AVR32101: Configuring the AVR32 Interrupt Controller

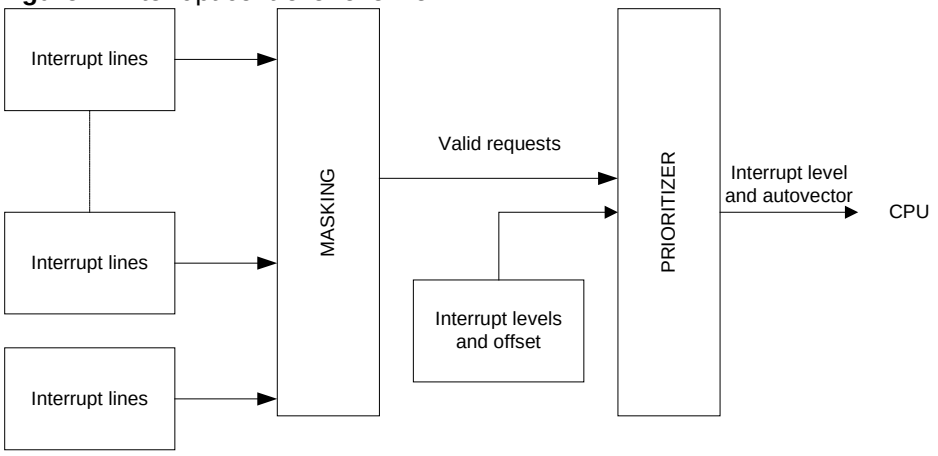
Features

- **Flexible interrupt handling**
- **Multiple levels, groups and priorities**
  - **User configurable**
- **Low latency**
- **Nested interrupts**

1 Introduction

The AVR®32 has highly flexible interrupt controller, which can be configured to suit a broad variety of implementation needs. Interrupts are grouped, and each group is given an interrupt level. As this is controllable by software, prioritization of different interrupts can be customized to suit system requirements. The interrupt controller manages interrupt prioritization in hardware, while the setup of these features is done in software and is covered in this appnote.

Figure 1: Interrupt controller overview



32-bit AVR® Microcontrollers

Application Note

Rev. 32010A-AVR-04/06

2 Interrupt organization

2.1 Overview

The interrupt controller featured on the AVR32 series is highly flexible. One or more interrupt lines are grouped together. These groups of interrupts are given an interrupt level, and therefore the interrupt itself has the same interrupt level as the group. A non-maskable interrupt (NMI) is also available. This level is reserved for highly critical interrupts that may occur in a system.

2.2 Interrupt levels

There are distinctive interrupt levels, namely level 0 through 3. The highest priority level is level 3. Higher prioritized interrupts have a larger number of shadowed registers than interrupts with lower priority. This ensures low latency interrupt handling. NMI interrupts are handled as a separate group and has its own interrupt level. This level has a higher priority than the other levels.

Each interrupt level has its own associated mode. The AVR32 changes the mode of operation according to the level of the interrupt that is currently active and thus giving each interrupt level it's own context.

2.2.1 Shadow registers

If a register is labeled as shadowed, the value of the registered is stored whenever a mode change occurs. When the actual mode is reentered, the previous values are restored through the use these shadow registers. The context switch due to a change in the running mode can thus be kept to a minimum. The number of shadowed registers is different from mode to mode and from part to part. Consult the datasheet for your specific device to the get the number of shadowed registers available for distinctive modes.

2.2.2 Masking

Global interrupt masking is available. In addition all interrupt levels can be individually masked. A non-maskable interrupt cannot be masked, as the name indicates.

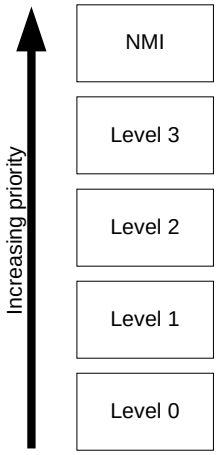
Whenever an interrupt occur, hardware automatically sets the mask bits corresponding to all sources with equal or lower priority.

Masking an interrupt level is by all practical means the same as disabling it.

