

27. Advanced Interrupt Controller (AIC)

27.1 Description

The Advanced Interrupt Controller (AIC) is an 8-level priority, individually maskable, vectored interrupt controller, providing handling of up to thirty-two interrupt sources. It is designed to substantially reduce the software and real-time overhead in handling internal and external interrupts.

The AIC drives the nFIQ (fast interrupt request) and the nIRQ (standard interrupt request) inputs of an ARM processor. Inputs of the AIC are either internal peripheral interrupts or external interrupts coming from the product's pins.

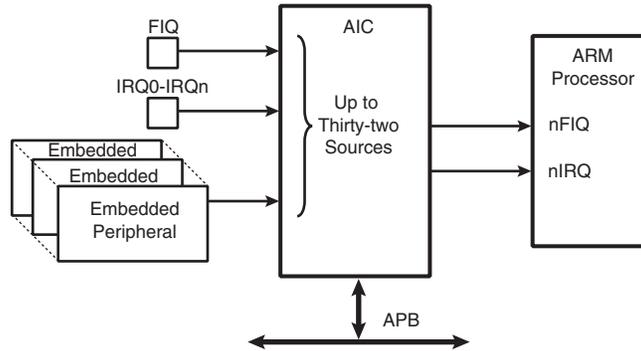
The 8-level Priority Controller allows the user to define the priority for each interrupt source, thus permitting higher priority interrupts to be serviced even if a lower priority interrupt is being treated.

Internal interrupt sources can be 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.

The fast forcing feature redirects any internal or external interrupt source to provide a fast interrupt rather than a normal interrupt.

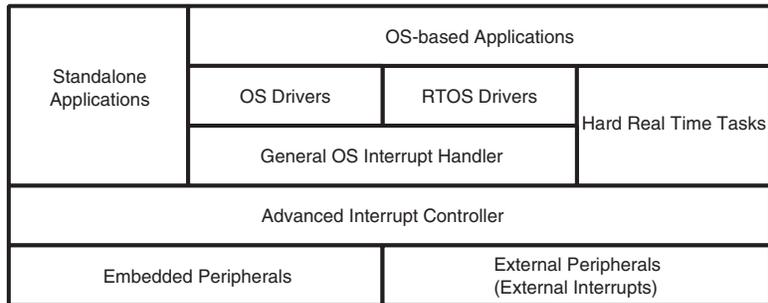
27.2 Block Diagram

Figure 27-1. Block Diagram



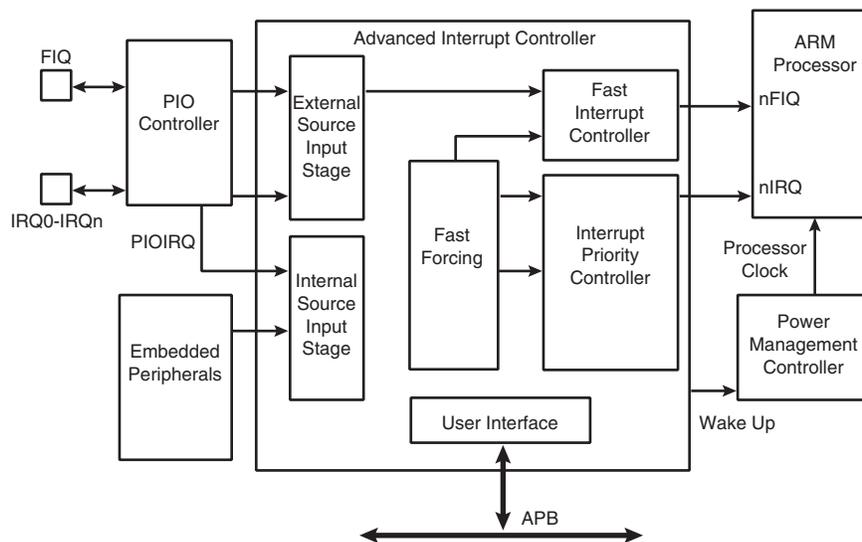
27.3 Application Block Diagram

Figure 27-2. Description of the Application Block



27.4 AIC Detailed Block Diagram

Figure 27-3. AIC Detailed Block Diagram



27.5 I/O Line Description

Table 27-1. I/O Line Description

Pin Name	Pin Description	Type
FIQ	Fast Interrupt	Input
IRQ0 - IRQn	Interrupt 0 - Interrupt n	Input

27.6 Product Dependencies

27.6.1 I/O Lines

The interrupt signals FIQ and IRQ0 to IRQn are normally multiplexed through the PIO controllers. Depending on the features of the PIO controller used in the product, the pins must be programmed in accordance with their assigned interrupt function. This is not applicable when the PIO controller used in the product is transparent on the input path.

27.6.2 Power Management

The Advanced Interrupt Controller is continuously clocked. The Power Management Controller has no effect on the Advanced Interrupt Controller behavior.

