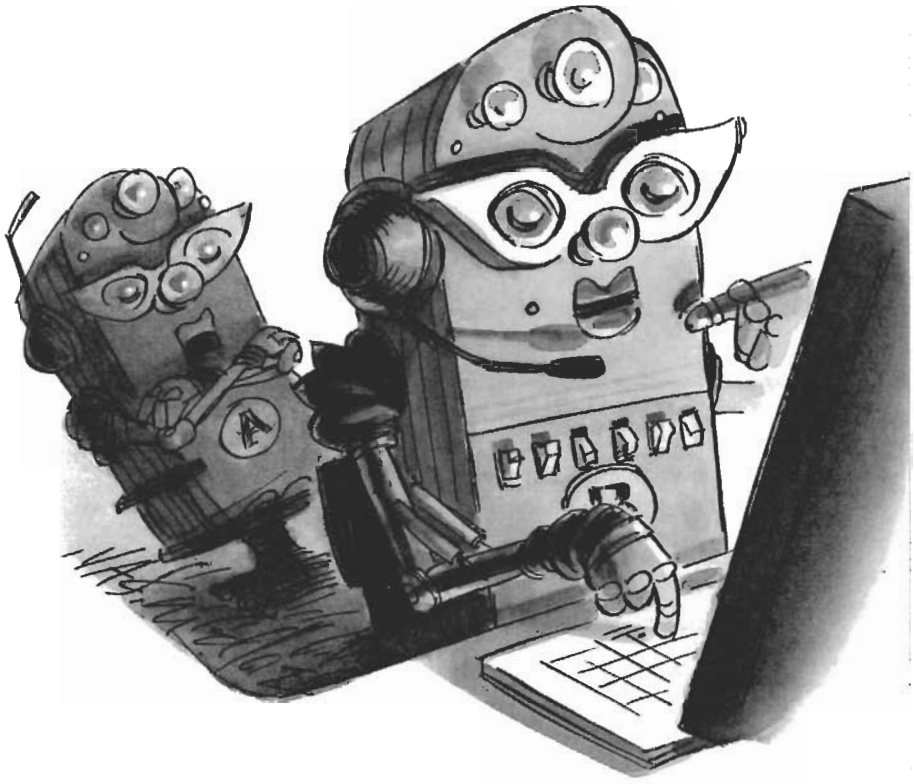
Passive part becomes aggressive



DANIEL VASCONCELLOS

In 1981, I landed a job maintaining a computerized telephone-answering service. The system comprised a 16-bit minicomputer, disk drives, a paper-tape reader, a card reader, and a Teletype. The computer had switches and lights on the front panel and external I/O cards for all of the devices. The answering service assisted doctors, ambulances, and other critical-care clients, and the computer had to stay up and running as much as possible. There were two computers, aptly named A and B. One was supposed to be online, and the other was to be ready to go online if the online system failed.

The system ran on a real-time operating system and was very impressive in its time. The computer had 32 kbytes of core memory and four 1-Mbyte disk drives, yet it could route hundreds of phone calls to 16 operators.

The B computer was the online system, and the A computer was the offline system. My colleagues informed me that, although the A system had passed all of the diagnostic tests, it could not run the online system, and the B system could not pass the diagnostic tests but ran the online system fine. I did my own testing and found this situation to be true. I wasn't content to leave it that way, though.

The A system would run the online software for 10 minutes, then become unresponsive. The previous repair people had replaced every card in the A system, and it still failed.

I had to somehow compare the systems to determine differences, and I first looked at the interrupts. I knew that the diagnostic software disabled every interrupt except for the interrupt for the device under test. I also knew that the real-time operating system had good interrupt handlers for the devices it used. The only interrupt left to worry about was a spurious interrupt. The minicomputer had 128 possible interrupts, and the real-time operating system used only 10. I wrote software to count the number of spurious interrupts and found that the offline system had hundreds of spurious interrupts per minute. The online system had thousands of spurious interrupts per minute. The B system was far noisier electrically than the A system and still worked better.

Next, I modified the operating system, a backup copy, to "think" it was online. I forced disk, terminal, and Teletype activity to load the system. After 15 minutes, it went to sleep. I ran that test three more times just to prove that the A system would fail. The next step was to run only the Teletype to see whether the computer would continue to run. The test ran this way for several hours.

I then added one device at a time. The test failed with the addition of the second disk-controller card. This result was confusing because this card passed all of the offline diagnostics. The only part that I had not repeatedly swapped was the backplane card that held the I/O cards.

I found that I could run both disk-controller cards, with some rewiring, in the same backplane. When I did, the system tests ran perfectly. I swapped out the bad backplane.

