R/W	0

Table 28. Interrupt Level Register for Interrupt Number x (0 to 31) (ILRx)

Offset: 0x1C + 0x4 * Interrupt number				
Bit	**Name**	**Function**	**R/W**	**Reset**
31:7	Reserved			
6:2	PRIORITY	Defines the priority level when the corresponding interrupt is routed to IRQ or FIQ. 0 is the highest priority level. 31 is the lowest priority level.	R/W	00000
1	SENS_LEVEL	0: The corresponding interrupt is falling-edge sensitive. 1: The corresponding interrupt is low-level sensitive.	R/W	0
0	FIQ	0: The corresponding interrupt is routed to IRQ. 1: The corresponding interrupt is routed to FIQ.	R/W	0

Table 29. Software Interrupt Set Register (SIR)

Offset: 0x9C				
Bit	**Name**	**Function**	**R/W**	**Reset**
31:0	SIR	See below.	R/W	0x0000 0000

The user can program the SIR register to emulate the interrupt generation. The corresponding ILR must be programmed as edge-sensitive while using SIR register. The procedure to generate an edge-sensitive interrupt is: SIR(bit i) = 0 – SIR(bit i) = 1 (rising edge generated). SIR will not be cleared automatically (user must program it). A zero is always returned from a read to this register. External interrupts are merged with the software interrupts before they are sent to the MIR mask register for interrupt masking.

Table 30. Global Mask Interrupt Register (GMR)

Offset: 0xA0				
Bit	**Name**	**Function**	**R/W**	**Reset**
31:1	Reserved			
0	GLOBAL_MASK	When 1, the interrupt handler module and the output signal INT_DIS are set to 1. (INT_DIS is the output signal that indicates the interrupt handler has been disabled.) For power management, CLK&RST can turn off the interrupt handler functional clock. A handshaking protocol is defined for the CLK&RST module to idle or wake up the interrupt handler.	R/W	0

4 DSP Level 1 and Level 2.0 Interrupt Handler and Interface

4.1 Description

DSP interrupts are handled through two cascaded interrupt controllers:

- The DSP interrupt interface (level 1 handler) handles 24 interrupts.
- The DSP interrupt handler (level 2.0 handler) handles 16 level 2 interrupts that are routed to the DSP interrupt interface through low-priority interrupt request (IRQ) and fast-priority interrupt request (FIQ).

4.1.1 Interrupt Control and Configuration

If an interrupt occurs, the DSP_ITR register stores the incoming interrupt in the corresponding bit. When there are several incoming interrupts, the DSP interrupt handler compares the priority level of the interrupts before sending an IRQ or FIQ to the DSP core. The selected interrupt number is stored in DSP_SIR_IRQ or DSP_SIR_FIQ for the DSP to determine which interrupt service routine to execute. Reading either of these registers by the DSP resets the corresponding bit in DSP_ITR. The DSP can also individually clear each bit in DSP_ITR by writing a 0 to the corresponding bits. Writing a 1 keeps its previous value.

Each incoming interrupt can be masked individually by setting the corresponding bit in DSP_MIR to 1.

One interrupt level register (DSP_ILR) is associated with each incoming interrupt. The DSP_ILR sets whether the interrupt is to be edge-triggered or level-sensitive and assigns it a priority level: 0 (the highest priority level), 1, ... 14, 15 (the lowest priority level). If several interrupts have the same priority level assigned, they are serviced in a predefined order: IRQ_15, IRQ_14, ..., IRQ_1, IRQ_0. The DSP_ITR also allows routing of each of the 16 interrupts to either of the two DSP core input interrupts: FIQ or IRQ.

The IRQ or FIQ outputs can be reset by writing a 1 to the corresponding bit of DSP_CONTROL_REG to enable new IRQ or FIQ generation. The writing also clears DSP_SIR_IRQ or DSP_SIR_FIQ. The corresponding bit in DSP_ITR must be cleared before writing to the DSP_CONTROL_REG.

4.1.2 Software Interrupt

The interrupt handler also provides a 16-bit software interrupt register (DSP_SISR), which corresponds to the same 16-bit external interrupt lines. Writing a 0 followed by a 1 to the targeted bit generates an interrupt if the corresponding DSP_ILR is set to edge-sensitive; otherwise, no interrupt is generated.

