Features

- **Compatible with an Embedded ARM7TDMI™ Processor**
- **8-level Priority**
- **From 2 to 32 Interrupt Sources**
- **Individually Maskable and Vectored**
- **Substantially Reduces the Software and Real-time Overhead in Handling Internal and External Interrupts**
- **Fast Forcing Feature Available**
- **Includes Protect Mode Feature Via a Configuration Register**
- **Can be Directly Connected to the Atmel Implementation of the AMBA™ Peripheral Bus (APB)**
- **Fully Scan Testable (up to 99%)**

32-bit Embedded ASIC Core Peripheral

Advanced Interrupt Controller 2 (AIC2)

Description

The Advanced Interrupt Controller 2 (AIC2) is an 8-level priority, individually maskable, vectored interrupt controller. It can substantially reduce the software and real-time overhead in handling internal and external interrupts.

This peripheral can be used with any 32-bit microcontroller core if the timing diagram shown on page 4 is respected. The interrupt handling must be performed as described on page 7, using the corresponding assembler instructions for the 32-bit core used.

When using an ARM7TDMI® as the core, the Atmel bridge must be used to provide the correct bus interface to the AIC2.

The interrupt controller can be directly connected to the nFIQ (fast interrupt request) and the nIRQ (standard interrupt request) inputs of an ARM7TDMI processor, or the equivalent inputs of another 32-bit microcontroller core. The fast forcing feature available in the AIC2 allows redirection of any internal or external interrupt source to provide a fast interrupt rather than a normal interrupt. The processor's nFIQ line can be asserted either by the external fast interrupt request input FIQ, or by the other interrupt sources via the fast forcing feature. The nIRQ line is asserted by all other interrupt sources except those that assert FIQ.

An 8-level priority encoder allows the customer to define the priority between the different nIRQ interrupt sources. Internal interrupt sources are programmed to be level sensitive or edge triggered. External interrupt sources can be programmed to be positive-edge or negative-edge triggered or high-level or low-level sensitive.

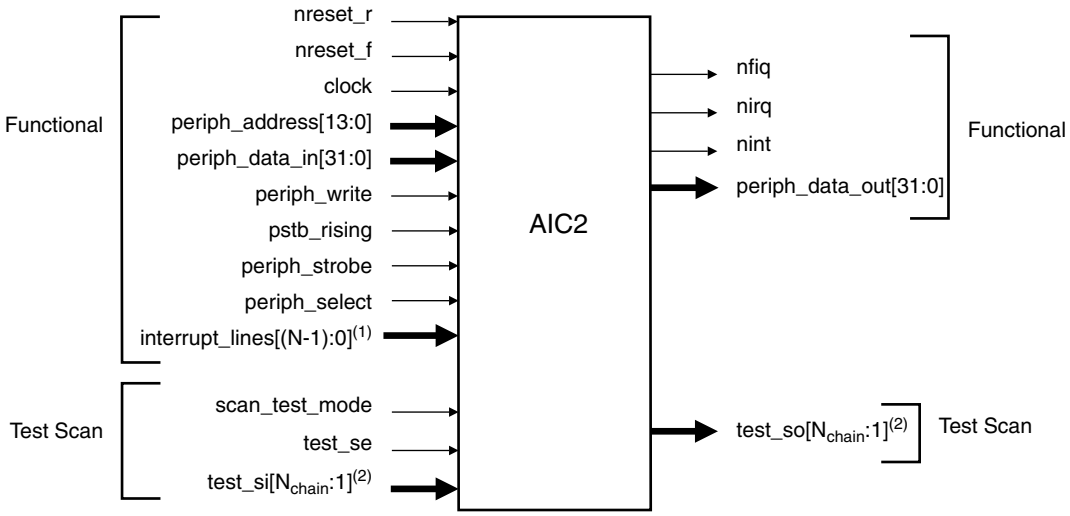
This document describes the AIC2 in a specific configuration as specified below:

- One FIQ is connected on the interrupt line[0].
- Eight internal interrupt sources are connected on interrupt lines[8:1].
- Three external interrupt sources are connected on interrupt lines[11:9] but are mapped on interrupts[18:16].

The interrupt sources are listed in Table 2 and the AIC2 programmable registers in Table 3. The register interaction is shown in Figure 5.

Rev. 1796B–CASIC–03/02

Figure 1. AIC2 Pin Configuration



Notes: 1. N = Number of interrupt sources
2. N_{chain} = Number of scan chains

Table 1. Pin Description List

Name	Definition	Type	Active Level	Comments
Functional				
nreset_r	System Reset	Input	Low	Resets all counters and signals clocked on rising edge of clock
nreset_f	System Reset	Input	Low	Resets all counters and signals clocked on falling edge of clock
clock	System Clock	Input	–	–
periph_address[13:0]	Peripheral Address Bus	Input	–	From host (bridge). Software user interface address bus
periph_data_in[31:0]	Peripheral Data Bus Input	Input	–	From host (bridge). Software user interface data bus
periph_data_out[31:0]	Peripheral Data Bus Input	Output	–	To host (bridge). Software user interface data bus
periph_write	Peripheral Write Enable	Input	–	From host (bridge). Transfer enable. When high, indicates that the host processor is writing to a register or executing a command.
pstb_rising	–	Input	–	From host (bridge). Clock for all DFFs controlling configuration registers
periph_strobe	Peripheral Strobe	Input	High	When high, indicates that data and address buses are stable
periph_select	Peripheral Selection	Input	High	When high, in03/02dicates that the host is accessing the AIC2

Table 1. Pin Description List (Continued)

