Interrupt Management: Auto-vectoring and Prioritization

Background

The AT91 is based on the ARM7TDMI™ microcontroller core. It features the Advanced Interrupt Controller (AIC), an 8-level priority, individually maskable, vectored interrupt controller.

This microcontroller core implements two physically independent sources of interrupt:

- FIQ - Fast Interrupt
- IRQ - Normal Interrupt

Each of these interrupts has a corresponding vector, at addresses 0x00000018 for the IRQ and 0x0000001C for the FIQ.

The AIC is connected to the NFIQ (Fast Interrupt Request) and the NIRQ (Standard Interrupt Request) inputs of the ARM7TDMI processor.

The processor's NFIQ line can only be asserted by the external fast interrupt request input: FIQ (multiplexed with the PIO P12). Therefore, when an FIQ occurs, it is not necessary to de-multiplex the handler according to the cause of the interrupt (it is assumed that there is no multiplexing added by the external hardware). The FIQ management code can be reached either directly from the vector (0x0000001C), or by using the Fast Interrupt Vector Register (AIC_FVR) as described in the datasheet of the AT91 products.

The NIRQ line can be asserted by the interrupts generated by the on-chip peripherals and the external interrupt request lines: IRQ0 to IRQ2. Therefore it is necessary to manage a prioritization when several interrupt sources are asserted at once and to de-multiplex the handler according to the source of the interrupt.

AT91 Series
ARM® Thumb® Microcontrollers

Application Note

Rev. 1168A–10/98

Auto-Vectoring

This feature consists of a set of registers which provide the address of the handler to execute according to the source of an interrupt.

Each interrupt source is associated with a Source Vector Register (AIC_SVR1 - AIC_SVR31) which contains the address of the function corresponding to the active interrupt. When the Interrupt Vector Register (AIC_IVR) is read, it automatically returns the contents of the source vector register corresponding to the active interrupt with the highest priority. Note that AIC_IVR is located at address 0xFFFFF100.

During the boot sequence and before enabling the interrupts, the software must:

1. Initialize the source vector registers for each interrupt
2. Initialize the IRQ vector at address 0x00000018 with the following code:

```
ldr   pc,[pc,#-0xF20]
```

When an interrupt occurs, the core performs the following (see the ARM Architectural Reference Manual):

R14_irq = address of next instruction to be executed + 4

SPSR_irq = CPSR

CPSR[5:0] = 0b010010 Interrupt mode

CPSR[6] = unchanged Fast interrupt status is unchanged

CPSR[7] = 1 Normal interrupts disabled

PC = 0x00000018

When the instruction at the address 0x00000018 is executed, the effective address is:

0x00000020 - 0x0F20 = 0xFFFFF100

(0x00000020 is the value of the PC when the instruction at address 0x18 is executed)

This causes the core to load the PC with the value read in AIC_IVR which returns the value of AIC_SVR corresponding to the active interrupt. This has the effect of directly jumping to the correct interrupt service routine.

Also note that when the AIC_IVR is read, the AIC does the following:

- deasserts the NIRQ line on the core
- determines which pending interrupt has the highest priority
- pushes the level of this interrupt in its internal hardware stack
- clears the interrupt if it is configured to be edge triggered

The interrupt level is popped when the End of Interrupt (EOI) is indicated to the AIC by a write in AIC_EOICR (see “Prioritization” on page 3).

Prioritization

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 AIC receives more than one unmasked interrupt at a time, the interrupt with the highest priority is serviced first. The interrupt management of the interrupt with the lower priority level is therefore delayed.

The AIC manages the prioritization by using an internal stack on which the current interrupt level is automatically pushed when AIC_IVR is read, and popped when AIC_EOICR is written (any value). Between these two events, the software can manage the state and the mode of the core in order to re-enable the IRQ line and to allow an interrupt with a higher priority.

When an interrupt is managed by the core, R14_irq and SPSR_irq are automatically overwritten without being saved: it is mandatory to save these registers before re-enabling the IRQ line and to restore them before exiting the interrupt management routine. Moreover, if the interrupt treatment performs function calls (Branch with Link), R14_irq is used. In this case, IRQ can not be re-enabled while the core is in IRQ mode. It is mandatory to first change the mode of the core. In order to keep all exceptions available, the SYSTEM mode must be used. Therefore, the stack used during the interrupt execution is the same as that used out of the interrupt. This must be taken into account in the sizing of the SYSTEM/USER stack.