An external interrupt request and an internal software request are merged before being sent to the interrupt handler to be serviced. The software interrupt register is always read back with a 0. You can use this software interrupt mechanism to simulate an external interrupt and test the corresponding interrupt driver as long as the interrupt line is programmed as edge-sensitive.

4.1.3 Latency

The DSP interrupt handler resides on the 16-bit DSP private TI peripheral bus (TIPB) and runs at half the frequency of the DSP clock. The latency from an incoming interrupt to FIQ/IRQ generation depends on the number of interrupts arriving at the same time. If there is only one, the latency is 5 clock cycles. If more than one interrupt become active at the same time and are routed to the same FIQ/IRQ, the latency can reach 3 + N*2 cycles, where N is the number of incoming interrupts.

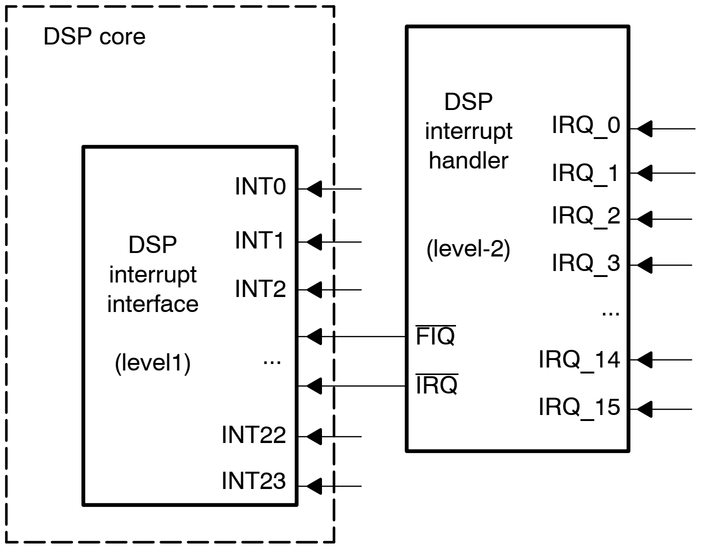
All interrupts for the DSP subsystem and OMAP subsystem are brought to the OMAP subchip boundary to provide maximum flexibility for system integration.

4.1.4 Interrupt Interface

The DSP interrupt interface augments DSP interrupt handler capability by enabling you to define edge-triggered or level-sensitive implementations for each of external interrupt lines. The DSP interrupt interface allows you to program the edge- or level-sensitivity of the two level 1 interrupts where IRQ and FIQ are routed.

Figure 14−1 shows an example of interrupt handling.

Figure 5. An Example of DSP Interrupt Handling



4.1.5 Interrupt Sequence

Table 31 shows the DSP interrupt sequence for an IRQ interrupt only. The FIQ interrupt sequence is identical.

Table 31. DSP Interrupt Sequence

Step	Interrupt Handler Action	DSP Action
One or several interrupts occur that set the corresponding bits in DSP_ITR register	If one active interrupt occurs and the IRQ is not already active, the interrupt handler sends an IRQ. If several active interrupts occur, the interrupt handler must locate the interrupt with the highest priority. If an IRQ is not already active, the interrupt handler sends an IRQ.	
Processing interrupt	When the IRQ is sent, the DSP_SIR_IRQ is updated and the priority resolver is reset.	The DSP must read the DSP_SIR_IRQ to determine the interrupt line being serviced. Then the DSP runs the corresponding subroutine.
Finish the interrupt		The DSP must first clear the interrupt bit in the DSP_ITR (by writing a 0 in the corresponding bit or by reading the DSP_SIR_IRQ). For a level-sensitive interrupt, the level must be removed for the next interrupt to occur. Sets the NEW_IRQ_AGR of the DSP_CONTROL_REG to reset IRQ output and the DSP_SIR_IRQ, thus allowing a new IRQ generation.

4.1.6 Interrupt Handler Software

To process edge-triggered and level-sensitive interrupts correctly, the following sequences must occur in the system. Here the sequence is described only for the IRQ interrupt; the FIQ sequence is exactly identical.

4.1.7 Edge-Triggered Interrupts

1) The DSP interrupt handler module receives one or more incoming interrupts from outside OMAP and registers it in the DSP interrupt register (DSP_ITR).

2) The DSP interrupt handler determines the highest priority interrupt and puts it in the N_IRQ register, which is not seen by software.