Name	Definition	Type	Active Level	Comments
interrupt_lines[(N-1):0](1)	Interrupt Input Lines (Sources)	Input	–	Polarities can be defined in the configuration registers for inputs corresponding to external interrupt sources. Generally, they are connected as follows: - FIQ source on interrupt_lines[0] - N_{int} internal sources on interrupt_lines[N_{int}:1](2) - N_{ext} external sources on interrupt_lines[(N-1):(N-N_{ext})](1, 3) Note: N = N_{ext} + N_{int} + 1(FIQ)
nfiq	Fast Interrupt Request	Output	Low	Destination: Input nFIQ of ARM core Source: Fast Interrupt Request; usually connected to interrupt_iines[0]
nirq	Interrupt Request	Output	Low	Destination: Input nIRQ of ARM core Source: All interrupt requests (except Fast Interrupt Request) usually connected to interrupt_lines[(N-1):1](1)
nint	–	Output	Low	Low if either nfiq or nirq is active. If nfiq and nirq are disabled by bit DEBUG_MASK in register AIC2_DEBUG, nint generates an interrupt depending on the interrupt sources
Test Scan				
scan_test_mode	Scan Test Mode	Input	–	Must be tied to 1 during scan test. Must be tied to 0 in functional mode.
test_se	Test Scan Shift Enable	Input	–	Scan shift enabled when tied to 1
test_si[N_{chain}:1](4)	Test Scan Input	Input	–	Input to scan chains
test_so[N_{chain}:1](4)	Test Scan Output	Output	–	Output from scan chains

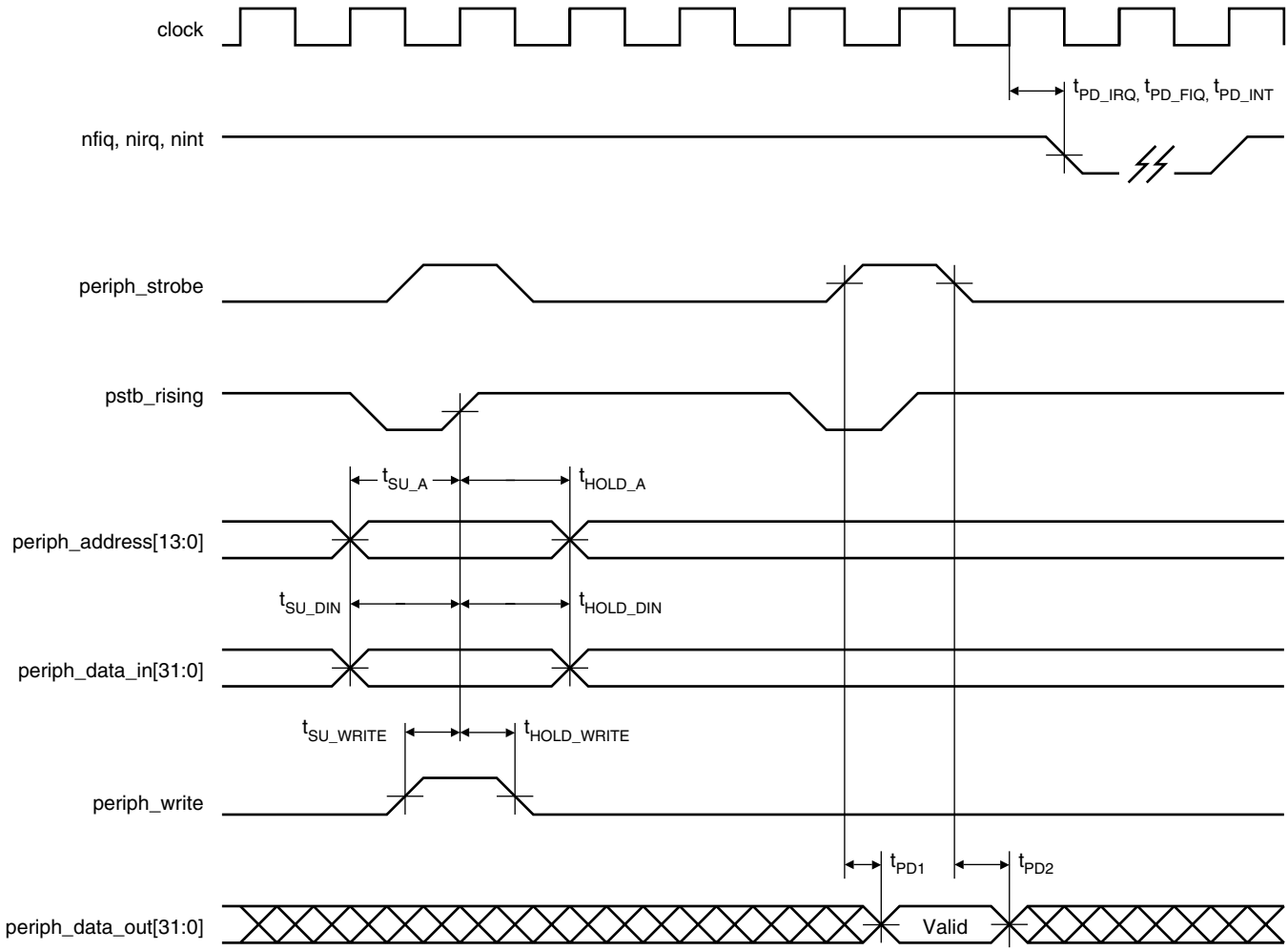
Notes: 1. N = Number of interrupt sources
2. N_{int} = Number of internal interrupt sources
3. N_{ext} = Number of external interrupt sources
4. N_{chain} = Number of scan chains

Scan Test Configuration

The fault coverage is maximum if all non-scan inputs can be controlled and all non-scan outputs can be observed. In order to achieve this, the ATPG vectors must be generated on the entire circuit (top-level) which includes the AIC2 or all AIC2 I/Os must have a top level access and ATPG vectors must be applied to these pins.