This is performed as follows:

1. Save R14_irq and SPSR_irq in the IRQ stack (current)
2. Set the mode bits in CPSR with the SYSTEM value (0b11111)
3. Re-enable IRQ by clearing bit I in CPSR
4. Execute the actions related to the interrupt
5. Disable IRQ by clearing bit I in CPSR
6. Set the mode bits in CPSR with the USER value (0b10000)
7. Restore R14_irq and SPSR_irq from the IRQ stack

Note that this sequence is automatically preceded by a read of AIC_IVR (see “Auto-Vectoring” on page 2) and must be followed by a write in AIC_EOICR before exiting from the interrupt.

AT91M40400 Implementation

The implementation of the auto-vectoring is done by initializing the IRQ vector at address 0x0000018 with the following instruction:

```
ldr pc,[pc,#-0xF20]
```

The implementation of the prioritization is described in the file “irq.mac” which is included in the folder “at91_include” of the AT91 library (examples of use can be found in the files “irq_*.s” of the folder “at91_lib”). This file includes 2 macros:

- IRQ_ENTRY which saves the registers and switches to SYSTEM mode
- IRQ_EXIT which switches back to IRQ mode, restores the registers and exits from the interrupt after acknowledging the current interrupt by writing in AIC_EOICR.

The standard format of an interrupt handler is:

1. Auto-Vectoring: instruction “ldr pc,[pc,#-0xF20]”
2. Validate the nested interrupts: macro-definition IRQ_ENTRY
3. Perform interrupt treatment (e.g. for the USART transmitter, write a new byte in the US_THR)
4. Disable the nested interrupts: macro-definition IRQ_EXIT

IRQ_ENTRY Macro Definition

```
        MACRO
        IRQ_ENTRY $reg
    ;- Adjust and save LR of current mode in current stack
            sub             r14, r14, #4
            stmfd           sp!, {r14}
    ;- Save SPSR and r0 in current stack
            mrs             r14, SPSR
            stmfd           sp!, {r0, r14}
    ;- Read Modify Write the CPSR to Enable the Core Interrupt
    ;- and Switch in SYS Mode ( same LR and stack than USR Mode )
            mrs             r14, CPSR
            bic             r14, r14, #I_BIT
            orr             r14, r14, #ARM_MODE_SYS
            msr             CPSR, r14
    ;- Save used registers and LR_usr in the System/User Stack
            stmfd           sp!, {r1-r3, $reg, r12, r14}
        MEND
```

The parameter “$reg” allow the list of the registers used by the interrupt treatment to be pushed on the SYSTEM/USER stack by using the instruction which pushes R14_User. This list must be the same for the IRQ_EXIT call.

Note that in this application note, all registers defined as “scratched” by APCS (r0, r1, r2, r3, r12) are saved by IRQ_ENTRY and restored by IRQ_EXIT.

IRQ_EXIT Macro Definition

```
        MACRO
        IRQ_EXIT    $reg,
    ;- Restore used registers and LR_usr from the System/User Stack
            ldmfd           sp!, {r1-r3, $reg, r12, r14}
    ;- Read Modify Write the CPSR to disable interrupts
    ;- and to go back in the mode corresponding to the exception
            mrs             r0, CPSR
            bic             r0, r0, #ARM_MODE_SYS
            orr             r0, r0, #I_BIT:OR:ARM_MODE_IRQ
            msr             CPSR, r0

    ;- Mark the End of Interrupt on the interrupt controller
            ldr             r0, = AIC_BASE
            str             r0, [r0, #AIC_EOICR]
    ;- Restore SPSR_irq and r0 from the IRQ stack
            ldmfd           sp!, {r0, r14}
            msr             SPSR, r14
    ;- Restore ajusted LR_irq from IRQ stack directly in the PC
            ldmfd           sp!, {pc}^
        MEND
```

The IRQ Stack

The IRQ stack pointer (R13_irq) must be initialized at the top (upper address) of a reserved space. The size needed for this stack is 12 bytes (3 words for registers r0, r14 and SPSR) per level used in the application. If all levels are used, the stack space must be 96 bytes.

Constants

The constants used in this application note are as follows:

```
ARM_MODE_SYS        EQU     0x1F
I_BIT               EQU     0x80
AIC_BASE            EQU     0xFFFFF000
AIC_EOICR           EQU     0x0130
```