2.3 Grouping

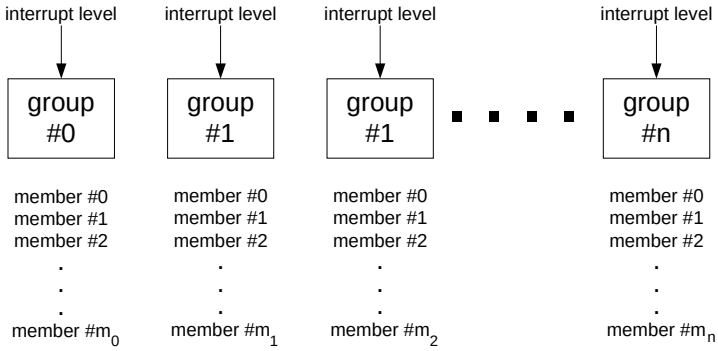
Each interrupt is member of a group and that group is then associated with an interrupt level. That gives the specific interrupt an interrupt level.

Figure 2. Interrupt level priority



The number of groups and members per group is specific to each individual part.

Figure 3. Interrupt grouping



In Figure 3 the number of groups is labeled as *n*, while the number of members per group is referred to as *m*. To find the number of groups and members per group, consult the datasheet for your specific device.

The maximum number of different groups available is 64 and the number of interrupts in a group is 32 or lower. These are maximum numbers, and very often there are groups that have few members. Typically interrupt lines from modules are ordered into different groups, though not all groups may be available in your selected device. Consult the datasheet for your specific part for more information.

2.4 Priority

If several interrupts occur at the same time, selecting the highest prioritized interrupt is done in hardware. The following rules apply:

- If only one interrupt group request is issued, the pending interrupt will be handled
- If two or more groups have pending interrupts, the group with the highest interrupt level will be handled first

- If two or more groups have pending interrupts, and they all have the same priority, the lowest numbered group will be handled first

The last rule also applies in the case that two or more members of a single group issue an interrupt. The lowest numbered irq-line has the highest priority and therefore handled first.

3 Interrupt controller

In order to utilize the interrupt-handling scheme available in the AVR32, the interrupt controller must first be initialized. Every interrupt that can occur must be registered before normal execution starts to ensure a well-behaved system.

3.1 Registers

The AVR32 interrupt controller uses set of different registers, which is unique for each group. This chapter has a brief description of the register operation associated with the interrupt controller. The user interface is covered in depth in your datasheet.

3.1.1 Interrupt Request Register

There is one Interrupt Request Registers (IRR) for every group. Each IRR has 32 bits, one for each possible interrupt request for that group. When an interrupt is pending, the bitfield for the corresponding interrupt is asserted. These registers are read-only. These registers are labeled INTC_IRR*m*, where *m* is the actual group.

3.1.2 Interrupt Priority Registers

There is a number Interrupt Priority Registers (IPR) that reflects the number of groups. These registers have a reset value of zero. In these registries there are two bitfields, one named INTLEVEL and one named AUTOVECTOR. The INTLEVEL field gives the level of the group while the AUTOVECTOR field gives an offset. This offset is relative to EVBA (Exception Vector Base Address) and contains the interrupt handler for the group. As there is an IPR for each group, these registers are labeled in the same matter as INTC_IRR*m*; INTC_IPR*m*, where m is the group number.

EVBA is also used for other parts of the system (like exceptions) and defines an offset for all relative interrupt handler addresses.

3.1.3 Interrupt Cause Registers

There are four Interrupt Cause Registers (ICR). They are all dedicated to an interrupt label, 0 through 3, and named INTC_ICR*n*, where *n* is the interrupt level. The group that has triggered the interrupt is stored here, whenever an interrupt occurs.

