Clearing Module Interrupt Flags in LPW SoC Devices

By Kristoffer Flores

Keywords

- *Lost module interrupt flags*
- *Write-0 register bits*
- *Read-Modify-Write*
- *Interrupt Handling*
- *CC1110Fx*
- *CC1111Fx*
- *CC2510Fx*
- *CC2511Fx*
- *CC2430*
- *CC2431*
- *CC2530*

1 Introduction

The traditional software method for clearing module interrupt flags in TI's 8051-based low power wireless System-on-Chip (LPW SoC) devices can unintentionally mask other interrupt flags. Depending on the peripheral and the application, these masked or missing module interrupts can lead to unexpected system behavior. For example, in the Direct Memory Access (DMA) controller, a missing interrupt flag can freeze a channel until a system restart reinitializes the controller.

Lost interrupts can result from read-modify-write (RMW) operations to clear older interrupt flags in the same module flag register. Examples of these traditional RMW operations to clear interrupt flags can be found in datasheets and sample code. This design note describes how module interrupt flags can be lost when using these operations and recommends alternative code that takes advantage of the Write-0 design of the module interrupt flag bits.

Although this note assumes a working knowledge of interrupt programming, a brief overview of interrupt flags prefaces the issue of lost module interrupt flags and its avoidance.

Table of Contents

2 Abbreviations

CPU	Central Processing Unit
DMA	Direct Memory Access
ISR	Interrupt Service Routine
RMW	Read Modify Write
SoC	System On Chip

3 CPU and Module Interrupt Flags

For most modules found in TI's LPW SoC devices, the hardware asserts, or sets to 1, two flags when an interrupt condition occurs: a CPU interrupt flag and a more descriptive flag in a module-specific register. The CPU interrupt flag tells the CPU which module has an interrupt to service while the flag in the module register gives an indication of what event occurred.

For example, when using the DMA controller and a DMA channel reaches its transfer count, both the DMAIF bit in the IRCON CPU flag register and a bit in the DMAIRQ register are asserted. DMAIF tells the CPU to call the DMA's interrupt service routine (ISR). The bit asserted in DMAIRQ corresponds to the channel that triggered the transfer complete interrupt, i.e. channel 0's flag is 0x01, channel 1's flag is 0x02, and so on. The ISR reads the value of DMAIRQ to determine which DMA channel reached its transfer count. The SoC datasheets ([1], [2], [3], and [4]) each contain a list of all the CPU interrupt flags as well as the module-specific interrupt flags and their descriptions.

Unless the flags are cleared by hardware, both the CPU interrupt flag and the flag in the module interrupt register must be cleared, or set to 0, during a module's ISR. If an ISR completes without clearing the CPU interrupt flag, the CPU will call the ISR again after returning to the interrupted process, which is usually the main program loop. Clearing a module interrupt flag prevents the ISR from interpreting old flags as new interrupts on the next ISR call.

4 Read-modify-write Operations and Lost Module Interrupt Flags

Depending on the code used to clear a module interrupt flag, it is possible to accidentally mask an incoming interrupt. Consider the C code below for a typical DMA ISR for a system using two DMA channels for single block transfers. The ISR uses DMAIRQ to determine which channel has an interrupt, clears the flag in DMAIRQ, and rearms the channel:

```
1  #pragma vector = DMA_VECTOR
2  __interrupt void dma_irq (void)
3  {
4      EA = 0;              // Disable all other interrupts
5      DMAIF = 0;           // Clear the main CPU DMA interrupt flag
6
7      if (DMAIRQ & 0x01)   // Check if DMA ch. 0 transfer complete
8      {
9        DMAIRQ &= ~0x01;   // Clear channel 0 interrupt flag
10
11       // Any other desired actions before rearming the channel.
12       // e.g. increment counter, toggle LED, set case variable
13       // for program loop
14
15       DMAARM |= 0x01;    // Rearm the channel so it can be
16                          // triggered again
17     }
18     else if (DMAIRQ & 0x02) // Check if ch. 1 transfer complete
19     {
20       DMAIRQ &= ~0x02;   // Clear channel 1 interrupt flag
21
22       // Any other desired actions before rearming the channel.
23       // e.g. increment counter, toggle LED, set case variable
24       // for program loop
25
26       DMAARM |= 0x02;    // Rearm the channel so it can be
27                          // triggered again
28     }
29
30     EA = 1;   // Re-enable interrupts
31 }
```

The line `DMAIF = 0` performs a bit-clear on the DMA controller's CPU interrupt flag in the IRCON register. A bit-clear has no effect on the other bits in the same register. This bit-clear operation is possible because IRCON is bit-addressable. In the 8051-based LPW SoC's, all of the CPU interrupt flag registers are bit-addressable except for the S1CON register. S1CON contains two identical copies of the general RF interrupt flag RFIF, and the other bits of the register are unused.