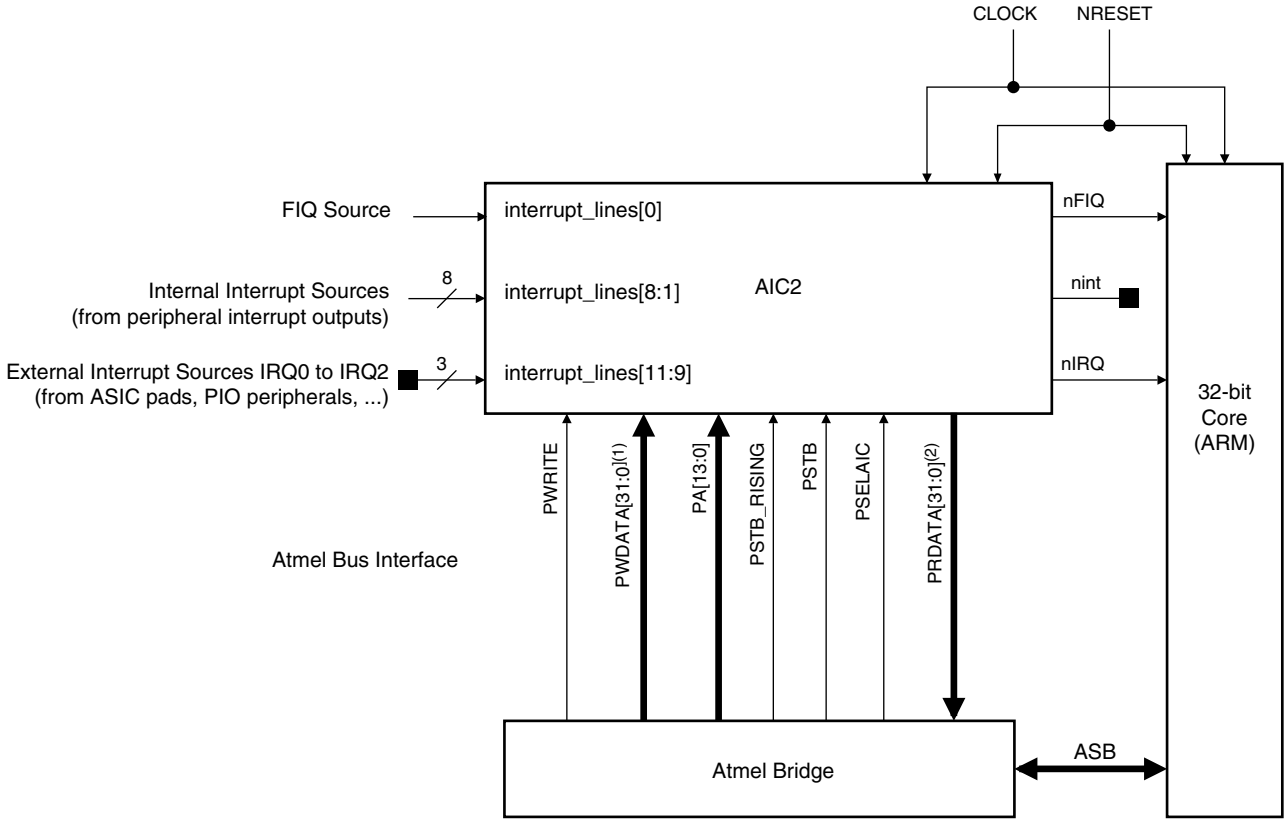
Timing Diagram

Figure 2. AIC2 Timing Diagram



Functional Description

Figure 3. Connecting the AIC2 to an ARM®-based Microcontroller



Notes: 1. data_to_periph = signal name from Atmel Bridge, periph_data_in[31:0] = signal name from AIC2 peripheral, PWDATA[31:0] = signal name from Atmel AMBA System Architecture specification.
2. data_from_periph = signal name from Atmel Bridge, periph_data_out[31:0] = signal name from AIC2 peripheral, PRDATA[31:0] = signal name from Atmel AMBA System Architecture specification.

Figure 4. Interrupt Controller Block Diagram

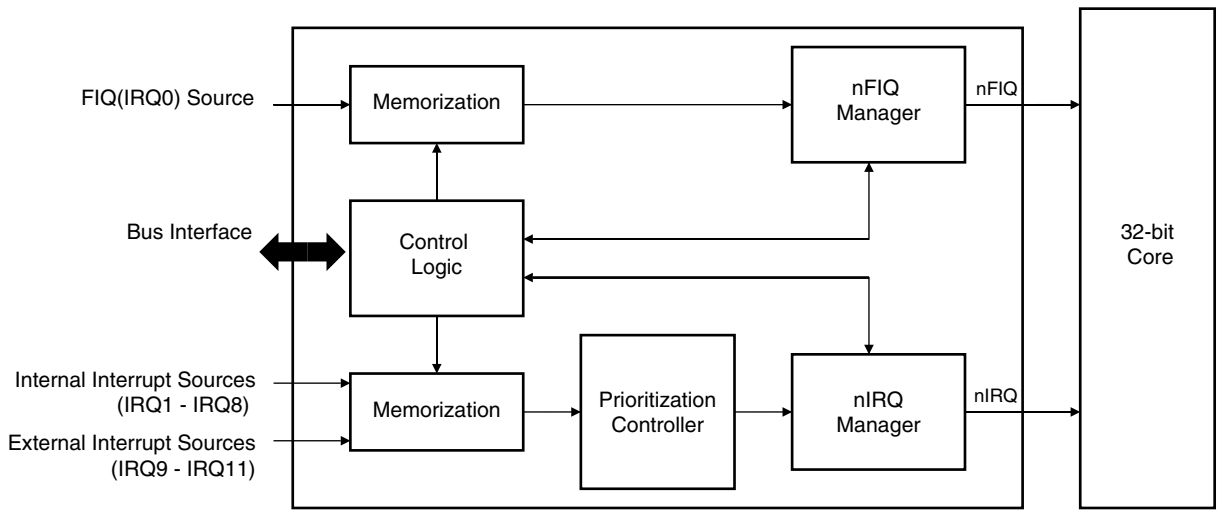


Table 2. AIC2 Interrupt Sources

Interrupt Source	Interrupt Name	Interrupt Description[1]	Comment	AT91x40 Series Interrupt Assignment[1]
0	FIQ	Fast Interrupt	Edge/Level Negative/Positive	Fast Interrupt
1	IRQ1	Interrupt 1	Edge/Level Positive Only	Software Interrupt
2	IRQ2	Interrupt 2		USART Channel 0 Interrupt
3	IRQ3	Interrupt 3		USART Channel 1 Interrupt
4	IRQ4	Interrupt 4		Timer Channel 0 Interrupt
5	IRQ5	Interrupt 5		Timer Channel 1 Interrupt
6	IRQ6	Interrupt 6		Timer Channel 2 Interrupt
7	IRQ7	Interrupt 7		Watchdog Interrupt
8	IRQ8	Interrupt 8		Parallel I/O interrupt
9	–	Reserved	–	Reserved
10	–	Reserved	–	Reserved
11	–	Reserved	–	Reserved
12	–	Reserved	–	Reserved
13	–	Reserved	–	Reserved
14	–	Reserved	–	Reserved
15	–	Reserved	–	Reserved
16	IRQ9	Interrupt 9	Edge/Level Negative/Positive	External Interrupt 0
17	IRQ10	Interrupt 10		External Interrupt 1
18	IRQ11	Interrupt 11		External Interrupt 2
19	–	Reserved	–	Reserved
20	–	Reserved	–	Reserved
21	–	Reserved	–	Reserved
22	–	Reserved	–	Reserved
23	–	Reserved	–	Reserved
24	–	Reserved	–	Reserved
25	–	Reserved	–	Reserved
26	–	Reserved	–	Reserved
27	–	Reserved	–	Reserved
28	–	Reserved	–	Reserved
29	–	Reserved	–	Reserved
30	–	Reserved	–	Reserved
31	–	Reserved	–	Reserved

Note: 1. Reserved interrupt sources are not available. Corresponding registers must not be used and read 0.