The assertion of the Advanced Interrupt Controller outputs, either nIRQ or nFIQ, wakes up the ARM processor while it is in Idle Mode. The General Interrupt Mask feature enables the AIC to wake up the processor without asserting the interrupt line of the processor, thus providing synchronization of the processor on an event.

27.6.3 Interrupt Sources

The Interrupt Source 0 is always located at FIQ. If the product does not feature an FIQ pin, the Interrupt Source 0 cannot be used.

The Interrupt Source 1 is always located at System Interrupt. This is the result of the OR-wiring of the system peripheral interrupt lines, such as the System Timer, the Real Time Clock, the Power Management Controller and the Memory Controller. When a system interrupt occurs, the service routine must first distinguish the cause of the interrupt. This is performed by reading successively the status registers of the above mentioned system peripherals.

The interrupt sources 2 to 31 can either be connected to the interrupt outputs of an embedded user peripheral or to external interrupt lines. The external interrupt lines can be connected directly, or through the PIO Controller.

The PIO Controllers are considered as user peripherals in the scope of interrupt handling. Accordingly, the PIO Controller interrupt lines are connected to the Interrupt Sources 2 to 31.

The peripheral identification defined at the product level corresponds to the interrupt source number (as well as the bit number controlling the clock of the peripheral). Consequently, to simplify the description of the functional operations and the user interface, the interrupt sources are named FIQ, SYS, and PID2 to PID31.

27.7 Functional Description

27.7.1 Interrupt Source Control

27.7.1.1 *Interrupt Source Mode*

The Advanced Interrupt Controller independently programs each interrupt source. The SRC_TYPE field of the corresponding AIC_SMR (Source Mode Register) selects the interrupt condition of each source.

The internal interrupt sources wired on the interrupt outputs of the embedded peripherals can be programmed either in level-sensitive mode or in edge-triggered mode. The active level of the internal interrupts is not important for the user.

The external interrupt sources can be programmed either in high level-sensitive or low level-sensitive modes, or in positive edge-triggered or negative edge-triggered modes.

27.7.1.2 *Interrupt Source Enabling*

Each interrupt source, including the FIQ in source 0, can be enabled or disabled by using the command registers; AIC_IECR (Interrupt Enable Command Register) and AIC_IDCR (Interrupt Disable Command Register). This set of registers conducts enabling or disabling in one instruction. The interrupt mask can be read in the AIC_IMR register. A disabled interrupt does not affect servicing of other interrupts.

27.7.1.3 *Interrupt Clearing and Setting*

All interrupt sources programmed to be edge-triggered (including the FIQ in source 0) can be individually set or cleared by writing respectively the AIC_ISCR and AIC_ICCR registers. Clearing or setting interrupt sources programmed in level-sensitive mode has no effect.

The clear operation is perfunctory, as the software must perform an action to reinitialize the “memorization” circuitry activated when the source is programmed in edge-triggered mode. However, the set operation is available for auto-test or software debug purposes. It can also be used to execute an AIC-implementation of a software interrupt.

The AIC features an automatic clear of the current interrupt when the AIC_IVR (Interrupt Vector Register) is read. Only the interrupt source being detected by the AIC as the current interrupt is affected by this operation. (See “Priority Controller” on page 291.) The automatic clear reduces the operations required by the interrupt service routine entry code to reading the AIC_IVR. Note that the automatic interrupt clear is disabled if the interrupt source has the Fast Forcing feature enabled as it is considered uniquely as a FIQ source. (For further details, See “Fast Forcing” on page 295.)

The automatic clear of the interrupt source 0 is performed when AIC_FVR is read.

27.7.1.4 *Interrupt Status*

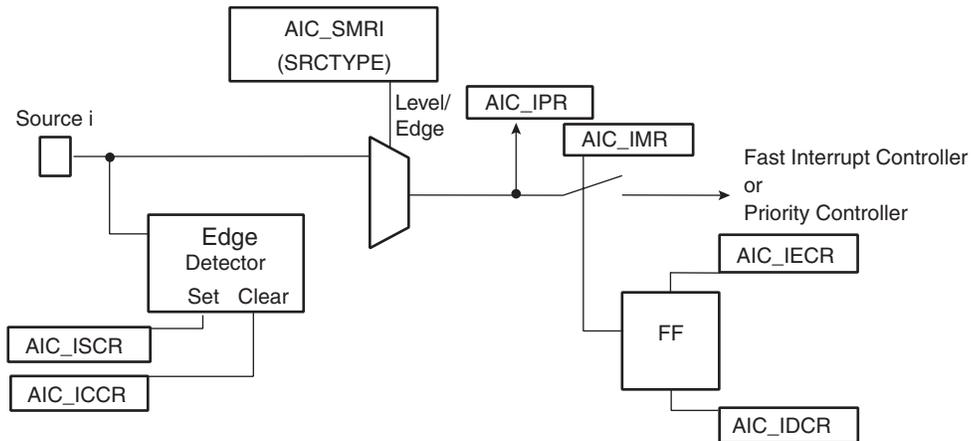
For each interrupt, the AIC operation originates in AIC_IPR (Interrupt Pending Register) and its mask in AIC_IMR (Interrupt Mask Register). AIC_IPR enables the actual activity of the sources, whether masked or not.