DMAIRQ, as with most module registers, is not bit-addressable. The lines `DMAIRQ &= ~0x01` and `DMAIRQ &= ~0x02` are the statements to clear the module interrupt flags. A general form of these statements is `RegisterName &= ~FlagBit`, an instruction intended to clear FlagBit while leaving the remaining bits in RegisterName unaffected. The 8051 CPU executes this instruction as a bit-wise AND between a direct data register, RegisterName, and an immediate data byte, ~FlagBit, and stores the result back into RegisterName. The compiler-generated assembly instruction for the statement is `ANL RegisterName, #(~FlagBit)` and requires 4 clock cycles.

The possibility of losing interrupt flags arises because this instruction requires a read-modify-write operation. The CPU reads the value from RegisterName, the necessary operations are performed with the read value, and the computed value is written back to RegisterName. Suppose DMAIRQ is initially 0x01 due to a channel 0 interrupt and the example DMA ISR has been called. The ISR finds the channel 0 flag in DMAIRQ and executes `DMAIRQ &= ~0x01` to clear the flag. In executing this instruction, the CPU first reads the value of DMAIRQ, or 0x01. A bit-wise AND is then performed between this read value and 0xFE (i.e. ~0x01), resulting in 0x00. Finally, the CPU writes 0x00 back to the DMAIRQ register.

Suppose another channel, DMA channel 1, reaches its transfer count in the time between the read and write in the RMW operation above. DMAIF is reasserted and the channel 1 interrupt bit in DMAIRQ is asserted by hardware so that the physical value of DMAIRQ is 0x03. However, the CPU is performing operations on the previously read value of DMAIRQ, 0x01. The CPU calculates `0x01 & 0xFE = 0x00`, and writes 0x00 to DMAIRQ, clearing the interrupt flag for DMA channel 1. The ISR continues and completes with no apparent errors. Figure 1 illustrates this sequence by showing the operations of the CPU and the physical contents of the register.

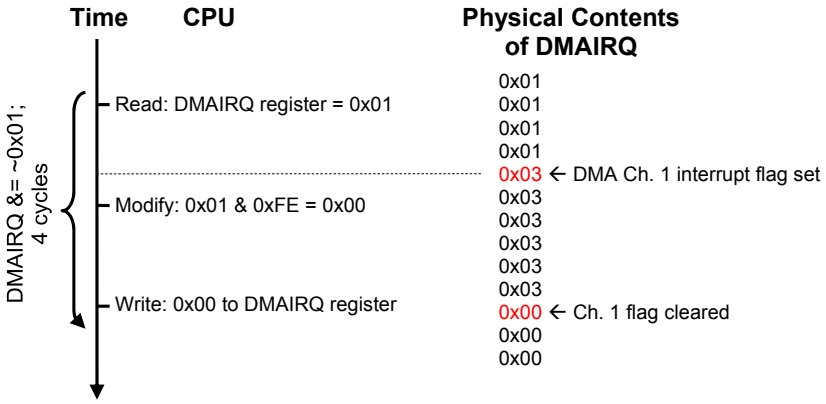

Figure 1. Timing diagram showing a missed interrupt flag on DMA channel 1 due to the instruction, `DMAIRQ &= ~0x01`, to clear the DMA channel 0 interrupt flag.

Continuing the above scenario, the CPU calls the DMA ISR again (since DMAIF = 1) immediately after the previous ISR execution completes. This time, DMAIRQ is 0x00 and both if-statements fail. DMA channel 1 is not rearmed and will not process new transfers when triggered. If channel 1 is not rearmed anywhere else in the program code, the channel remains idle until the DMA initialization code (which arms the channels) executes again due to a system reset or fresh power-up. In single transfer modes, a DMA channel must be rearmed after each completed transfer count before it can be triggered again to execute other

transfers. Refer to the “DMA Controller” section in the SoC datasheets ([1], [2], [3], and [4]) for more detailed information about the DMA controller.