Interrupt Handling

Hardware Interrupt Vectoring

The hardware interrupt vectoring reduces the number of instructions required to manage the interrupt handlers to only one. By storing the following instruction at address 0x00000018 (*if an ARM7TDMI is used as the host processor*), the processor loads the program counter with the interrupt handler address stored in the AIC2_IVR register. Execution is then vectored to the interrupt handler corresponding to the current interrupt.

```
LDR     PC,[PC,# -&F20]
```

The current interrupt is the interrupt with the highest priority when the Interrupt Vector Register (AIC2_IVR) is read. The value read in the AIC2_IVR corresponds to the address stored in the Source Vector Register (AIC2_SVR) of the current interrupt. Each interrupt source has its corresponding AIC2_SVR. In order to take advantage of the hardware interrupt vectoring, it is necessary to store the address of each interrupt handler in the corresponding AIC2_SVR, at system initialization.

Priority Controller

The nIRQ line is controlled by an 8-level priority encoder. Each source has a programmable priority level of 7 to 0. Level 7 is the highest priority and level 0 the lowest.

When the AIC2 receives more than one unmasked interrupt at a time, the interrupt with the highest priority is serviced first. If both interrupts have equal priority, the interrupt with the lowest interrupt source number (see Table 2) is serviced first.

The current priority level is defined as the priority level of the current interrupt at the time the register AIC2_IVR is read (the interrupt which will be serviced).

In the case when a higher priority unmasked interrupt occurs while an interrupt already exists, there are two possible outcomes depending on whether the AIC2_IVR has been read.

1. If the nIRQ line has been asserted but the AIC2_IVR has not been read, then the processor will read the new higher priority interrupt handler address in the AIC2_IVR register and the current interrupt level is updated.
2. If the processor has already read the AIC2_IVR then the nIRQ line is reasserted. When the processor has authorized nested interrupts to occur and reads the AIC2_IVR again, it reads the new, higher priority interrupt handler address. At the same time the current priority value is pushed onto a first-in last-out stack and the current priority is updated to the higher priority.

When the end of interrupt command register (AIC2_EOICR) is written the current interrupt level is updated with the last stored interrupt level from the stack (if any). Hence at the end of a higher priority interrupt, the AIC2 returns to the previous state corresponding to the preceding lower priority interrupt which had been interrupted.

Software Interrupt Handling

The interrupt handler must read the AIC2_IVR as soon as possible. This de-asserts the nIRQ request to the processor and clears the interrupt in case it is programmed to be edge triggered. This permits the AIC2 to assert the nIRQ line again when a higher priority unmasked interrupt occurs.

At the end of the interrupt service routine, the end of interrupt command register (AIC2_EOICR) must be written. This allows pending interrupts to be serviced.

Interrupt Masking

Each interrupt source, including FIQ, can be enabled or disabled using the command registers AIC2_IECR and AIC2_IDCR. The interrupt mask can be read in the read only register AIC2_IMR. A disabled interrupt does not affect the servicing of other interrupts.

Interrupt Clearing and Setting

All interrupt sources which are programmed to be edge triggered (including FIQ) can be individually set or cleared by respectively writing to the registers AIC2_ISCR and AIC2_ICCR. This function of the interrupt controller is available for auto-test or software debug purposes.

Fast Interrupt Request

The external FIQ line is the only source which can raise a fast interrupt request to the processor. Therefore, it has no priority controller. It can be programmed to be positive or negative edge triggered or high- or low-level sensitive in the related AIC2_SMR register (SMR[0]).

The fast interrupt handler address can be stored in the related AIC2_SVR register (SVR[0]). The value written into this register is available by reading the AIC2_FVR register when an FIQ interrupt is raised. By storing the following instruction at address 0x0000001C (*if an ARM7TDMI is used as the host processor*), the processor will load the program counter with the interrupt handler address stored in the AIC2_FVR register.

```
LDR     PC,[PC,# -&F20]
```

Alternatively, the interrupt handler can be stored starting from address 0x0000001C as described in the ARM7TDMI datasheet.

