ARM: Cortex-M3/M4 Interrupts Happening Twice?

Information in this knowledgebase article applies to:

- Cortex-M3 and Cortex-M4 Devices

SYMPTOM

Cortex-M3 and Cortex-M4 interrupts appear to be triggering twice.

CAUSE

This may happen with devices:

- That add an external, system-level write buffer in their Cortex-M3 or Cortex-M4 design, AND
- The ISR code exits immediately after a write to clear the interrupt.

In this situation, the write to clear the interrupt may not complete before interrupt triggers a second time.

REASON

For the Cortex-M3 and Cortex-M4 cores, writes (STR, STMIA or PUSH) to memory are internally buffered. The Harvard architecture allows the MCU to fetch and execute instructions without waiting for data writes to memory to complete. The Cortex-M cores are aware of the internal buffer and prevent subsequent interrupts until the internal buffer empties and the write completes.

Sometimes vendors incorporate an additional external, system-level write buffer in their Cortex-M3 and Cortex-M4 designs for better performance. But unfortunately, the core is not aware of this external write buffer and cannot access it's status. For these externally-buffered devices, if an ISR exits immediately after clearing the interrupt register, a second interrupt could trigger again before the write to clear the interrupt completes.

For example, this ISR exits immediately after clearing the timer interrupt. Without the external buffer implementation, this code would work as expected. However, on a device with an external, system-level write buffer, this code could cause this "double IRQ" condition:

```
void Timer_IRQHandler (void) {
  Timeout_counter++;                /* Increment timeout counter */
  PortD->PTOR |= 1<<0;              /* Toggle output on port D0  */
  Timer->MSR |= TIMER_MASK;         /* Clear timer interrupt     */
}
```

RESOLUTION

Using the DSB instruction or __dsb(0) intrinsic before exiting will force a wait for the internal write buffer to empty, but that instruction cannot test the status of an optional system-level write buffer if there happens to be one. To make sure the peripheral interrupt register gets set properly, just perform another memory write before exiting the ISR.

Given the example above, one way to do this is by incrementing the timeout counter AFTER clearing the interrupt:

```
void Timer_IRQHandler (void) {
  PortD->PTOR |= 1<<0;              /* Toggle output on port D0         */
  Timer->MSR |= TIMER_MASK;         /* Clear timer interrupt            */
  Timeout_counter++;                /* Count timeout & insure IRQ clear */
}
```

But any type of memory write will accomplish this. The **Timeout_counter++** works because it performs a read-modify-write to memory. If you can't move an instruction like in the above example, just add harmless code that performs a read-write like this:

```
void Timer_IRQHandler (void) {
  Timeout_counter++;                /* Increment timeout counter */
  PortD->PTOR |= 1<<0;              /* Toggle output on port D0  */
  Timer->MSR |= TIMER_MASK;         /* Clear timer interrupt     */
  PortD->PTOR = PortD->PTOR;        /* Insure IRQ clear          */
}
```

Last Reviewed: Wednesday, June 14, 2017