Disabling Interrupts at Processor Level

AT91
ARM Thumb®
Microcontrollers

Application Note

Background

The AT91 is based on the ARM7TDMI™ microcontroller core.

This microcontroller core implements two physically independent sources of interrupt:

- FIQ - Fast Interrupt
- IRQ - Normal Interrupt

Both of these interrupts can be enabled/disabled at core level, by clearing/setting the corresponding bit in the CPSR (Current Processor Status Register):

- FIQ - bit 6
  (clear = enabled, set = disabled)
- IRQ - bit 7
  (clear = enabled, set = disabled)

On the AT91, all interrupts are managed by the Advanced Interrupt Controller (AIC) which asserts the internal IRQ, or FIQ, according to the interrupt source.

When the microcontroller core detects an IRQ or FIQ, and if the corresponding bit in CPSR is cleared, it ends the instruction currently in progress, then fetches the corresponding interrupt vector at address 0x0018 (IRQ) or 0x001C (FIQ).

Rev. 1156A–08/98

Issue

When the application must not be interrupted, interrupts must be disabled at core level. This can be done using the following code:

```
mrs   r0, CPSR
orr   r0, r0, #0x80  (or 0x40 for FIQ)
msr   CPSR, r0
```

While the processor is executing the 'msr CPSR, r0' instruction, the active interrupts are disabled only on the next clock cycle. If an IRQ or FIQ interrupt occurs during the execution of this instruction the bit IRQ or FIQ is set both in the CPSR and in the SPSR (Saved Program Status Register) and *the processor enters the interrupt handler routine.*

The application developer must take care not to modify the IRQ or FIQ bit in the SPSR before exiting from the interrupt service routine. If the interrupt handler does something like:

```
mrs   r0, SPSR
bic   r0, r0, #0x80  (or 0x40 for FIQ)
msr   SPSR, r0
```

The IRQ or FIQ interrupt is re-enabled at the exit and the processor resumes execution after the 'msr CPSR, r0' instruction. *The interrupt is enabled while the application routine assumes that it is not.*

Workarounds

There are 3 different ways to avoid the situation described above.

1. The first is to always take care, during the interrupt routine, to restore the I and F bits in the register SPSR before exiting.

   Advantage: additional code is not necessary.

   Inconvenience: the interrupt treatments must strictly follow this rule, which is not fully secure.

2. The second is to disable the interrupt with the following sequence:

```
DisIRQ
      ldr         r1,#0x80
      b           DisInt
DisFIQ
      ldr         r1,#0x40
DisInt
      mrs         r0,CPSR
      orr         r0,r0,r1
      msr         CPSR,r0
      mrs         r0,CPSR
      ands        r0,r0,r1
      beq         DisInt
```

   Advantage: more secure and no time needed during the interrupt routine.

   Inconvenience: more time is needed to disable interrupt at the task level.

3. The third is to check the state of the core at the beginning of the interrupt, and to exit if it is disabled.

   The sequence to manage the IRQ is:

```
ManageIRQ
      sub         lr, lr, #4        ; Adjust LRirq
      stmfd       sp!, {lr}         ; save LRirq
      mrs         r14, SPSR         ; Abort if IRQ disabled
      ands        r14, r14, #0x80   ; (I_BIT = 1)
      ldmnefd     sp!, {pc}^
```

   Insert here the IRQ management

```
      ldmfd       sp!, {pc}^
```

   The sequence to manage the FIQ is :

```
ManageFIQ
      sub         lr, lr, #4        ; Adjust LRfiq
      stmfd       sp!, {lr}         ; save LRfiq
      mrs         r14, SPSR         ; Abort if FIQ disabled
      ands        r14, r14, #0x40   ; (F_BIT = 1)
      ldmnefd     sp!, {pc}^
```

   Insert here the FIQ management

```
      ldmfd       sp!, {pc}^
```

   Advantage: fully secure if each interrupt treatment includes this sequence.

   Inconvenience: three fetch cycles are needed to manage the interrupt.