If it's not tested, it doesn't work!

Bob Japenga, MicroTools Inc

Joseph: Bob, you got a call while you were out. It seems that manufacturing has reported that those new systems are occasionally locking up after power on.
Bob: Did they say how frequently?
Joseph: Well, Al said about one in every hundred units fails. But when he takes the failed unit, powers it off and on, everything works fine. Then he powers it up and down another six or seven times, and it comes up properly every time. Have you ever seen anything like this?
Bob: I might've seen that problem once or twice on the breadboards, but I attributed it to the wirewrap boards. You know how flaky those were. And the day after tomorrow is the big rollout for all the dealers and distributors. What great timing! How many systems are they shipping?
Joseph: A lot—you'd better call Al and get the problem straightened out.

Power-on intermittent failures—what a nightmare! In the roughly two decades of designing systems, I've experienced them in just a few products I've worked on, but enough to know they smell of time, money and headaches.

What's the problem in the above scenario, which is based on an actual incident? Was it software or hardware? In talking with manufacturing, we didn't learn any more. The failing units were ROM-based embedded PC-like systems using a custom BIOS, an smx executive, a Greenleaf Comm++ serial library and 600,000 bytes of custom application code. Software development was performed about 50 miles from the hardware design and manufacturing facility.

Let's take a look at how we attacked the problem. The first step in troubleshooting began with attempting to duplicate the problem. After several hours of laborious power cycling of three units, the manufacturing department located a unit that failed approximately once out of every three times. Al, the hardware engineer, obtained that machine so we could readily duplicate the problem.

Upon further investigation, we de-

Listing 1a—Before the fix

```
Turn interrupts off
Set up the serial interrupt vector
Turn interrupts on
Turn interrupts off
Set up the PIC          //** At this point interrupts are enabled
          //** The PIC is enabled but interrupts are turned off
          //** at the UART.  The fatal flaw is that the
          //** interrupt out of the UART, though disabled, is floating
Turn interrupts on
Turn interrupts off
Set up the Interrupt Enable Register (IER)
Set up the Modem Control Register (MCR)
          //** Here the OUT2 line is set and the
          //** interrupt line goes Low
Turn interrupts on
```

Listing 1b—After the fix

```
Turn interrupts off
Set up the serial interrupt vector
Turn interrupts on
Turn interrupts off
Set up the Modem Control Register
Set up the PIC
Turn interrupts on
Turn interrupts off
Set up the Interrupt Enable Register (IER)
Turn interrupts on
```

Hardware	Software
• Improper hardware initialization	• Incorrect hardware initialization
• Temperature-sensitive race conditions	• Unprotected interrupt windows
• Vibration-sensitive interconnects	• Power On System Test problems
• Component-sensitive race conditions	• Noisy or noise-susceptible power circuitry

Some design problems that can cause intermittent operation following power up

termined that the problem manifested itself as follows: the system would successfully perform its power-on self test (POST), initialize the parallel ports and then hang up. Sometimes it would display the application's startup screen and sometimes not. We noticed another quirk that arose occasionally, but an abnormality we could associate with the failure: the POST routine, which normally is a 5-sec test, would take 5-10 sec longer.

We wanted to take a closer look at the logic and so connected a logic analyzer to the unit—but the unit wouldn't fail. I was beginning to smell a hardware problem. After several hours of prodding, heating, cooling, and power cycling, we finally got the unit to fail using the logic analyzer. With it we encountered enough failures to get an adequate trace of what the system was doing and found that it was getting hung up in a serial-port interrupt service routine (ISR). We began to speculate on various causes and suspected that perhaps spurious serial interrupts were occurring during power up and the system wasn't handling them well.

We took several units that weren't failing and began power cycling them at the software facility while blasting the ports with streams of serial data. Bingo! The units that previously didn't fail were exhibiting the same symptom at power up as reported by manufacturing. We then found that the application code didn't take into account data coming in on the serial port during power up.

We fixed the code bug, and now we could power up the test units while blasting them with data on all four serial ports without any crashes. We reasoned that the system must've been getting spurious serial-port interrupts.

Where were these spurious interrupts coming from? In pursuing the source of the interrupts, Al observed that the interrupt line out of the UART (an 8250 type) was floating during the POST routine. Shortly after the application code started, the line went Low, thus allowing the UART to send an interrupt to the programmable interrupt controller (PIC). Upon reviewing the Greenleaf initialization code, we found that it was enabling PIC interrupts before it enabled the special interrupt line used on the PC implementation of the serial port (the OUT2 line on the 8250). On our UART chip, a Texas Instrument TL16C451, until the software enabled OUT2, the interrupt line connected directly to the PIC floated. The TI spec didn't state that this signal was tristated. However, because this chip is commonly used in PCs where the interrupt line to the PIC might be shared, the hardware was correct. Listing 1 shows the pseudocode fragment before and after the fix. This fragment represents several modules, explaining why it turns interrupt on and off so many times.

We had found both a window of opportunity for spurious interrupts to crash the system and the source of the interrupts! We closed the window in the application. With this latest discovery, we shot off the source of those interrupts by modifying the Greenleaf library. Because the interrupt line was dangling in the breeze during a brief period of time that interrupts were enabled, if this line was ever above the logic threshold, it would act just like blasting the unit with serial data. We suspected that certain units must be encountering some spurious interrupts at power up that caused the crash. Having convinced ourselves of the logic, we downloaded the fix to the hardware engineer, confident that the bug was fixed and went home satisfied.