The AIC_ISR register reads the number of the current interrupt (see “Priority Controller” on page 291) and the register AIC_CISR gives an image of the signals nIRQ and nFIQ driven on the processor.

Each status referred to above can be used to optimize the interrupt handling of the systems.

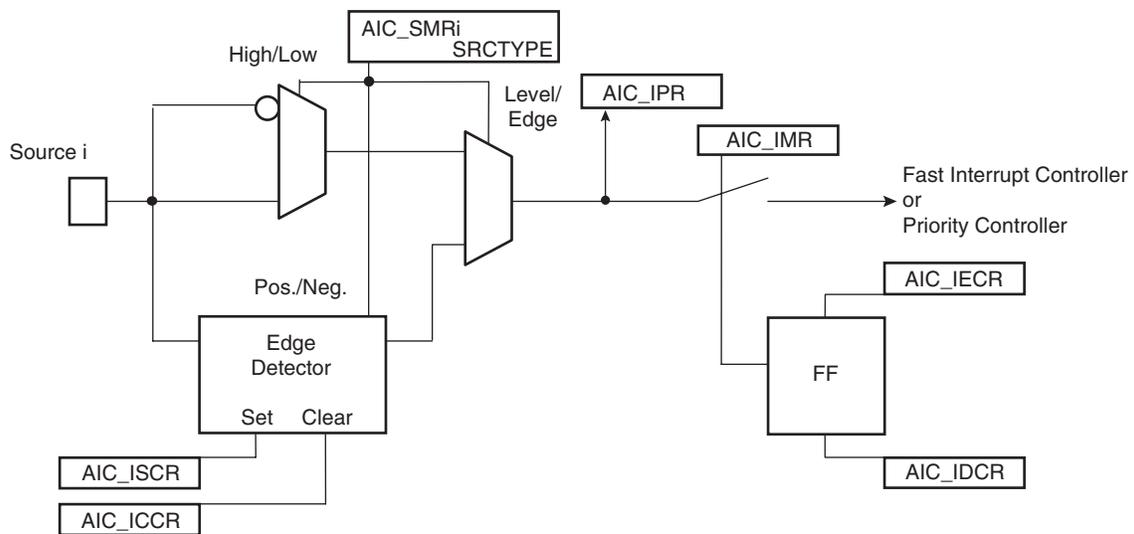
27.7.1.5 Internal Interrupt Source Input Stage

Figure 27-4. Internal Interrupt Source Input Stage



27.7.1.6 External Interrupt Source Input Stage

Figure 27-5. External Interrupt Source Input Stage



27.7.2 Interrupt Latencies

Global interrupt latencies depend on several parameters, including:

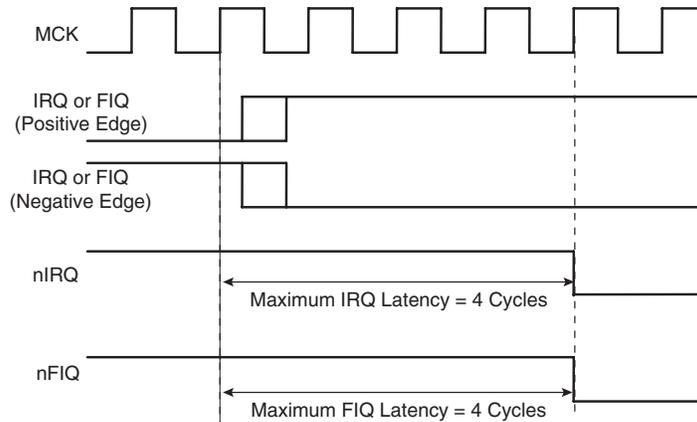
- The time the software masks the interrupts.
- Occurrence, either at the processor level or at the AIC level.
- The execution time of the instruction in progress when the interrupt occurs.
- The treatment of higher priority interrupts and the resynchronization of the hardware signals.

This section addresses only the hardware resynchronizations. It gives details of the latency times between the event on an external interrupt leading in a valid interrupt (edge or level) or the assertion of an internal interrupt source and the assertion of the nIRQ or nFIQ line on the processor. The resynchronization time depends on the programming of the interrupt source and on its type (internal or external). For the standard interrupt, resynchronization times are given assuming there is no higher priority in progress.

The PIO Controller multiplexing has no effect on the interrupt latencies of the external interrupt sources.