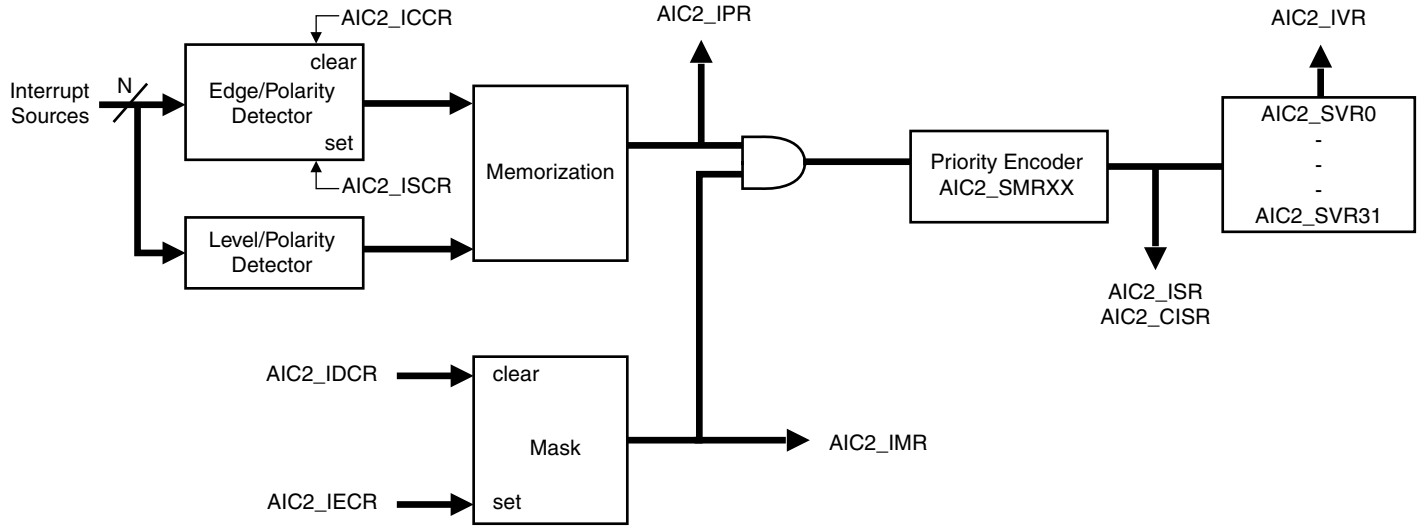
Software Interrupt

Any interrupt source of the AIC2 can be a software interrupt. It must be programmed to be edge triggered in order to set or clear it by writing to the AIC2_ISCR and AIC2_ICCR. This is totally independent of the SWI instruction of the ARM7TDMI processor.

Spurious Interrupt

A spurious interrupt is a signal of very short duration on one of the interrupt input lines. See “Spurious Interrupt Sequence” on page 23 for the sequence of actions necessary to handle this situation.

Figure 5. Register Interaction



Fast Forcing

The Fast Forcing feature is enabled or disabled by using the Fast Forcing Enable Register (AIC2_FFER) or the Fast Forcing Disable Register (AIC2_FFDR). Writing to these registers results in an update of the Fast Forcing Status Register (AIC2_FFSR) that controls the feature for each internal or external interrupt source. When the feature is disabled, the interrupt source is handled as is typical for the current operating mode.

When Fast Forcing is enabled, the edge/level programming and, in certain cases, edge detection of the interrupt source are both still active, but the source cannot trigger a normal interrupt to the core and is not seen by the priority handler.

If the interrupt source is programmed in level mode and an active level is sampled, the Fast Forcing results in the assertion of the nFIQ line to the core.

If the interrupt source is programmed in edge mode and an active edge is detected, the Fast Forcing results in the assertion of the nFIQ line to the core.

The Fast Forcing feature does not affect the Source 0 pending bit in the Interrupt Pending Register.

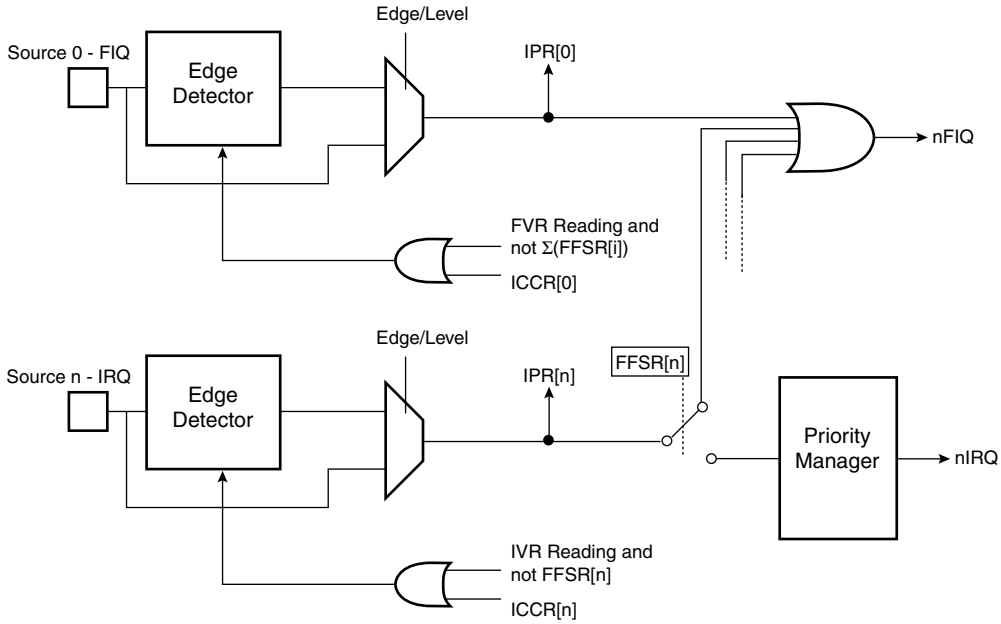
The Fast Interrupt Vector Register (AIC2_FVR) reads the contents of the Source Vector Register 0, whatever the source of the Fast Interrupt may be. The read of the FVR does not clear the Source 0 when the Fast Forcing feature is used; the interrupt is cleared by writing to the Interrupt Clear Command Register.

All enabled and pending interrupt sources that have the Fast Forcing feature enabled and that are programmed in edge mode must be cleared by writing to the Interrupt Clear Command Register. In doing so, they are cleared independently and thus lost interrupts are prevented.

The read of the IVR does not clear the source that has the Fast Forcing feature enabled.

The FIQ pin continues to operate normally and becomes one of the Fast Interrupt sources.