3) If the IRQ (interrupt from DSP interrupt handler to the DSP) is not active, the DSP interrupt handler sends the highest priority interrupt (interrupt in N_IRQ register) to the DSP as an IRQ signal. Then the DSP_SIR_IRQ register is updated with contents of the N_IRQ register (the DSP_SIR_IRQ contains encoded information that conveys the interrupt line number of the IRQ).

4) The DSP recognizes the interrupt and jumps to the interrupt service routine (ISR) code.

5) Within the ISR code, the DSP reads the DSP_SIR_IRQ in the DSP interrupt handler to determine which interrupt line caused the interrupt. The DSP executes specific code appropriately.

6) When the DSP reads the DSP_SIR_IRQ, the corresponding bit is reset in the DSP_ITR of the DSP interrupt handler module. The IRQ is still active.

7) The DSP writes a 1 to new IRQ agreement bit (NEW_IRQ_AGR) in the DSP_CONTROL_REG when it is about to exit the ISR routine to deassert the IRQ going to the DSP and to enable a new IRQ generation.

8) The DSP exits the ISR and continues its normal code execution.

9) When the NEW_IRQ_AGR bit is written, the process jumps to Step 2.

4.1.8 Level-Sensitive Interrupts

1) The DSP interrupt handler module receives one or more incoming interrupts from outside OMAP. Level-sensitive interrupts are not registered, but are used in the logic as is. The DSP interrupt handler assumes that the peripheral will not deassert the level-sensitive incoming interrupts until it is told to do so by the DSP.

2) The DSP interrupt handler determines the highest priority interrupt and puts it in the N_IRQ register.

3) If the IRQ (interrupt from DSP interrupt handler to the DSP) is not active, the DSP interrupt handler sends the highest priority interrupt (interrupt in N_IRQ register) to the DSP as IRQ signal. Then the DSP_SIR_IRQ is updated with contents of N_IRQ register (the DSP_SIR_IRQ register contains encoded information that conveys the interrupt line number of IRQ).

4) The DSP recognizes the interrupt and jumps to the ISR code.

5) Within the ISR code, the DSP reads the DSP_SIR_IRQ in the DSP interrupt handler to determine which interrupt line caused the interrupt.

6) The ISR code must be capable of doing one of the following things:

- Letting the peripheral know that the interrupt generated by it has been serviced so the peripheral can deassert the interrupt request

- Writing to the DSP mask interrupt register (DSP_MIR) to mask the level-sensitive interrupt

  Here the peripheral must deassert the interrupt before the mask to the interrupt can be removed, so that the next interrupt can be recognized.

  If the peripheral deasserts the interrupt before the code in ISR tells it to do so, then the behavior is unpredictable and the Interrupt may be lost.

7) The DSP, when it is about to exit the ISR routine, must write a 1 to the NEW_IRQ_AGR in the DSP_CONTROL_REG to deassert the IRQ going to DSP and to enable a new IRQ generation.

8) The DSP exits the ISR and continues its normal code execution.

9) When the NEW_IRQ_AGR in the DSP_CONTROL_REG is written into by DSP, the process jumps to Step 2.

4.1.9 Registers

This section describes the DSP interrupt interface and DSP interrupt handler registers for the OMAP 3.2 hardware engine. DSP software configures these registers through the DSP private TIPB. To determine the base addresses for these registers, see the Multimedia Processor Interrupts Reference Guide (literature number SPRU757).

4.1.10 DSP Interrupt Interface

Table 32 lists the DSP interface registers. Table 33 and Table 34 provide register bit descriptions. All these registers are 16 bits wide and are controlled directly by the DSP private TIPB.

Table 32. DSP Interrupt Interface Registers

Name	Description	R/W	Offset
EDGE_EN_HI	Incoming interrupt high register	R/W	0x00
EDGE_EN_LO	Incoming interrupt low register	R/W	0x02

Table 33. Incoming Interrupt High Register (EDGE_EN_HI)

Offset: 0x00				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:8	Reserved			
7	HOST_INTERRUPT	Defines whether the host interrupt from DSP to MPU is edge-triggered or level-sensitive: 0: HOST_INTERRUPT is level-sensitive. 1: HOST_INTERRUPT is edge-sensitive.	R/W	0
6	NMI	Defines whether the nonmaskable interrupt is edge or level sensitive: 0: NMI is level-sensitive. 1: NMI is edge-sensitive.	R/W	0
5:0	CHx	Defines channel CHx as edge-triggered or level-sensitive, where CHx corresponds to interrupt channels: 0: CHx is level-sensitive. 1: CHx is edge-sensitive.	R/W	000000