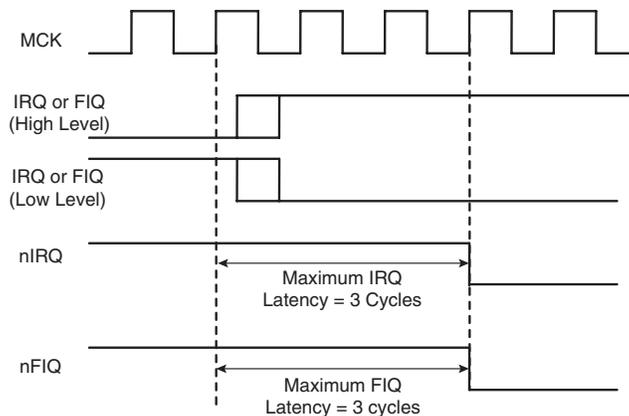
27.7.2.1 External Interrupt Edge Triggered Source

Figure 27-6. External Interrupt Edge Triggered Source



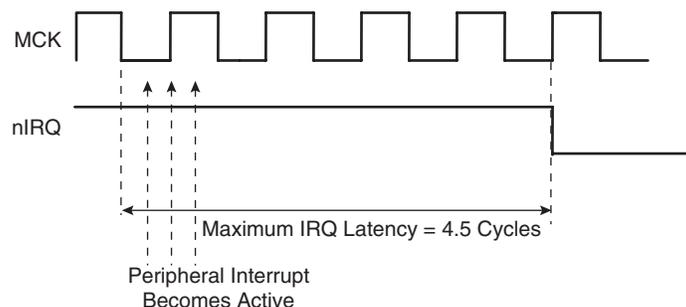
27.7.2.2 External Interrupt Level Sensitive Source

Figure 27-7. External Interrupt Level Sensitive Source



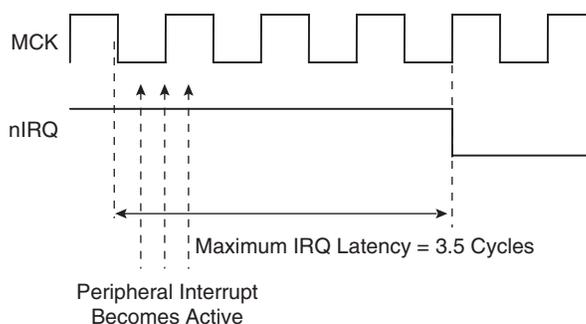
27.7.2.3 Internal Interrupt Edge Triggered Source

Figure 27-8. Internal Interrupt Edge Triggered Source



27.7.2.4 Internal Interrupt Level Sensitive Source

Figure 27-9. Internal Interrupt Level Sensitive Source



27.7.3 Normal Interrupt

27.7.3.1 Priority Controller

An 8-level priority controller drives the nIRQ line of the processor, depending on the interrupt conditions occurring on the interrupt sources 1 to 31 (except for those programmed in Fast Forcing).

Each interrupt source has a programmable priority level of 7 to 0, which is user-definable by writing the PRIOR field of the corresponding AIC_SMR (Source Mode Register). Level 7 is the highest priority and level 0 the lowest.

As soon as an interrupt condition occurs, as defined by the SRCTYPE field of the AIC_SMR (Source Mode Register), the nIRQ line is asserted. As a new interrupt condition might have happened on other interrupt sources since the nIRQ has been asserted, the priority controller determines the current interrupt at the time the AIC_IVR (Interrupt Vector Register) is read. **The read of AIC_IVR is the entry point of the interrupt handling** which allows the AIC to consider that the interrupt has been taken into account by the software.

The current priority level is defined as the priority level of the current interrupt.

If several interrupt sources of equal priority are pending and enabled when the AIC_IVR is read, the interrupt with the lowest interrupt source number is serviced first.

The nIRQ line can be asserted only if an interrupt condition occurs on an interrupt source with a higher priority. If an interrupt condition happens (or is pending) during the interrupt treatment in progress, it is delayed until the software indicates to the AIC the end of the current service by writing the AIC_EOICR (End of Interrupt Command Register). **The write of AIC_EOICR is the exit point of the interrupt handling.**

27.7.3.2 *Interrupt Nesting*

The priority controller utilizes interrupt nesting in order for the high priority interrupt to be handled during the service of lower priority interrupts. This requires the interrupt service routines of the lower interrupts to re-enable the interrupt at the processor level.

When an interrupt of a higher priority happens during an already occurring interrupt service routine, the nIRQ line is re-asserted. If the interrupt is enabled at the core level, the current execution is interrupted and the new interrupt service routine should read the AIC_IVR. At this time, the current interrupt number and its priority level are pushed into an embedded hardware stack, so that they are saved and restored when the higher priority interrupt servicing is finished and the AIC_EOICR is written.

The AIC is equipped with an 8-level wide hardware stack in order to support up to eight interrupt nestings pursuant to having eight priority levels.

27.7.3.3 *Interrupt Vectoring*

The interrupt handler addresses corresponding to each interrupt source can be stored in the registers AIC_SVR1 to AIC_SVR31 (Source Vector Register 1 to 31). When the processor reads AIC_IVR (Interrupt Vector Register), the value written into AIC_SVR corresponding to the current interrupt is returned.

This feature offers a way to branch in one single instruction to the handler corresponding to the current interrupt, as AIC_IVR is mapped at the absolute address 0xFFFF F100 and thus accessible from the ARM interrupt vector at address 0x0000 0018 through the following instruction:

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

When the processor executes this instruction, it loads the read value in AIC_IVR in its program counter, thus branching the execution on the correct interrupt handler.