I then put the A system online and watched it for several hours. It did not fail, and none of the users knew anything had changed. I was finally able to take the B system down for some much needed rest. EDN

Craig Hermann is underemployed in Fort Myers, FL. You can reach him at chermann@att.net. Like Craig, you can share your Tales from the Cube and receive $200. Contact edn.editor@reedbusiness.com.

www.edn.com/tales

In the days of old, when engineers were bold

In the 1970s, around the time that MOS Technology introduced the 8-bit 6502 processor, I took a job as chief engineer at a company that developed, manufactured, and supplied military and industrial security-monitoring systems. Job 1 was an intermittent problem in the current model that caused the watchdog timer to reset the system. The default condition for a system reset was to secure all doors and sensors throughout the monitored areas to condition "red." Then, when a normally "green" door opened, alarms sounded. This problem was a particularly disturbing one because this equipment was monitoring sensitive areas, such as nuclear-storage facilities in which the false alarms could cause response teams to react.

The 6502-based equipment had the usual ROM, RAM, I/O, and a built-in small Seiko line printer (**Reference 1**). In those days, we tried to make one little processor do everything; why not? Popular theory at the time was "it is a hardware problem," and there was good reason to believe it. First, the design was poor and had mixed-logic families on a poorly laid out bus. It also had some ground bounce, because these were the days before multilayer boards were common. Worst of all, we were just learning about the "dirty dozen," 12 illegal instructions for the 6502 that, if executed, would cause the processor to halt. We used to call it "halt and catch fire"; some of these instructions required a power-down and backup to clear. After three months of many ECOs (engineering-change orders) to correct all the hardware deficiencies, the problem had not changed at all in nature. Now, with solid hardware, we were really scratching our heads.

There was one thing about the design that rubbed me the wrong way right from the beginning: the fact that the little Seiko line printer was tied to the NMI (nonmaskable interrupt). The 6502 controlled the printer, and the NMI monitored the synchronous pulse from the printer. Now that hardware was looking good, there was some finger-pointing over whether the problem now was hardware or software. After many hours in the lab and reports from customers, we narrowed the behavior for the intermittent failure as occurring only after the printer was winding down from printing.

After yet another long day in the lab, the NMI was still bothering me, especially now that I knew that the printer was involved. I began thinking about the 6820 PIA (peripheral-interface adapter) and the initialization sequence; then, it hit me. It was so simple: The 6820 PIA takes two instructions executed in sequence to initialize it. You cannot interrupt the instructions. I drove back to the office, and the software guys were still there. I checked, and, sure enough, when the printer was done printing, they went out and reinitialized the PIA just for grins!

For me, it was so obvious and "case closed." The short story is that I had them take out two lines of code, and we never saw the "hardware" problem again. The long story is that management and software people spent many more days testing because they just could not believe it. After much more testing, the senior programmer and I flew to one of our main problem locations and installed two new ROMs; no further problems occurred. EDN

REFERENCE

1 http://en.wikipedia.org/wiki/MOS_Technology_6502.

Lynn Smith was a consultant and hardware engineer in Silicon Valley for 30 years before recently pulling up stakes in San Jose, CA, to move to Alabama, where he is building an airpark. Like Lynn, you can share your Tales from the Cube and receive $200. Contact Maury Wright at mgwright@edn.com.

DANIEL VASCONCELLOS

BY HOWARD JOHNSON, PhD

Diagnostic testing (and tasting?)

In 1971, I enjoyed one blissful summer away from school working in a TV-repair shop. The first day on the job, my boss, Paul, explained how to work with customers at the repair counter: "Mainly, use your sense of smell. If it's just dusty, that's probably fine. If it smells like a burned resistor or burned capacitor oil, that's going to cost. If it smells like baked milk, tell me, and I'll sell 'em a new set. They already know their kid spilled milk into it, so they feel guilty. I'll have the new set in place before hubby comes home."

Wow! I had a lot to learn, and fast. Paul used all of his senses diagnosing a broken set. He could feel the 60-Hz hum from the power transformer, see the blue ionizing glow of a bad tube, hear the squeal of the 15-kHz horizontal oscillator, and smell the difference between a blown capacitor and a burned resistor. As for taste, Paul licked the terminals of a 9V battery to tell whether it was any good.

Today, human senses play a smaller role in diagnosing digital systems, but the philosophy remains the same: Use every tool at your disposal. Stick your hand into the chassis to feel the airflow. Touch the processor to see how hot it's getting. Listen to the disk chatter. Gather as much information as possible.

Compliance testing rests at the opposite end of the test spectrum. A compliance test begins with a specific list of features or performance metrics that you must verify. The test determines whether a system meets the acceptable criteria. If the system fails, the operator either discards the unit or sets it aside for rework. He makes no attempt at diagnosis.

The most interesting part of compliance testing happens before you hook up the first device under test. You must prove that your test-equipment configuration works, and works well enough to make the required measurements.