3.1.4 Interrupt masking

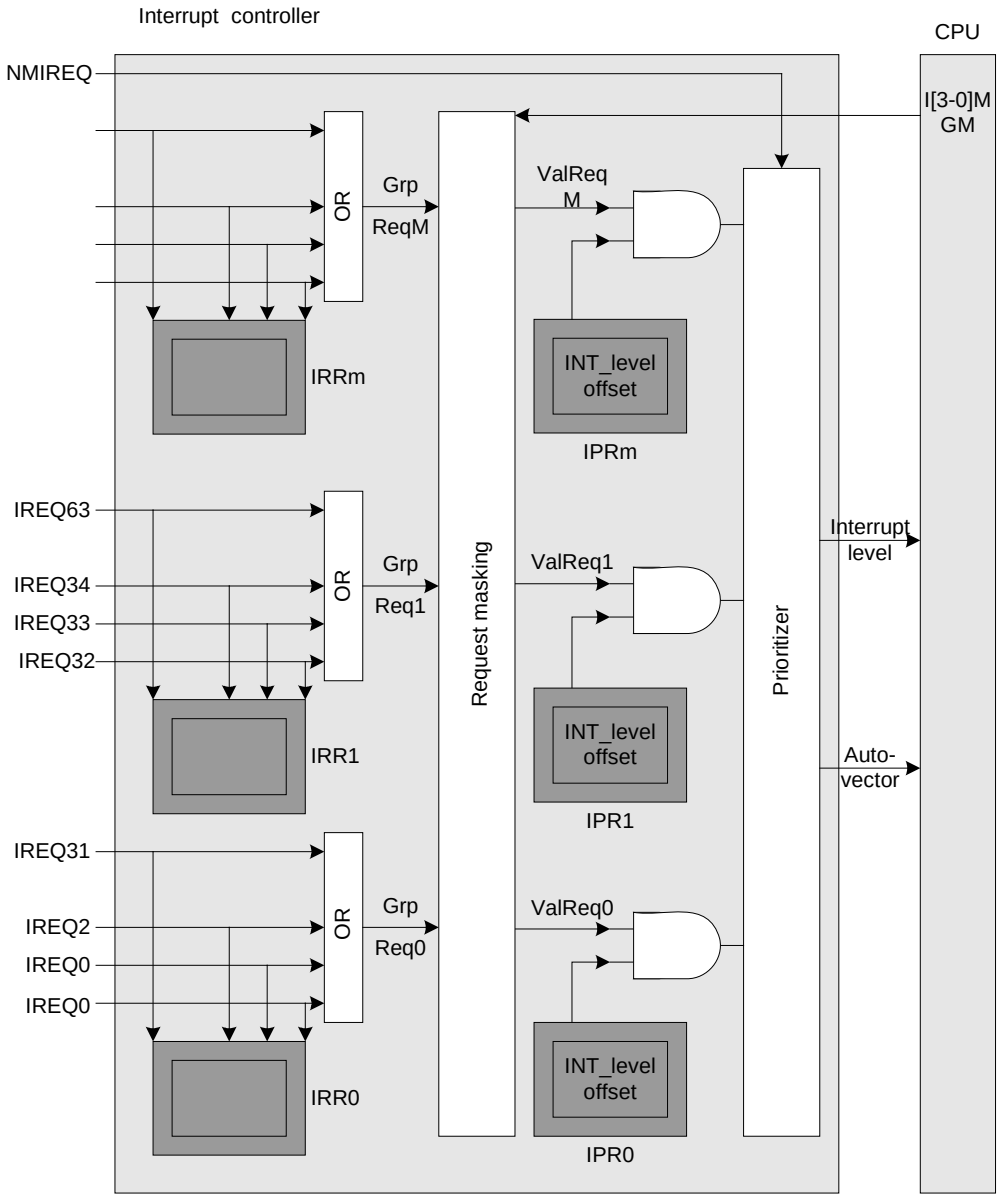
All masking of interrupt is done in the Status Register (SR). As described in chapter 2.2.2, there are two ways for each interrupt level to be masked; either by global masking or by interrupt level masking. Global masking can be applied by setting the GM bit in SR, and this is default set after a reset. To disable a specified interrupt level, the corresponding masking bit in SR must be set for the interrupt level. These bits are named I*n*M, where *n* is the interrupt level.

An important note concerning masking is that the GM bit overrules the individual masking of the different interrupt levels.

3.2 NMI

A NMI is by nature non-maskable. This is also a type of interrupt that seldom should occur. It treated independently of the group handling which apply to other interrupt sources.