This feature is often not used when the application is based on an operating system (either real time or not). Operating systems often have a single entry point for all the interrupts and the first task performed is to discern the source of the interrupt.

However, it is strongly recommended to port the operating system on AT91 products by supporting the interrupt vectoring. This can be performed by defining all the AIC_SVR of the interrupt source to be handled by the operating system at the address of its interrupt handler. When doing so, the interrupt vectoring permits a critical interrupt to transfer the execution on a specific very fast handler and not onto the operating system's general interrupt handler. This facilitates the support of hard real-time tasks (input/outputs of voice/audio buffers and software peripheral handling) to be handled efficiently and independently of the application running under an operating system.

27.7.3.4 *Interrupt Handlers*

This section gives an overview of the fast interrupt handling sequence when using the AIC. It is assumed that the programmer understands the architecture of the ARM processor, and especially the processor interrupt modes and the associated status bits.

It is assumed that:

1. The Advanced Interrupt Controller has been programmed, AIC_SVR registers are loaded with corresponding interrupt service routine addresses and interrupts are enabled.
2. The instruction at the ARM interrupt exception vector address is required to work with the vectoring

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

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

1. The CPSR is stored in SPSR_irq, the current value of the Program Counter is loaded in the Interrupt 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 Interrupt mode, if it has not already done so.
3. When the instruction loaded at address 0x18 is executed, the program counter is loaded with the value read in AIC_IVR. Reading the AIC_IVR has the following effects:
 - Sets the current interrupt to be the pending and enabled interrupt 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, AIC_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 and the current interrupt number on to the stack.
 - Returns the value written in the AIC_SVR corresponding to the current interrupt.
4. The previous step has the effect of branching to the corresponding interrupt service routine. This should start by saving the link register (R14_irq) and SPSR_IRQ. 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 CPSR, allowing re-assertion of the nIRQ to be taken into account by the core. This can happen if an interrupt with a higher priority than the current interrupt occurs.
6. The interrupt handler can then proceed as required, saving the registers that will be used and restoring them at the end. During this phase, an interrupt of higher priority than the current level will restart the sequence from step 1.

Note: 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 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 (AIC_EOICR) must be written in order to indicate to the AIC 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 the 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. SPSR_irq is restored. Finally, the saved value of the link register is restored directly into the PC. This has the effect of returning from the interrupt to whatever was being exe-

cutted before, and of loading the CPSR with the stored SPSR, masking or unmasking the interrupts depending on the state saved in SPSR_irq.

Note: The “I” bit in SPSR is significant. If it is set, it indicates that the ARM core was on the verge of masking an interrupt when the mask instruction was interrupted. Hence, when SPSR is restored, the mask instruction is completed (interrupt is masked).

27.7.4 Fast Interrupt

27.7.4.1 Fast Interrupt Source

The interrupt source 0 is the only source which can raise a fast interrupt request to the processor except if fast forcing is used. The interrupt source 0 is generally connected to a FIQ pin of the product, either directly or through a PIO Controller.

27.7.4.2 Fast Interrupt Control

The fast interrupt logic of the AIC has no priority controller. The mode of interrupt source 0 is programmed with the AIC_SMR0 and the field PRIOR of this register is not used even if it reads what has been written. The field SRCTYPE of AIC_SMR0 enables programming the fast interrupt source to be positive-edge triggered or negative-edge triggered or high-level sensitive or low-level sensitive

Writing 0x1 in the AIC_IECR (Interrupt Enable Command Register) and AIC_IDCR (Interrupt Disable Command Register) respectively enables and disables the fast interrupt. The bit 0 of AIC_IMR (Interrupt Mask Register) indicates whether the fast interrupt is enabled or disabled.

27.7.4.3 Fast Interrupt Vectoring

The fast interrupt handler address can be stored in AIC_SVR0 (Source Vector Register 0). The value written into this register is returned when the processor reads AIC_FVR (Fast Vector Register). This offers a way to branch in one single instruction to the interrupt handler, as AIC_FVR is mapped at the absolute address 0xFFFF F104 and thus accessible from the ARM fast interrupt vector at address 0x0000 001C through the following instruction:

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

When the processor executes this instruction it loads the value read in AIC_FVR in its program counter, thus branching the execution on the fast interrupt handler. It also automatically performs the clear of the fast interrupt source if it is programmed in edge-triggered mode.

27.7.4.4 Fast Interrupt Handlers

This section gives an overview of the fast interrupt handling sequence when using the AIC. It is assumed that the programmer understands the architecture of the ARM processor, and especially the processor interrupt modes and associated status bits.