Figure 6. Fast Forcing Interaction



AIC2 User Interface

Table 3. AIC2 Memory Map

Address[2,3]	Register	Name	Access	Reset State
0x3000	Source Mode Register 0	AIC2_SMR0	Read/write	0
0x3004	Source Mode Register 1	AIC2_SMR1	Read/write	0
–	–	–	Read/write	0
0x307C	Source Mode Register 31	AIC2_SMR31	Read/write	0
0x3080	Source Vector Register 0	AIC2_SVR0	Read/write	0
0x3084	Source Vector Register 1	AIC2_SVR1	Read/write	0
–	–	–	Read/write	0
0x30FC	Source Vector Register 31	AIC2_SVR31	Read/write	0
0x3100	IRQ Vector Register	AIC2_IVR	Read-only	0
0x3104	FIQ Vector Register	AIC2_FVR	Read-only	0
0x3108	Interrupt Status Register	AIC2_ISR	Read-only	0
0x310C	Interrupt Pending Register	AIC2_IPR	Read-only	(see Note 1)
0x3110	Interrupt Mask Register	AIC2_IMR	Read-only	0
0x3114	Core Interrupt Status Register	AIC2_CISR	Read-only	0
0x3118	Reserved	–	–	–
0x311C	Reserved	–	–	–
0x3120	Interrupt Enable Command Register	AIC2_IECR	Write-only	–
0x3124	Interrupt Disable Command Register	AIC2_IDCR	Write-only	–
0x3128	Interrupt Clear Command Register	AIC2_ICCR	Write-only	–
0x312C	Interrupt Set Command Register	AIC2_ISCR	Write-only	–
0x3130	End of Interrupt Command Register	AIC2_EOICR	Write-only	–
0x3134	Spurious Interrupt Vector Register	AIC2_SPU	Read/write	0
0x3138	Debug Control Register (Protect)	AIC2_DEBUG	Read/write	0
0x3140	Fast Forcing Enable Register	AIC2_FFER	Write-only	–
0x3144	Fast Forcing Disable Register	AIC2_FFDR	Write-only	–
0x3148	Fast Forcing Status Register	AIC2_FFSR	Read-only	0

Notes:
1. The reset value of this register depends on the level of the external interrupt sources (IRQ9 - IRQ11). All other sources are cleared at reset.
2. The address takes into account the 2 LSBs [1:0], but the macrocell does not implement these bits.
3. To use relative offset from interrupt exception vector addresses (see “Standard Interrupt Sequence” on page 21), the base address for the AIC2 into an ARM7TDMI-based system must be 0xFFFFF000. As the 14 LSBs of the address system bus are used to address the AIC2 registers, the minimum offset is 0x3000.

In the following register descriptions, all undefined bits (“–”) read “0”.

If the user selects an address which is not defined in Table 3, the value of periph_data_out[31:0] is 0x00000000.

AIC2 Source Mode Register

Register Name: AIC2_SMR0..AIC2_SMR31

Access Type: Read/write

Reset Value: 0

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	SRCTYPE		–	–	PRIOR		

- **PRIOR: Priority Level**

Program the priority level for all sources except FIQ source (source 0).

The priority level can be between 0 (lowest) and 7 (highest).

The priority level is not used for the FIQ, in the related SMR register (SMR[0]).

- **SRCTYPE: Interrupt Source Type**

Program the input to be positive- or negative-edge triggered or positive- or negative-level sensitive.

The active level or edge is not programmable for the internal interrupt sources (IRQ1 - IRQ8).

SRCTYPE		Internal Interrupt Sources (IRQ1 - IRQ8)	External Interrupt Sources (IRQ9 - IRQ11)
0	0	High-level Sensitive	Low-level Sensitive
0	1	Positive-edge Triggered	Negative-edge Triggered
1	0	High-level Sensitive	High-level Sensitive
1	1	Positive-edge Triggered	Positive-edge Triggered

AIC2 Source Vector Register

Register Name: AIC2_SVR0..AIC2_SVR31

Access Type: Read/write

Reset Value: 0

31	30	29	28	27	26	25	24
VECTOR							
23	22	21	20	19	18	17	16
VECTOR							
15	14	13	12	11	10	9	8
VECTOR							
7	6	5	4	3	2	1	0
VECTOR							

- **VECTOR: Interrupt Handler Address**

The user may store in these registers the addresses of the corresponding handler for each interrupt source.

AIC2 Interrupt Vector Register

Register Name: AIC2_IVR

Access Type: Read-only

Reset Value: 0

31	30	29	28	27	26	25	24
IRQV							
23	22	21	20	19	18	17	16
IRQV							
15	14	13	12	11	10	9	8
IRQV							
7	6	5	4	3	2	1	0
IRQV							

- **IRQV: Interrupt Vector Register**

The IRQ Vector Register contains the vector programmed by the user in the Source Vector Register corresponding to the current interrupt.

The Source Vector Register (1 to 31 in the case of FIQ mapped on interrupt[0]) is indexed using the current interrupt number when the Interrupt Vector Register is read.

When there is no current interrupt, the IRQ Vector Register reads 0.

AIC2 FIQ Vector Register

Register Name: AIC2_FVR

Access Type: Read-only

Reset Value: 0

31	30	29	28	27	26	25	24
FIQV							
23	22	21	20	19	18	17	16
FIQV							
15	14	13	12	11	10	9	8
FIQV							
7	6	5	4	3	2	1	0
FIQV							

- **FIQV: FIQ Vector Register**

The FIQ Vector Register contains the vector programmed by the user in the Source Vector Register (0 in case of FIQ mapped on interrupt[0]) that corresponds to FIQ.

AIC2 Interrupt Status Register

Register Name: AIC2_ISR

Access Type: Read-only

Reset Value: 0

31	30	29	28	27	26	25	24
0	0	0	0	0	0	0	0
23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0
15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
0	0	0	IRQID				

- **IRQID: Current IRQ Identifier**

The Interrupt Status Register returns the current interrupt source number.

AIC2 Interrupt Pending Register

Register Name: AIC2_IPR

Access Type: Read-only

Reset Value: 0

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Interrupt Pending**

0 = Corresponding interrupt is inactive.

1 = Corresponding interrupt is pending.

AIC2 Interrupt Mask Register

Register Name: AIC2_IMR

Access Type: Read-only

Reset Value: 0

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Interrupt Mask**

0 = Corresponding interrupt is disabled.

1 = Corresponding interrupt is enabled.

AIC2 Core Interrupt Status Register

Register Name: AIC2_CISR

Access Type: Read-only

Reset Value: 0

31	30	29	28	27	26	25	24
0	0	0	0	0	0	0	0
23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0
15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
0	0	0	0	0	0	NIRQ	NFIQ

- **NFIQ: NFIQ Status**

0 = NFIQ line inactive.

1 = NFIQ line active.

- **NIRQ: NIRQ Status**

0 = NIRQ line inactive.

1 = NIRQ line active.

AIC2 Interrupt Enable Command Register

Register Name: AIC2_IECR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Interrupt Enable**

0 = No effect.

1 = Enables corresponding interrupt.

AIC2 Interrupt Disable Command Register

Register Name: AIC2_IDCR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Interrupt Disable**

0 = No effect.