Figure 4: The AVR32 Interrupt Controller



3.3 Identifying interrupts

When an interrupt is asserted, the CPU does not need to know which of the numerous interrupt lines that have trigged. It only needs to know what level the interrupt has and the autovector to jump to a specified memory location relative to EVBA. If more information is needed, to handle the interrupt request, the registers listed in chapter 3.1 can be used to identify the source.

3.4 Selecting interrupts

Whenever interrupts occur, the correct interrupt function must be run and then return to normal execution unless no other interrupts are pending. An interrupt does not return in the same matter as normal functions. This is due to the fact that the AVR32 changes the active operating mode whenever an interrupt occurs. To ensure correct mode transition, all interrupts must be handled as such. To register the physical interrupt in the interrupt controller, the address of the interrupt handler must be provided, as well as group, line and priority.

Each module's interrupt request lines (IRQ) are numbered, which is defined in the part specific header file. This number consists of both the group, which the interrupt is a member of, and the line number for the member. Modulo 32 of the IRQ gives the line number, while the positive quotient gives the group.

- IRQ % 32 = line number
- IRQ / 32 = group

This information needs to be passed to the interrupt controller to successfully register the interrupt handler for the specific interrupt.

3.5 Enabling interrupts

When all interrupts that might occur have been successfully registered, interrupts can be enabled. After reset, the AVR32 has by default global interrupt masking turned on. It is essential that global interrupt masking be left on until the interrupt controller has been correctly initialized to ensure a well-defined behavior.

4 Interface

To provide the flexible interrupt handling in the AVR32, the interrupt controller manages these settings. This chapter describes the interface to the interrupt controller.

4.1 Writing an interrupt handler

Whenever an interrupt occurs, the autovector and interrupt level is passed on to the CPU and the jump and mode change is issued. I.e. on the AP7000 a NMI interrupt is specified to present at EVBA+0x10. When an interrupt is asserted, a mode change is made. The interrupt handler's location must be specified relative to EVBA (Exception Vector Base Address). An interrupt handler may be regarded as a normal function without any arguments, but with a special return instruction instead. Different compilers use various type definitions and conventions for identifying an interrupt handler. For examples of interrupt handlers for different compilers, consult chapter 6.

4.2 Adding an interrupt handler

When you have written different types of interrupt handlers for various interrupts that may occur in your system, they must be added to the system. Then information as interrupt group, interrupt line and priority must be issued alongside the interrupt handler to register the interrupt in your system. As with writing interrupt handlers, these functions are compiler specific. Chapter 6 includes references to example code for adding interrupt handlers to different compilers.

When all appropriate handlers for your system is added to the interrupt controller, clearing the GM bit in SR enable interrupts for your system.

5 Design considerations

Whenever designing an interrupt configuration for a specific system, care should be taken to get the optimal performance and the best system behavior.

5.1 Nested interrupts

As all interrupts with the same and lower priority is automatically masked whenever an interrupt is triggered, lower priority interrupts will not preempt the current running interrupt. An interrupt with a higher priority may nevertheless preempt the current running interrupt and thus nested interrupts is enabled by default. To disable nested interrupts, set the global interrupt mask bit when entering an interrupt routine and in turn clear it when the routine is completed. Then the penalty of a context switch must be taken into account. See chapter 2.2.1 for additional information.

If an interrupt enters a critical section, all other interrupts should be masked while the handler is currently executing in a critical section.

5.2 Interrupt routines

Interrupt routines should as a rule of thumb be kept as small as possible. Other interrupts may have a lower priority, but that does not make them unimportant. This is of even greater importance if nested interrupts is disabled, as a higher priority interrupt may be blocked while an interrupt of lower priority is being executed.

6 Package information

Included with the application note is a example source code for different compilers. The different examples are named interrupt_ex_<*compiler*>.c, where <*compiler*> is the name of the compiler that supports the convention used.

6.1 Documentation

Function specific documentation is available in the package. Refer to readme.html.

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
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
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

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
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

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

Literature Requests
www.atmel.com/literature

32010A-AVR-04/06