Assuming that:

1. The Advanced Interrupt Controller has been programmed, AIC_SVR0 is loaded with the fast interrupt service routine address, and the interrupt source 0 is enabled.
2. The Instruction at address 0x1C (FIQ exception vector address) is required to vector the fast interrupt:

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

3. The user does not need nested fast interrupts.

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 AIC_FVR. Reading the AIC_FVR has effect of automatically clearing the fast interrupt, 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 enables branching to the corresponding interrupt service routine. It is not necessary to save the link register R14_fiq and 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 interrupt source 0.
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 the effect of returning from the interrupt to whatever was being executed before, loading the CPSR with the SPSR and masking or unmasking the fast interrupt depending on the state saved in the SPSR.

Note: The “F” bit in 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).

Another way to handle the fast interrupt is to map the interrupt service routine at the address of the ARM vector 0x1C. This method does not use the vectoring, so that reading AIC_FVR must be performed at the very beginning of the handler operation. However, this method saves the execution of a branch instruction.

27.7.4.5 Fast Forcing

The Fast Forcing feature of the advanced interrupt controller provides redirection of any normal Interrupt source on the fast interrupt controller.

Fast Forcing is enabled or disabled by writing to the Fast Forcing Enable Register (AIC_FFER) and the Fast Forcing Disable Register (AIC_FFDR). Writing to these registers results in an update of the Fast Forcing Status Register (AIC_FFSR) that controls the feature for each internal or external interrupt source.

When Fast Forcing is disabled, the interrupt sources are handled as described in the previous pages.

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

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

If the interrupt source is programmed in edge-triggered mode and an active edge is detected, 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 (AIC_IPR).

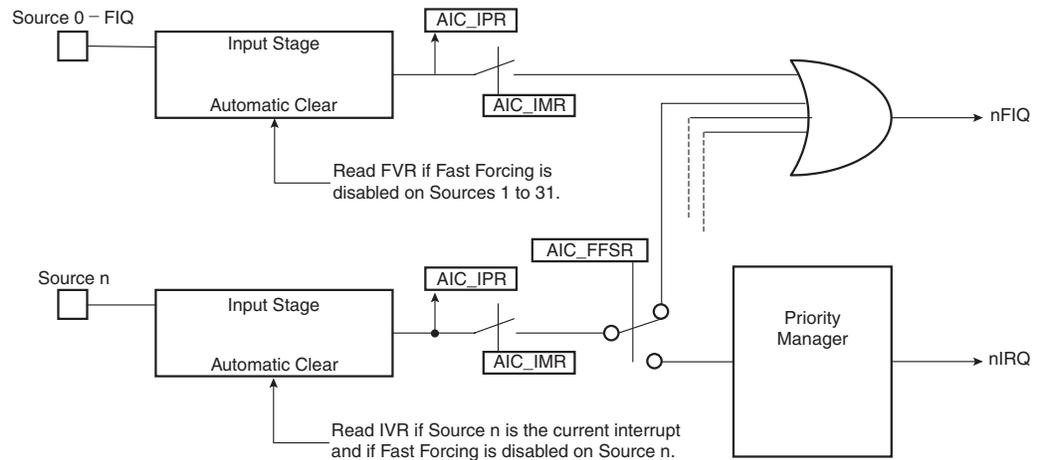
The FIQ Vector Register (AIC_FVR) reads the contents of the Source Vector Register 0 (AIC_SVR0), whatever the source of the fast interrupt may be. The read of the FVR does not clear the Source 0 when the fast forcing feature is used and the interrupt source should be cleared by writing to the Interrupt Clear Command Register (AIC_ICCR).

All enabled and pending interrupt sources that have the fast forcing feature enabled and that are programmed in edge-triggered 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 AIC_IVR does not clear the source that has the fast forcing feature enabled.

The source 0, reserved to the fast interrupt, continues operating normally and becomes one of the Fast Interrupt sources.

Figure 27-10. Fast Forcing



27.7.5 Protect Mode

The Protect Mode permits reading the Interrupt Vector Register without performing the associated automatic operations. This is necessary when working with a debug system. When a debugger, working either with a Debug Monitor or the ARM processor's ICE, stops the applications and updates the opened windows, it might read the AIC User Interface and thus the IVR. This has undesirable consequences:

- 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 AIC. This operation is generally not performed by the debug system as the debug system would become strongly intrusive and cause the application to enter an undesired state.

This is avoided by using the Protect Mode. Writing DBGM in AIC_DCR (Debug Control Register) at 0x1 enables the Protect Mode.

When the Protect Mode is enabled, the AIC performs interrupt stacking only when a write access is performed on the AIC_IVR. Therefore, the Interrupt Service Routines must write (arbitrary data) to the AIC_IVR just after reading it. The new context of the AIC, including the value of the

Interrupt Status Register (AIC_ISR), is updated with the current interrupt only when AIC_IVR is written.

An AIC_IVR read on its own (e.g., by a debugger), modifies neither the AIC context nor the AIC_ISR. Extra AIC_IVR reads perform the same operations. However, it is recommended to not stop the processor between the read and the write of AIC_IVR of the interrupt service routine to make sure the debugger does not modify the AIC context.