1 = Disables corresponding interrupt.

AIC2 Interrupt Clear Command Register

Register Name: AIC2_ICCR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Interrupt Clear**

0 = No effect.

1 = Clears corresponding interrupt.

AIC2 Interrupt Set Command Register

Register Name: AIC2_ISCR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Interrupt Set**

0 = No effect.

1 = Sets corresponding interrupt.

AIC2 End of Interrupt Command Register

Register Name: AIC2_EOICR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

The End of Interrupt Command Register is used by the interrupt routine to indicate that the interrupt treatment is complete. Any value can be written because it is only necessary to make a write to this register location to signal the end of interrupt treatment.

AIC2 Spurious Interrupt Vector Register

Register Name: AIC2_SPU

Access Type: Read/write

Reset Value: 0

31	30	29	28	27	26	25	24
SIQV							
23	22	21	20	19	18	17	16
SIQV							
15	14	13	12	11	10	9	8
SIQV							
7	6	5	4	3	2	1	0
SIQV							

- **SIQV: Spurious Interrupt Vector Register** (see “Spurious Interrupt Sequence” on page 23 for more details)

This register contains the 32-bit address of an interrupt routine which is used to treat cases of spurious interrupts.

The programmed address is read in the AIC2_IVR if it is read when the nIRQ line is not asserted.

The programmed address is read in the AIC2_FVR if it is read when the nFIQ line is not asserted.

AIC2 Debug Control Register (Protect Control Register)

Register Name: AIC2_DEBUG

Access Type: Read/write

Reset Value: 0

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	–	–	–	–	DBG_MASK	DBG_MODE

- **DBG_MASK**

When high, nfiq and nirq outputs are disabled (set to high). Note that nint output is not affected.

- **DBG_MODE**

Activates Protect Mode when set to high. See “Protect Mode” on page 23 for more information.

AIC2 Fast Forcing Enable Register

Register Name: AIC2_FFER

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Fast Forcing Enable**

0 = No effect.

1 = Enables fast forcing feature on corresponding interrupt.

AIC2 Fast Forcing Disable Register

Register Name: AIC2_FFDR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Fast Forcing Disable**

0 = No effect.

1 = Disables fast forcing feature on corresponding interrupt.

AIC2 Fast Forcing Status Register

Register Name: AIC2_FFSR

Access Type: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	IRQ11	IRQ10	IRQ9
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	IRQ8
7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	FIQ

- **Fast Forcing Status**

0 = Disabled.

1 = Fast forcing feature enabled on corresponding interrupt.

Application Examples

Standard Interrupt Sequence

For details on the registers mentioned in the steps below, refer to the ARM7TDMI Embedded Core datasheet.

It is assumed that:

1. The Advanced Interrupt Controller 2 has been programmed, AIC2_SVR is loaded with corresponding interrupt service routine addresses and interrupts are enabled.
2. The instruction at address 0x18 (IRQ exception vector address) is
   LDR PC, [PC, #-&F20]

When nIRQ is asserted, if the bit “I” of CPSR is 0, the sequence is:

1. The CPSR is stored in SPSR_irq, the current value of the Program Counter is loaded in the IRQ link register (R14_IRQ) and the Program Counter (R15) is loaded with 0x18. In the following cycle during fetch at address 0x1C, the ARM core adjusts R14_IRQ, decrementing it by four.
2. The ARM core enters IRQ mode, if it is not already.
3. When the instruction loaded at address 0x18 is executed, the Program Counter is loaded with the value read in AIC2_IVR. Reading the AIC2_IVR has the following effects:

   - Sets the current interrupt to be the pending one with the highest priority. The current level is the priority level of the current interrupt.

   - De-asserts the nIRQ line on the processor. (Even if vectoring is not used, AIC2_IVR must be read in order to de-assert nIRQ.)

   - Automatically clears the interrupt, if it has been programmed to be edge-triggered.

   - Pushes the current level on to the stack

   - Returns the value written in the AIC2_SVR corresponding to the current interrupt
4. The previous step has effect to branch to the corresponding interrupt service routine. This should start by saving the Link Register(R14_IRQ) and the SPSR(SPSR_IRQ). Note that the Link Register must be decremented by four when it is saved if it is to be restored directly into the Program Counter at the end of the interrupt. For example, the instruction SUB PC, LR, #4 may be used.
5. Further interrupts can then be unmasked by clearing the “I” bit in the CPSR, allowing re-assertion of the nIRQ to be taken into account by the core. This can occur if an interrupt with a higher priority than the current one occurs.
6. The Interrupt Handler can then proceed as required, saving the registers which will be used and restoring them at the end. During this phase, an interrupt of priority higher than the current level will restart the sequence from step 1. Note that if the interrupt is programmed to be level sensitive, the source of the interrupt must be cleared during this phase.
7. The “I” bit in the CPSR must be set in order to mask interrupts before exiting, to ensure that the interrupt is completed in an orderly manner.
8. The End of Interrupt Command Register (AIC2_EOICR) must be written in order to indicate to the AIC2 that the current interrupt is finished. This causes the current level to be popped from the stack, restoring the previous current level if one exists on the stack. If another interrupt is pending, with lower or equal priority than old current level but with higher priority than the new current level, the nIRQ line is re-asserted, but the interrupt sequence does not immediately start because the “I” bit is set in the core.

9. The SPSR (SPSR_IRQ) is restored. Finally, the saved value of the Link Register is restored directly into the PC. This has effect of returning from the interrupt to whatever was being executed before, and of loading the CPSR with the stored SPSR, masking or unmasking the interrupts depending on the state saved in the SPSR (the previous state of the ARM core).

Note: The "I" bit in the SPSR is significant. If it is set, it indicates that the ARM core was just about to mask IRQ interrupts when the mask instruction was interrupted. Hence, when the SPSR is restored, the mask instruction is completed (IRQ is masked).

Fast Interrupt Sequence

For details on the registers mentioned in the steps below, refer to the ARM7TDMI Embedded Core datasheet.

It is assumed that:

1. The Advanced Interrupt Controller 2 has been programmed, the AIC2_SVR register corresponding to FIQ interrupt (AIC2_SVR[0] in this example) is loaded with fast interrupt service routine address, and the fast interrupt is enabled.
2. The Instruction at address 0x1C(FIQ exception vector address) is:
   LDR PC, [PC, #-&F20].
3. Nested Fast Interrupts are not needed by the user.