Imagine another application that relies on Port 0 input interrupts: five logic-level signals from sensors and a power supply. Like the DMA, Port 0 has a CPU interrupt flag, P0IF, and a module flag register, P0IFG, where each bit of P0IFG corresponds to an interrupt flag for a pin. While P0IF is a flag bit in the bit-addressable IRCON register, the module register P0IFG is not bit-addressable. If the timing between interrupt-triggering edges on the port pins fall just right, clearing an interrupt flag bit using `P0IFG &= ~FlagBit` can lead to lost interrupts and erroneous system operation.

5 Write-0 Registers

Fortunately, a quick modification to existing code can prevent missing interrupt flags. In TI’s 8051-based LPW SoC devices, all module interrupt flags are Write-0 bits, or W0 as listed in the datasheets. Write-0 bits cannot be set to 1 by software; that is, writing a 1 into a Write-0 bit has no effect on the bit’s value. However, a Write-0 bit that has been set to 1 by hardware due to an interrupt can be cleared by writing a 0 to the bit. Figure 2 shows a simplified model of a Write-0 bit consisting of a D flip-flop with a feedback AND gate between the data to be written and the current bit value, Q. An interrupt event changes the bit value to 1 via the flip-flop’s Preset pin. Writing a 0 to the input AND gate will clear the bit value to 0. Writing a 1 to the input AND gate will maintain whatever value is already in the flip-flop.

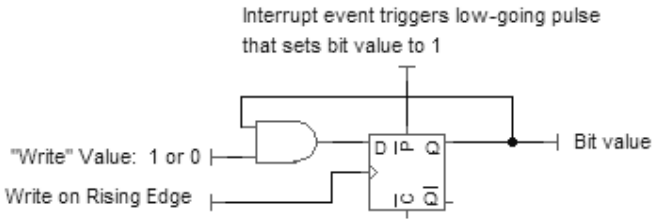

Figure 2. Simplified Write-0 Bit Model

The solution to prevent missing interrupts takes advantage of the Write-0 bit behavior. For the DMA example, instead of using `DMAIRQ &= ~0x01` and `DMAIRQ &= ~0x02`, the ISR should use `DMAIRQ = ~0x01` and `DMAIRQ = ~0x02`, respectively. These instructions will clear the desired bit with a 0 and write 1s to the other bits to preserve the physical register values. In a general form, for a module interrupt flag register with all Write-0 bits, replace `RegisterName &= ~FlagBit` with `RegisterName = ~FlagBit`.

The compiler-generated assembly instruction for the new statement is a memory move operation: `MOV RegisterName, #(~FlagBit)`. Though the operation still requires 3 clock cycles, it carries no risk of inadvertently clearing new interrupt flag bits.

Some Write-0 interrupt flag bits share a register with non-Write-0 bits. For example, the upper four bits of the T1CTL register are Write-0 interrupt flags for Timer 1 while the lower four bits of the register control the timer settings. To clear a flag, use the following code:

```
T1CTL = (~FlagBit & 0xF0) | (T1CTL & 0x0F);
```

This will preserve the values for the lower four bits, write a 0 to the flag bit to be cleared, and write 1s to the remaining flag bits. Again, the 1s written to the other Write-0 flag bits pass the existing values in the T1CTL register. This is a safer approach than using `T1CTL &= ~FlagBit`.

6 Summary

To avoid losing module interrupt flags, take advantage of the Write-0 design of the module interrupt flag bits. Replace instructions that require read-modify-write operations with instructions that write 1s to the flags not being cleared. Often, this is as simple as removing a single symbol, `&`, to change the read-modify-write operation `RegisterName &= ~FlagBit` to `RegisterName = ~FlagBit`.

7 References

[1] CC1110Fx/CC1111Fx Datasheet (SWRS033)

[2] CC2510Fx/CC2511Fx Datasheet (SWRS055)

[3] CC2430 Datasheet (SWRS036)

[4] CC253x User Guide (SWRU191)

8 General Information

8.1 Document History

Revision	Date	Description/Changes
SWRA303	2009.08.04	Initial release.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265