To summarize, in normal operating mode, the read of AIC_IVR performs the following operations within the AIC:

1. Calculates active interrupt (higher than current or spurious).
2. Determines and returns the vector of the active interrupt.
3. Memorizes the interrupt.
4. Pushes the current priority level onto the internal stack.
5. Acknowledges the interrupt.

However, while the Protect Mode is activated, only operations 1 to 3 are performed when AIC_IVR is read. Operations 4 and 5 are only performed by the AIC when AIC_IVR is written.

Software that has been written and debugged using the Protect Mode runs correctly in Normal Mode without modification. However, in Normal Mode the AIC_IVR write has no effect and can be removed to optimize the code.

27.7.6 Spurious Interrupt

The Advanced Interrupt Controller features protection against spurious interrupts. A spurious interrupt is defined as being the assertion of an interrupt source long enough for the AIC to assert the nIRQ, but no longer present when AIC_IVR is read. This is most prone to occur when:

- An external interrupt source is programmed in level-sensitive mode and an active level occurs for only a short time.
- An internal interrupt source is programmed in level sensitive and the output signal of the corresponding embedded peripheral is activated for a short time. (As in the case for the Watchdog.)
- An interrupt occurs just a few cycles before the software begins to mask it, thus resulting in a pulse on the interrupt source.

The AIC detects a spurious interrupt at the time the AIC_IVR is read while no enabled interrupt source is pending. When this happens, the AIC returns the value stored by the programmer in AIC_SPU (Spurious Vector Register). The programmer must store the address of a spurious interrupt handler in AIC_SPU as part of the application, to enable an as fast as possible return to the normal execution flow. This handler writes in AIC_EOICR and performs a return from interrupt.

27.7.7 General Interrupt Mask

The AIC features a General Interrupt Mask bit to prevent interrupts from reaching the processor. Both the nIRQ and the nFIQ lines are driven to their inactive state if the bit GMSK in AIC_DCR (Debug Control Register) is set. However, this mask does not prevent waking up the processor if it has entered Idle Mode. This function facilitates synchronizing the processor on a next event and, as soon as the event occurs, performs subsequent operations without having to handle an interrupt. It is strongly recommended to use this mask with caution.

27.8 Advanced Interrupt Controller (AIC) User Interface

27.8.1 Base Address

The AIC is mapped at the address **0xFFFF F000**. It has a total 4-Kbyte addressing space. This permits the vectoring feature, as the PC-relative load/store instructions of the ARM processor support only a ± 4 -Kbyte offset.

27.8.2 Register Mapping

Table 27-2. Register Mapping

Offset	Register	Name	Access	Reset
0000	Source Mode Register 0	AIC_SMR0	Read-write	0x0
0x04	Source Mode Register 1	AIC_SMR1	Read-write	0x0
...
0x7C	Source Mode Register 31	AIC_SMR31	Read-write	0x0
0x80	Source Vector Register 0	AIC_SVR0	Read-write	0x0
0x84	Source Vector Register 1	AIC_SVR1	Read-write	0x0
...
0xFC	Source Vector Register 31	AIC_SVR31	Read-write	0x0
0x100	Interrupt Vector Register	AIC_IVR	Read-only	0x0
0x104	FIQ Interrupt Vector Register	AIC_FVR	Read-only	0x0
0x108	Interrupt Status Register	AIC_ISR	Read-only	0x0
0x10C	Interrupt Pending Register ⁽²⁾	AIC_IPR	Read-only	0x0 ⁽¹⁾
0x110	Interrupt Mask Register ⁽²⁾	AIC_IMR	Read-only	0x0
0x114	Core Interrupt Status Register	AIC_CISR	Read-only	0x0
0x118	Reserved
0x11C	Reserved
0x120	Interrupt Enable Command Register ⁽²⁾	AIC_IEMR	Write-only	...
0x124	Interrupt Disable Command Register ⁽²⁾	AIC_IDCR	Write-only	...
0x128	Interrupt Clear Command Register ⁽²⁾	AIC_ICCR	Write-only	...
0x12C	Interrupt Set Command Register ⁽²⁾	AIC_ISCR	Write-only	...
0x130	End of Interrupt Command Register	AIC_EOICR	Write-only	...
0x134	Spurious Interrupt Vector Register	AIC_SPU	Read-write	0x0
0x138	Debug Control Register	AIC_DCR	Read-write	0x0
0x13C	Reserved
0x140	Fast Forcing Enable Register ⁽²⁾	AIC_FFER	Write-only	...
0x144	Fast Forcing Disable Register ⁽²⁾	AIC_FFDR	Write-only	...
0x148	Fast Forcing Status Register ⁽²⁾	AIC_FFSR	Read-only	0x0

- Notes:
1. The reset value of this register depends on the level of the external interrupt source. All other sources are cleared at reset, thus not pending.
 2. PID2...PID31 bit fields refer to the identifiers as defined in the Peripheral Identifiers Section of the product datasheet.

