Application Note

AN2554/D
Rev. 0, 7/2003

Clearing and Disabling Interrupt Flags

By Gordon Borland

Introduction

A common feature of all Motorola micro-controllers is the use of flag bits to latch interrupts by the MCU.

Examples of modules, which utilise this feature, include the MSCAN, certain implementations of Key Wake-Up, and certain implementations of PLL.

An interrupt is pending whilst its respective flag bit is set and the interrupt is enabled. The interrupt service routine must reset the flag in order to handshake the interrupt. Typically this is achieved by writing a 1 to the corresponding flag bit.

Care should be taken when using bit manipulation instructions (BSET) to clear the interrupt flag, as this can cause interrupt requests to become lost and not get serviced.

The recommended way to clear interrupt flags is to use load and store instructions with bit masks.

Motorola micro-controllers also feature an I mask bit in the Condition Code Register (CCR). The I mask bit enables and disables maskable interrupt sources.

Disabling a specific interrupt source without previously setting the I mask bit in the CCR can result in spurious interrupts which cause the micro-controller to take the SWI vector instead of the expected interrupt vector.

It is recommended to set the I mask bit before disabling a maskable interrupt source to avoid spurious interrupts.

Clearing Interrupts

Using a bit manipulation instruction to clear an interrupt flag by writing a '1' can result in other interrupt flags being inadvertently cleared.

The BSET instruction is a read, modify, write command. That is it reads the contents of a memory location, modifies the contents, and then writes the modified value back to the memory location.

In the case of BSET it reads the memory contents, performs a logical OR with a user defined mask byte and then writes the contents back to the memory location.

BSET Operation: **(Memory Contents) + (Mask) $\Rightarrow$ (Memory Location)**

Thus it is possible that if two or more interrupt flag bits are set and a BSET command is used to clear one of the flag bits, all of the interrupt flags which are set will be cleared.

Using a MSCAN Receiver Flag Register (CRFLG) as an example, we see that it consists of 8 interrupts flags.

		Bit 7	6	5	4	3	2	1	Bit 0
CRFLG	R	WUPIF	RWRNIF	TWRNIF	RERRIF	TERRIF	BOFFIF	OVRIF	RXF
$0104	W								
RESET		0	0	0	0	0	0	0	0

Writing a '1' to the corresponding bit position clears each of these flags.

For an interrupt to become pending both the interrupt flag and the interrupt enable bit must be set. i.e.

INTERRUPT = FLAG & ENABLE_BIT

If we assume that two interrupt flags are set (OVRIF and RXF) then the CRFLG register will contain the value $03.

In order to handle the RXF interrupt, the interrupt service routine (ISR) should clear the appropriate flag bit by writing a '1' to bit position 0 of the CRFLG register.

Using a BSET instruction,

BSET CRFLG, $01

Becomes:

($03) + ($01) $\Rightarrow$ CRFLG

As $03 + $01 equals $03. $03 will be written to the CRFLG register clearing the RXF flag and causing the OVRIF flag to be cleared also. The OVRIF interrupt request has been cleared unintentionally and will go unnoticed.

Even if only one interrupt flag bit is set, it is still not appropriate to use the BSET command to clear the flag, as this could still cause interrupt requests to go unnoticed.

Again consider clearing the RXF flag. If we assume that the OVRIF flag is set after the RXF interrupt is recognized, but before the BSET instruction is executed. Using BSET to clear the RXF flag will again cause the OVRIF interrupt to go unnoticed in the same manner as before.

In order to avoid this happening users should use load and store instructions in conjunction with an appropriate bit mask.

Using load and store instructions to clear the RXF flag from our initial example, we get:

LDAA #$01

STAA CRFLG

Or

MOVB #$01,CRFLG

NOTE: *(MOVB is an HC12 instruction. HC08 devices should use the MOV command instead)*

After these instructions have been executed the CRFLG register will contain $02. The OVRIF flag will not be affected.

Alternatively it is possible to use the BCLR command to clear RXF.

The BCLR instruction reads the register contents, ANDs it with the inverse of a user supplied mask, the writes the result back to the register.

Thus to clear the RXF flag using the BCLR instruction we get:

BCLR CRFLG, $FE

Which becomes:

($03) · ($01) $\Rightarrow$ CRFLG

As $03 · $01 equals $01. $01 will be written to the CRFLG register clearing the RXF flag and leaving the OVRIF flag status unchanged. However, the programmer has to remember to use the complement of the normal bit mask for this instruction to have the desired effect in this particular case.

Disabling Interrupts

Disabling a maskable interrupt without setting the I mask in the CCR can cause the micro-controller to fetch a different interrupt vector than the expected vector.

A spurious interrupt occurs when the micro-controller starts interrupt processing due to an asserted interrupt, but when the interrupt vector is fetched, the interrupt has gone. In this case, the microcontroller will fetch the SWI vector.

This can occur when an interrupt source is disabled.

For an interrupt to occur both the interrupt flag and the interrupt enable bit must be set. i.e.

INTERRUPT = FLAG & ENABLE_BIT

Using a Keyboard Interrupt Enable Register (KBIER) as an example, we see that it consists of 5 interrupt enable bits.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	KBIE4	KBIE3	KBIE2	KBIE1	KBIE0
Write:	▨	▨	▨					
Reset:	0	0	0	0	0	0	0	0

In order to disable the KBIE4 interrupt after it has been previously enabled, bit 4 needs to be cleared.

```
LDAA KBIER

ANDA #$EF

STAA KBIER
```

Or

```
BCLR KBIER, $10
```

If the KBIE4 interrupt is asserted whilst the STAA (or BCLR) command is being executed, interrupt processing will start at the completion of the STAA (or

BCLR) command. In the meantime however the interrupt source has been disabled due to KBIE4 getting cleared. As a result a spurious interrupt will occur and the SWI vector will be fetched.

To avoid the risk of this happening, users should set the I mask bit before disabling the interrupt.

Using our previous example, the correct code should be

```
SEI
LDAA KBIER
ANDA #$EF
STAA KBIER
CLI
```

Or

```
SEI
BCLR KBIER, $10
CLI
```

Setting the I mask bit will prevent any maskable interrupt processing until after the interrupt source has been disabled and the I mask bit cleared.

NOTE: *There is an intentional one cycle delay in the clearing mechanism of the CLI instruction, which guarantees that the instruction after the CLI command will always be executed, even if an interrupt is pending. This is useful for entering STOP mode for example:*

```
SEI
.
.    ;Disable IRQs not required for STOP exit
.
CLI
STOP    ;This is always executed
```

[redacted]

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor

AN2554/D
Rev. 0
7/2003