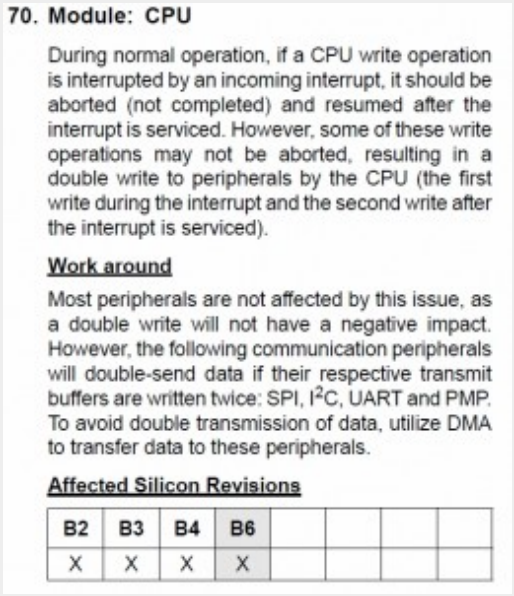
Magnus Errata Part 2

Posted on **April 1, 2012**

In the first part of this series, Magnus Errata Part 1, we looked at the problems caused by reading data from flash memory while interrupts are in use. In this second part we examine the more serious issue of double writes to sensitive I/O registers. Here is the issue as presented in the errata:



70. Module: CPU

During normal operation, if a CPU write operation is interrupted by an incoming interrupt, it should be aborted (not completed) and resumed after the interrupt is serviced. However, some of these write operations may not be aborted, resulting in a double write to peripherals by the CPU (the first write during the interrupt and the second write after the interrupt is serviced).

Work around

Most peripherals are not affected by this issue, as a double write will not have a negative impact. However, the following communication peripherals will double-send data if their respective transmit buffers are written twice: SPI, I^2C, UART and PMP. To avoid double transmission of data, utilize DMA to transfer data to these peripherals.

Affected Silicon Revisions

B2	B3	B4	B6				
X	X	X	X				

— Double Write Issue

In this scenario, a write operation is duplicated when it is interrupted at just the wrong moment. Now, in most cases, this poses no problem. A = 3; would simply assign 3 to A twice. However, some I/O registers are sensitive to multiple writes. The data registers for the SPI, UART, I2C, PMP, and GPIO Toggle will not perform correctly if double write occur.

The answer given, was to use the DMA controller to do the writing to these registers. This is a most draconian and unworkable “fix” for the problem. It replaces a simple write to a register with

an enormous amount of code to set up the DMA to transfer the data byte. And further since DMA controllers don't like to be shared, a whole extra layer of mutual exclusion logic is required.

To make matters worse, I just don't trust this report! If this issue were as bad as the docs make it out to be, you would expect a lot of problems. I mean the issue is I/O and interrupts! Where is the out cry? Why does my own code work without any trace of error? Here's what I saw in the forums; rpgZ wrote:

"I am running a project the has 4 SPIs clocking in framed mode at 2.048Mbit and a bitbashed uart (No USART pins left if all SPI in use) running at 57600 baud. The bitbashing runs interrupts at 172800 interrupts per second and the SPIs shift out 128 bytes at a time 8000 times persecond. The BB UART has highest priorty so it is interrupting the SPI interrupt continuously. Also have not had any errors so far.. Got my fingers crossed. "

Marc Coussement wrote

"Problem solved !!!

Confirmed by Microchip

Read errata Issue 44

First read the errata and look [at the] work around.

Conclusion: ..... stop using PIC32"

Mikael wrote:

"No, it's a mayor issue, CPU double writes (by mistake) to peripheral, think about it.. uart, spi i2c....

And the workaround is really ugly."

bsder wrote:

*"Do I have to disable *all* interrupts? Can they just be masked out? At peripheral level or global level? What about software interrupts? (always occur at a particular pipeline stage and may dodge the bullet) What about timer-based interrupts? (probably always occur relative to a clock edge and probably *don't* dodge the bullet–they can float through different pipeline stages).*

If I put on my microprocessor designer hat, it's probably that the peripheral commits a pipeline stage before the MIPS32 actually commits. If that's the case, simply masking off externally driven interrupts (either directly per unit or globally) should be sufficient and won't require a massive draining of the pipeline.

*This severe an errata needs a lot more *public* information and precision."*

and

“[snark] So … I can shut off interrupts for a cycle or two, or I can completely re-architect my application to use a poorly documented peripheral which has driven some quite competent people on this board absolutely mad. And that’s if I’m not actually using the DMA engine. Uh, yeah, I’ll get right on that Microchip… [/snark]”

Finally the moderator added:

“Microchip understands importance of having this issue fixed and is actively working on it. Please check with your local Microchip representative on availability dates and obtaining early samples (if desired). “

I know that Microchip will track down the cause of these silicon bugs and fix them. In spite of this, I know that we can’t just bury our heads in the sand and wait for the problem to go away. It won’t. Ever! For quite some time we will be living with a PIC32 population that includes a lot of defective chips. We need useful, feasible work-arounds, not wishful insanity, because these corrective fixes will be needed for some time.

That’s why I was pleased to see this from Stampede:

“Errata #44: This issue is very similar to errata #43, but in this case interrupt has to happen at exact time as you are writing to a peripheral. Depending on the application (how often peripheral is getting written and how often interrupts are occurring in the system), user may not see this issue very frequently. It also depends if their system can handle double write case, if it happens.

While workaround maybe an “overkill” for some applications, it is a valid workaround and it will prevent double write to a peripheral.

Disabling interrupts while performing write to peripheral is another valid workaround and we will update errata to state this.

So the errata seems not to be too severe as it might seem on the first sight…

Cheers Stefan”

Well, I’m not cheering yet, but controlling interrupts around writes to sensitive registers is a lot more feasible than hauling out the nastiest peripheral on the block to copy over just one byte.

I was planning on waiting for Microchip to update its errata docs, but that has still not happened. So, I am publishing now and will update these postings as more information becomes available. As always, your comments and observations are welcomed!

Peter Camilleri (aka Squidly Jones)

SHARE THIS:

Print LinkedIn Facebook Twitter

LIKE THIS:

Loading...

This entry was posted in **Embedded Development**, **Errata** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=590]** .