27.8.3 AIC Source Mode Register

Register Name: AIC_SMR0..AIC_SMR31

Access Type: Read-write

Reset Value: 0x0

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			0

- **PRIOR: Priority Level**

Programs 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 AIC_SMRx.

- **SRCTYPE: Interrupt Source Type**

The active level or edge is not programmable for the internal interrupt sources.

SRCTYPE		Internal Interrupt Sources	External Interrupt Sources
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

27.8.4 AIC Source Vector Register

Register Name: AIC_SVR0..AIC_SVR31

Access Type: Read-write

Reset Value: 0x0

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: Source Vector**

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

27.8.5 AIC Interrupt Vector Register

Register Name: AIC_IVR

Access Type: Read-only

Reset Value: 0x0

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

The Source Vector Register is indexed using the current interrupt number when the Interrupt Vector Register is read.

When there is no current interrupt, the Interrupt Vector Register reads the value stored in AIC_SPU.

27.8.6 AIC FIQ Vector Register

Register Name: AIC_FVR

Access Type: Read-only

Reset Value: 0x0

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. When there is no fast interrupt, the FIQ Vector Register reads the value stored in AIC_SPU.

27.8.7 AIC Interrupt Status Register

Register Name: AIC_ISR

Access Type: Read-only

Reset Value: 0x0

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	
-	-	-	IRQID					-

- **IRQID: Current Interrupt Identifier**

The Interrupt Status Register returns the current interrupt source number.

27.8.8 AIC Interrupt Pending Register

Register Name: AIC_IPR

Access Type: Read-only

Reset Value: 0x0

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

• **FIQ, SYS, PID2-PID31: Interrupt Pending**

0 = Corresponding interrupt is not pending.

1 = Corresponding interrupt is pending.

27.8.9 AIC Interrupt Mask Register

Register Name: AIC_IMR

Access Type: Read-only

Reset Value: 0x0

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

• **FIQ, SYS, PID2-PID31: Interrupt Mask**

0 = Corresponding interrupt is disabled.

1 = Corresponding interrupt is enabled.

27.8.10 AIC Core Interrupt Status Register

Register Name: AIC_CISR

Access Type: Read-only

Reset Value: 0x0

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
–	–	–	–	–	–	NIRQ	NFIQ

- **NFIQ: NFIQ Status**

0 = nFIQ line is deactivated.

1 = nFIQ line is active.

- **NIRQ: NIRQ Status**

0 = nIRQ line is deactivated.

1 = nIRQ line is active.

27.8.11 AIC Interrupt Enable Command Register

Register Name: AIC_IECR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

- **FIQ, SYS, PID2-PID31: Interrupt Enable**

0 = No effect.

1 = Enables corresponding interrupt.

27.8.12 AIC Interrupt Disable Command Register

Register Name: AIC_IDCR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

• **FIQ, SYS, PID2-PID31: Interrupt Disable**

0 = No effect.

1 = Disables corresponding interrupt.

27.8.13 AIC Interrupt Clear Command Register

Register Name: AIC_ICCR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

• **FIQ, SYS, PID2-PID31: Interrupt Clear**

0 = No effect.

1 = Clears corresponding interrupt.

27.8.14 AIC Interrupt Set Command Register

Register Name: AIC_ISCR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

- **FIQ, SYS, PID2-PID31: Interrupt Set**

0 = No effect.

1 = Sets corresponding interrupt.

27.8.15 AIC End of Interrupt Command Register

Register Name: AIC_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.

27.8.16 AIC Spurious Interrupt Vector Register

Register Name: AIC_SPU

Access Type: Read-write

Reset Value: 0x0

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**

The user may store the address of a spurious interrupt handler in this register. The written value is returned in AIC_IVR in case of a spurious interrupt and in AIC_FVR in case of a spurious fast interrupt.

27.8.17 AIC Debug Control Register

Register Name: AIC_DCR

Access Type: Read-write

Reset Value: 0x0

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
-	-	-	-	-	-	GMSK	PROT

- **PROT: Protection Mode**

0 = The Protection Mode is disabled.

1 = The Protection Mode is enabled.

- **GMSK: General Mask**

0 = The nIRQ and nFIQ lines are normally controlled by the AIC.

1 = The nIRQ and nFIQ lines are tied to their inactive state.

27.8.18 AIC Fast Forcing Enable Register

Register Name: AIC_FFER

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	–

- **SYS, PID2-PID31: Fast Forcing Enable**

0 = No effect.

1 = Enables the fast forcing feature on the corresponding interrupt.

27.8.19 AIC Fast Forcing Disable Register

Register Name: AIC_FFDR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	–

- **SYS, PID2-PID31: Fast Forcing Disable**

0 = No effect.

1 = Disables the Fast Forcing feature on the corresponding interrupt.

27.8.20 AIC Fast Forcing Status Register

Register Name: AIC_FFSR

Access Type: Read-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	–

- **SYS, PID2-PID31: Fast Forcing Status**

0 = The Fast Forcing feature is disabled on the corresponding interrupt.

1 = The Fast Forcing feature is enabled on the corresponding interrupt.