Another surprise awaits

However, the next day brought the following conversation:

Al: Bob, I've got bad news. The new software didn't fix the problem. The units still fail about once every 100 power ups.
Bob: What?!? Of course the fix works. We duplicated the problem here, remember? You must not have sent the right software to manufacturing. Have you checked the date on the code?
Al: Yup—everything checks out. But it still crashes, it's still hung up in the same place.

Well, back to the drawing board. We poured over the

Rather buy it than build it?

For those who prefer not to throw together their own power-on tester, a $295 unit from MicroTools Inc (Simsbury, CT (800) 651-6170) does the job. Named the Poc-it, the desktop unit cycles a unit-under-test by providing one 120V ac output at 10A as well as a 5A relay contact to cycle dc power. It can keep track of external events with two 5V dc inputs and one optically isolated 10-30V dc input. The user interface consists of a 1 line x 16-character LCD and a 4-button keypad. Simple menus show operators how to set On and Off times of each output with 10-msec resolution in the range from 0.01 sec to 99 min 59.99 sec, reset the input counters, start/stop a test and view cycle counters for all I/O lines. You can also delay input sampling following power on to allow unstable outputs to settle.

Greenleaf serial-interrupt code. It looked clean. What could be causing the code to hang in a loop? When an interrupt was pending, the software read the UART's data register but would not clear the interrupt. What could be the cause? More interrupts?

We thought that it must be a hardware problem. Al reported that the interrupt line looked clean with the latest fix. In addition, each running of the POST routine took the same amount of time. (At least we fixed one problem: Occasionally, some spurious interrupts were coming in and tying up the system during the POST).

At last, it dawned on someone that the UART's data register wasn't being read during the ISR. An 8250 UART automatically clears an incoming interrupt when the data register that caused the interrupt is read. From the logic-analyzer trace we knew that the data register appeared to be read. Yet the interrupt wasn't being cleared. This could happen if, when the data register is read, the 8250's Divisor Latch Access Bit (DLAB) is set. The data register shares the same address with one of the registers that sets the baud rate (Divisor Latch LS). When you set the DLAB bit in the Line Control register, you can then read or write the baud rate. When the software clears the DLAB bit, the program can read or write to the data register. If an interrupt occurs when the DL bit is set and the ISR assumes that the DLAB is cleared, the pending interrupt never gets cleared because the data port is never read.

We poured over Greenleaf's code for setting the baud rate and found two more flaws in this section of code. The ISR assumed that the DL bit was already cleared rather than defensively clearing it to assure no mixups. Secondly, the routine that set the baud rate did not disable interrupts while the DLAB bit was set. Before the baud-rate function cleared the DLAB bit, a tiny window existed where a serial interrupt could occur. During the ISR that occurred in this window, the ISR read the data register; but because the DLAB was set, the source of the interrupt never got cleared. The result was a hung system.

Having found yet another window in Greenleaf's software, it wasn't with quite the same confidence that we uploaded these new fixes to Al. Would it work? It was late in the evening, and even though Al cycled a test unit running the new code a hundred or so times without any failures, we had been this route before. He needed a way to test the system overnight because the next day we planned to ship the units for the big rollout. He jerryrigged a relay to cycle the power, a digital counter to monitor

This space left intentionally blank.

the number of cycles and a function generator to drive the relay. We had kludged together such arrangements before when dealing with similar problems, but in this case, he couldn't find all the pieces to test the fix. We'd just have to take a chance that the limited test proved the fix.

As it turned out, the fix was fine. But this experience got me thinking—in all these years as a designer, I've gone about testing completely wrong. As part of my design verification—hardware, software and systems—I never power-on cycled any design thousands of times to make sure it worked on every power-up. In spite of the fact that I've encountered these problems before, I never proactively sought out such failures; they always sought me out.

I also remembered that in the past these power-on failures first arose late in the development cycle. Often the manufacturing group found them, and even worse, we got word from a customer. If a product is even remotely successful, it will be powered up thousands of times.

This space left intentionally blank.

The power-on problems were sometimes due to hardware and sometimes software (see table), but the fix was never obvious. Frequently, these problems were difficult to duplicate, and occasionally our fix was questionable. Often we had to make several attempts to solve the problem. After each fix, we had to retest the system for thousands of cycles before the problem manifested itself again.

The next day, I resolved that I would subject every system I ever designed to a power-on cycle test before I released it. I searched for a simple piece of test equipment that would enable me to do that job. One reason I never tested my designs in this way was because it was difficult to do.

When I found none, I decided that my firm should create such a product (see sidebar, "Rather buy it than build it?"). A good example of a use for this instrument is in testing a switching power supply. It would activate the AC power and sample the appropriate output voltage. Another example would be an embedded PC system with a parallel port. After the system completes its diagnostics, it might strobe the printer port to initialize the printer and port. You could sample this signal as the success input. Another example would be to test for a problem that a *PE&IN* columnist had with an FPGA-based system when toggling the power (see reference).

Meanwhile, at our firm we've developed a new motto: If it's not tested, it doesn't work. Designing complex systems has confirmed the truth of this motto for me. I'm embarrassed to admit how often obvious designs that I am tempted not to test don't work when tested. One significant area is in Power On testing. No matter whether you use a commercial product or design your own Rube Goldberg contraption, test your designs with thousands of turn ons. Seek out failures before they seek you out. **PE&IN**

Reference

Rosenthal, S, "It sometimes takes a microcontroller to boot up an FPGA," *PE&IN*, Nov 1994, pgs 75-78.

Bob Japenga is director of marketing at MicroTools Inc (Simsbury, CT (800) 651-6170), a firm focused on developing embedded systems and PC-based tools. Since 1988, the firm has developed products for the aerospace, glass and mailing-systems industries.