Figure 1 A test professional pays close attention to every aspect of testing.

The diagnostic process, in contrast, is a more broadly based activity. It requires a keen awareness of all aspects of the system at hand. The operator must remain ever-vigilant during testing, aware of even the tiniest clue about system behavior (**Figure 1**).

David Agans, in his terrific book, *Debugging*, articulates a coherent nine-point strategy for debugging that involves, first, deciding when to invoke the strategy (**Reference 1**).

As an everyday matter, you aren't going to walk around trailing a cart full of data scopes, recording analyzers, and EMI detectors, because that would be too much of a hassle. You need not record everything, or take careful notes, or think before you act, as long as everything goes smoothly in the lab. You instinctively know how to tweak your schematic to casually fix the obvious problems that crop up.

To help keep things easy, break the system down into small pieces—so small that there is likely only one error (or no errors) in each section. Taken in small sections, problems tend to stand out in an obvious way. That makes debugging a snap.

The time to apply your high-level-debugging expertise is when things start to look ugly. Systems that harbor multiple errors often display inexplicable, impossible, or unreliable behavior. When that situation occurs, start thinking like Paul.

Take careful notes, augment your senses with recording devices that capture the history and current state of your equipment, and spend as much time thinking and planning for the next test as you do in the lab gathering data. EDN

REFERENCE

1 Agans, David, *Debugging*, Amacom, September 2006, ISBN 978-0814474570.

MORE AT EDN.COM

Check out Howard Johnson's videos at www.edn.com/techclips.

Howard Johnson, PhD, of Signal Consulting, frequently conducts technical workshops for digital engineers at Oxford University and other sites worldwide. Visit his Web site at www.sigcon.com or e-mail him at howie03@sigcon.com.

BY CLARK ROBBINS • GS ENGINEERING

Answers can be anywhere

When I was a young engineer, my boss asked me to design a discrete-logic controller for a mechanical/hydraulic apparatus. After weeks of work using CMOS logic and power transistors, I reanalyzed the design, and everything seemed correct. I then commissioned prototype PCBs (printed-circuit boards), and, by the time they were ready, I had enough parts to build some samples. I used switches and lights to simulate the apparatus. Again, everything worked OK. When I connected my controller to the first apparatus, everything still worked fine. Even after the engineers had tested everything using my controller, there were no major problems.

The team involved with the mechanical/hydraulic-apparatus design discussed how it would handle production. Because my work with the controller was essentially done, I volunteered to design an automated-test system. Although I tried to hide the anticipation of designing my first microcontroller system, I was elated when my boss gave me the task and got management to sufficiently fund it.

This part of the project turned out to be significantly more complicated than the original controller design, involving hardware design and software programming in assembly language. The test system would have to be able to power the complete apparatus. It would also have to cycle through all the modes, monitor the hydraulic solenoids and wiring to make sure that there were no shorts or opens, and simulate and control the same outputs as the apparatus controller would. Finally, it would need to accomplish all of these tasks under the control of a software program running the microcontroller chip.

This system was new and had bugs and idiosyncrasies. And though I eventually got it running, I was now behind schedule. After completing the automated-test-system-hardware design, I was working hard at learning the language for this new microcontroller. I don't remember my professor's saying that not all microcontrollers have the same assembly language. I figured that they would all be similar to the one I had learned about in class. Big mistake. The instructions were significantly different from what I had learned. I was beginning to worry that I wouldn't make the schedule.

The programming and debugging were not going well. In addition, I was having hardware problems. I was now working 12 hours a day, but, just when things were starting to improve, the black clouds came overhead.

Reconnaissance suggested to me that my system problem could be an error in the software program. I found and fixed some bugs, but the problem was still there. So, I spent hours debugging the system. Things were again looking bad for the schedule, despite the fact that I was now working 14 hours a day. For three or four days, I worked on nothing else yet made zero progress. I was so focused on one area that I was not looking anywhere else.

Then, I got really lucky; a student working for me suggested that I look at one of the PCBs. Though I disagreed with his suggestion, I decided to pursue it. (I did not want to discourage him from sharing future ideas.)

After probing some of the circuits, I found a problem with a Schmitt-trigger IC. Once I replaced the IC, the automated-test system worked the way it was supposed to.

Throughout my career, I have achieved some measure of success by developing a reputation for being able to solve problems. The lessons I learned during this project became ingrained in me and became the cornerstones of this success. I try never to dismiss a suggestion from anyone, regardless of the person's job, background, or education. And I try never to focus so narrowly that I miss the big picture, where a problem's source might reside. EDN

Consultant Clark Robbins is a software-application engineer. You can share your Tales from the Cube and receive $200. Contact Maury Wright at mgwright@edn.com.

DANIEL VASCONCELLOS