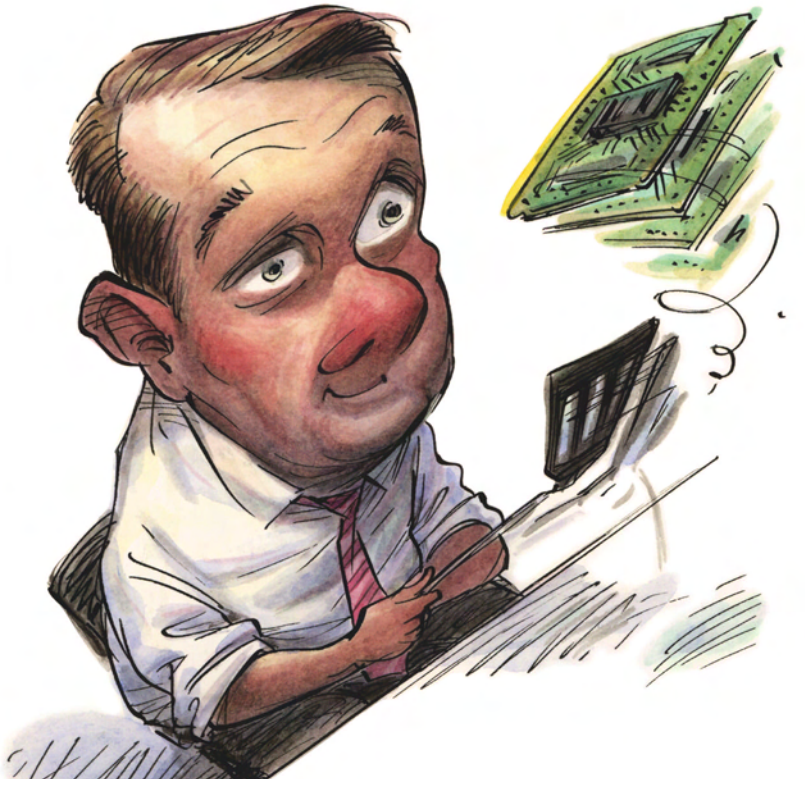
BY HOLGER STEFFENS • IDENT TECHNOLOGY

Flip-flop flap



DANIEL VASCONCELLOS

I recently was a product engineer in a DRAM company and usually spent my time hunting bugs in other people's designs. Two years ago, however, I got an additional task: I had to design a measurement board with a couple of ICs, an ADC, and a DAC to set and measure voltages, clocks, and temperature. I figured that this task would give me a good chance to gain some additional design experience.

I decided to use an Altera (www.altera.com) FPGA as the central controller chip. The FPGA connected through an FTDI (Future Technology Devices International, www.ftdichip.com) USB (universal-serial-bus) chip to the PC on which the application software was running. The development ran smoothly, and we could soon read and write data through the USB to and from the FPGA. Unfortunately, though, after a few seconds, the board hung up. I restarted it, and everything was fine—again for about 10 seconds. It soon became clear that the FPGA wasn't properly responding to the USB chip.

The design operated with eight parallel data lines and four control signals that resided between the FPGA master and the USB-chip slave. The USB chip indicated when received data was ready for pickup by sending a receiver-flag signal. Using a transmitter-enable signal, it indicated whether the USB chip's transmitter FIFO had enough room for data to write to it.

When the system hung up, I hooked up a scope and saw that the USB chip indicated receive data by pulling the receiver-flag signal, but the FPGA didn't pick it up. Not being an experienced digital designer, I couldn't figure out the problem by examining the Verilog code. Simulations didn't show any issues, either. The beauty of an FPGA, however, is that it is programmable, so I could connect virtually any internal node to free I/Os and probe them with the scope. Luckily, I didn't have to wait too long for the hang-up; I could count on it. By probing all sorts of nodes, the scope showed that almost everything inside—except the interface to the USB—was running correctly. I could even probe the 4-bit state machine of the interface controller. In the case of the hang-up, it was in an undefined state—that is, one that wasn't encoded in the state table. How did it get there?

I consulted an experienced engineer on our memory-design team, and, after a long session with him, it became clear: The controller state machine depended on the flags of the USB chip. If the state machine is idle and the receiver flag is zero, the machine can issue a read transaction. So, I typed my Verilog code exactly like that statement: If (RX_flag#==0)state<=RxTransaction, where RX_flag# is the receiver flag. In Verilog, that code all looks fine; in hardware, however, my state machine had 10 states. Therefore, 4 bits, or four flip-flops, represent the current state. Each of the flip-flops had its own combinational logic to encode when to transition, and the flag goes to all of them. Some of those combinational blocks are longer and slower than the others, and the flag occasionally transitions at just the wrong moment. It happened that only three of the flip-flops recognized that the flag sampled at zero, but the fourth, slower one sampled it at one, and the state machine became lost.

Thanks to the experienced engineer, I learned that I had committed one of the big no-nos in digital design: using an external signal directly without synchronizing it with a flip-flop to the internal clock. That lesson really helped! EDN

Holger Steffens is project manager at Ident Technology AG (Munich, Germany). You can reach him at Holger.Steffens@googlemail.com.