When nFIQ is asserted, if the bit "F" of CPSR is 0, the sequence is:

1. The CPSR is stored in SPSR_fiq, the current value of the Program Counter is loaded in the FIQ link register (R14_FIQ) and the Program Counter (R15) is loaded with 0x1C. In the following cycle, during fetch at address 0x20, the ARM core adjusts R14_FIQ, decrementing it by four.
2. The ARM core enters FIQ mode.
3. When the instruction loaded at address 0x1C is executed, the Program Counter is loaded with the value read in AIC2_FVR. Reading the AIC2_FVR has effect of automatically clearing the fast interrupt (source 0 connected to the FIQ line), if it has been programmed to be edge triggered. In this case only, it de-asserts the nFIQ line on the processor.
4. The previous step has effect to branch to the corresponding interrupt service routine. It is not necessary to save the Link Register(R14_FIQ) and the SPSR(SPSR_FIQ) if nested fast interrupts are not needed.
5. The Interrupt Handler can then proceed as required. It is not necessary to save registers R8 to R13 because FIQ mode has its own dedicated registers and the user R8 to R13 are banked. The other registers, R0 to R7, must be saved before being used, and restored at the end (before the next step). Note that if the fast interrupt is programmed to be level sensitive, the source of the interrupt must be cleared during this phase in order to de-assert the nFIQ line.
6. Finally, the Link Register (R14_FIQ) is restored into the PC after decrementing it by four (with instruction SUB PC, LR, #4 for example). This has effect of returning from the interrupt to whatever was being executed before, and of loading the CPSR with the SPSR, masking or unmasking the fast interrupt depending on the state saved in the SPSR.

Note: The "F" bit in the SPSR is significant. If it is set, it indicates that the ARM core was just about to mask FIQ interrupts when the mask instruction was interrupted. Hence when the SPSR is restored, the interrupted instruction is completed (FIQ is masked).

Spurious Interrupt Sequence

A spurious interrupt is a signal of very short duration on one of the interrupt input lines. It is handled by the following sequence of actions.

1. When an interrupt is active, the AIC2 asserts the nIRQ (or nFIQ) line and the ARM7TDMI enters IRQ (or FIQ) mode.
   At this moment, if the interrupt source disappears, the nIRQ (or nFIQ) line is de-asserted but the ARM7TDMI continues with the interrupt handler.
2. If the IRQ Vector Register (AIC2_IVR) is read when the nIRQ is not asserted, the AIC2_IVR is read with the contents of the Spurious Interrupt Vector Register.
3. If the register FIQ Vector Register (AIC2_FVR) is read when the nFIQ is not asserted, the AIC2_FVR is read with the contents of the Spurious Interrupt Vector Register.
4. The Spurious Interrupt Routine must as a minimum write an end of interrupt command. It is sufficient to write to the end of interrupt command register AIC2_EOICR.
   Until the AIC2_EOICR write is received by the interrupt controller, the nIRQ (or nFIQ) line is not re-asserted.
5. This causes the ARM7TDMI to jump into the Spurious Interrupt Routine.
6. During a Spurious Interrupt Routine, the Interrupt Status Register AIC2_ISR reads 0.

Protect Mode

Protect Mode permits reading of the Interrupt Vector Register without performing the associated automatic operations. This is necessary when working with a debug system. When a debug monitor or an ICE reads the AIC2 User Interface, the IVR could be read. This has the following consequences in normal mode:

- If an enabled interrupt with a higher priority than the current one is pending, it is stacked.
- If there is no enabled pending interrupt, the spurious vector is returned.

In either case, an End of Interrupt command is necessary to acknowledge and to restore the context of the AIC2. This operation is generally not performed by the debug system. Hence the debug system would become strongly intrusive, and could cause the application to enter an undesired state.

This is avoided by using Protect Mode. Protect Mode is enabled by setting the bit DBG_MODE in the AIC2 Debug Control Register. When Protect Mode is enabled, the AIC2 performs interrupt stacking only when a write access is performed on the AIC2_IVR. Therefore, the Interrupt Service Routines must write (arbitrary data) to the AIC2_IVR just after reading it. The new context of the AIC2, including the value of the Interrupt Status Register (AIC2_ISR), is updated with the current interrupt only when IVR is written. An AIC2_IVR read on its own (e.g., by a debugger), modifies neither the AIC2 context nor the AIC2_ISR. Extra AIC2_IVR reads performed between the read and the write can cause unpredictable results. Therefore, it is strongly recommended not to set a breakpoint between these two actions, nor to stop the software.The debug system must not write to the AIC2_IVR as this would cause undesirable effects. Table 4 shows the main steps of an interrupt and the order in which they are performed according to the mode.

Table 4. Interrupt Steps

Action	Normal Mode	Protect Mode
Calculate active interrupt (higher that current or spurious)	Read AIC2_IVR	Read AIC2_IVR
Determine and return the vector of the active interrupt	Read AIC2_IVR	Read AIC2_IVR
Memorize interrupt	Read AIC2_IVR	Read AIC2_IVR
Push on internal stack the current priority level	Read AIC2_IVR	Write AIC2_IVR
Acknowledge the interrupt[1]	Read AIC2_IVR	Write AIC2_IVR
No effect[2]	Write AIC2_IVR	

Notes: 1. NIRQ de-assertion and automatic interrupt clearing if the source is programmed as level-sensitive.
2. Software that has been written and debugged using Protect Mode runs correctly in Normal Mode without modification. However, in Normal Mode the AIC2_IVR write has no effect and can be removed to optimize the code.

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

Europe
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Memory
Atmel Corporate
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 436-4270
FAX 1(408) 436-4314

Microcontrollers
Atmel Corporate
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 436-4270
FAX 1(408) 436-4314

Atmel Nantes
La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Atmel Rousset
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

Atmel Colorado Springs
1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Atmel Smart Card ICs
Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive
Atmel Heilbronn
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

Atmel Colorado Springs
1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Atmel Grenoble
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-76-58-30-00
FAX (33) 4-76-58-34-80

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

1796B–CASIC–03/02 0M