Table 34. Incoming Interrupt Low Register (EDGE_EN_LO)

Offset: 0x02				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:0	CHx	This bit defines channel CHx as edge-triggered or level-sensitive, where CHx corresponds to interrupt channels: 0: CHx is level-sensitive. 1: CHx is edge-sensitive.	R/W	0x0000

4.1.11 DSP Interrupt Handler

Table 22 lists the DSP registers available to handle interrupts. Table 36 through Table 42 describe the register bits. All these registers are 16 bits wide and are controlled directly by the DSP private TIPB.

Table 35. DSP Interrupt Registers

Name	Description	R/W	Offset
DSP_ITR	DSP interrupt	R/W	0x00
DSP_MIR	DSP mask interrupt	R/W	0x02
DSP_SIR_IRQ	Interrupt encoded source for IRQ	R	0x04
DSP_SIR_FIQ	Interrupt encoded source for FIQ	R	0x06
DSP_CONTROL_REG	DSP interrupt control	R/W	0x08
DSP_SISR	DSP software interrupt set	R/W	0x0A
DSP_ILRx	DSP interrupt level for interrupt number x	R/W	0x0C + 0x2 * x

Table 36. Interrupt Register (DSP_ITR)

Offset: 0x00				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:0	ACT_IRQ	If edge-sensitive interrupt occurs, it stores the active line. The DSP can individually clear each bit by writing a 0 to the corresponding bit. Writing a 1 keeps its previous value.	R/W	0x0000

Table 37. Mask Interrupt Register (DSP_MIR)

Offset: 0x02				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:0	IRQ_MSK	Writing a 1 masks the corresponding interrupt. Each bit corresponds to one interrupt.	R/W	0xFFFF

Table 38. Interrupt Encoded Source Register for IRQ (DSP_SIR_IRQ)

Offset: 0x04				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:4	Reserved			
3:0	IRQ_NUM	Indicates the encoded interrupt number that has an IRQ request. Reading this register clears the corresponding bit in the DSP_ITR if the interrupt is set as edge-sensitive.	R	0000

Table 39. Interrupt Encoded Source Register for FIQ (DSP_SIR_FIQ)

Offset: 0x06				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:4	Reserved			
3:0	FIQ_NUM	Indicates the encoded interrupt number that has an FIQ request. Reading this register clears the corresponding bit in the DSP_ITR if the interrupt is set as edge-sensitive.	R	0000

Table 40. Interrupt Control Register (DSP_CONTROL_REG)

Offset: 0x08				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:2	Reserved			
1	NEW_FIQ_AGR	Writing a 1 resets the FIQ output, clears the source FIQ register, and enables a new FIQ generation, reset by internal logic. The corresponding bit of DSP_ITR must be cleared first.	R/W	0
0	NEW_IRQ_AGR	Writing a 1 resets an IRQ output, clears the source IRQ register, and enables a new IRQ generation, reset by internal logic. The corresponding bit of DSP_ITR must be cleared first.	R/W	0

Table 41. Software Interrupt Set Register (DSP_SISR)

Offset: 0x0A				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:0	DSP_SISR	Writing a 0 followed by a 1 to any bit generates an interrupt to the DSP if the corresponding DSP_ILR register is set as edge-triggered; otherwise, no interrupt is generated.	R/W	0x0000

Table 42. Interrupt Level Register for Interrupt Number x [0–15] (DSP_ILRx)

Offset: 0x0C				
Bit	**Name**	**Function**	**R/W**	**Reset**
15:6	Reserved			
5:2	PRIORITY	Defines the priority level when the corresponding interrupt is routed to IRQ or FIQ. 0 is the highest priority level. 15 is the lowest priority level.	R/W	0000
1	SENS_LEVEL	0: The corresponding interrupt is rising-edge sensitive. 1: The corresponding interrupt is high-level sensitive.	R/W	0
0	FIQ	0: The corresponding interrupt is routed to IRQ. 1: The corresponding interrupt is routed to FIQ.	R/W	0

Index