Ground Bounce

Bob Colwell

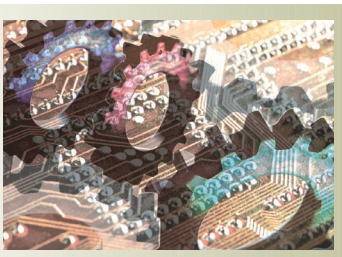
Analogies, rules of thumb, and folk wisdom are powerful tools in the arsenal of an engineer or designer. I have found these tools indispensable in my own design career, and I have observed my coworkers likewise collecting them along the way. Other fields of human intellectual endeavor seem to operate the same way.

When confronted by a chessboard on which the pieces have been randomly scattered, a chess grandmaster is not much better able to quickly memorize the pattern than anyone else. But stop a game in the middle, give that same master a quick view, and the master's ability to recall the pieces' exact positions far exceeds most people's. A key component of this mastery appears to be the ability to quickly identify the most important aspects of the board position and "abstract out" the rest.

THINKING IN ABSTRACTIONS

Some argue that the ability to think in analogies and nonmathematical abstractions is bad. I attended a lecture in 1981 by the late Edsger Dijkstra, of "Goto's Considered Harmful" fame, in which he excoriated the computer science faculty of a leading university for what he called "anthropomorphization."

To anthropomorphize is to ascribe human characteristics to things that are not human. Dijkstra castigated the CS research community for thinking about computing systems in anything other than strictly mathematical terms. He believed that to reason about system behavior in terms such as "this unit talks to that one" and "when this system wants more data, it goes out on the network and fetches it" was to be well on the way to creating buggy, unmaintainable code. The word "anthropomorphization" even *sounds* despicable.



The engineer or designer's toolkit includes analogies, rules of thumb, and folk wisdom.

Dijkstra's lecture worried me. It's daunting to hear a world-renowned computer scientist tell you that your entire way of thinking is fatally flawed and that those who don't wield math like Harry Potter (or better yet, Hermione) wields wands should find a new field to work in. I was sure that my intuition worked much better when I could find a useful abstraction for the concept I was considering. I could combine that new abstraction with the others I had already collected, and I could then consider the abstractions together and mentally simulate them to see whether they led in a promising direction.

I don't doubt that some people do this abstraction process in terms of mathematics—which is exactly what Dijkstra was demanding. But while I could wield enough math to help prove an idea would work, given the idea in the first place, I had never found math to be a great generator of the ideas themselves.

After fretting over Dijkstra's comments for a few days, the answer came to me: Ignore him and hope he's wrong. He was haranguing the entire CS faculty, many of whom didn't seem to agree with his gloomy assessment. On the other hand, not all of their rebuttals seemed particularly inspired, but since I wanted them to be right, I was in an unusually forgiving mood.

USEFUL ABSTRACTIONS

Digital circuit designers know many ways to capture the required functionality in a silicon chip subsystem. They know which ways are fast, which use the least silicon die area, and which have the lowest power requirements. They know about testing, reset, and exceptional conditions. They know where design errata like to hide, and they take pains to minimize those places in the design.

Digital designers generally think in terms of ones and zeroes—or, at most, ones, zeroes, and Xs—during chip development. An X is the "don't know what state this signal is right now" value, which is extremely useful when designing and debugging the parts of a system that get you from a cold start to a warmed-up, executing machine. Most people would be surprised at how easy it is to design a machine that works if and only if it is already working properly. But push the reset button, and all bets are off.

Analog circuit designers sometimes smile tolerantly when they interact with digital circuit designers. The analog wizards know that the "logical one" abstraction that digital designers use is actually a voltage level on a wire. That level is arrived at by a transistor providing charge at a rate that is a function of numerous physical topologies and device sizes, as well as what the previous voltage level on that wire is and where the driver of that last voltage happens to reside physically on the wire.

These designers know that the wire is a certain length and that a portion of

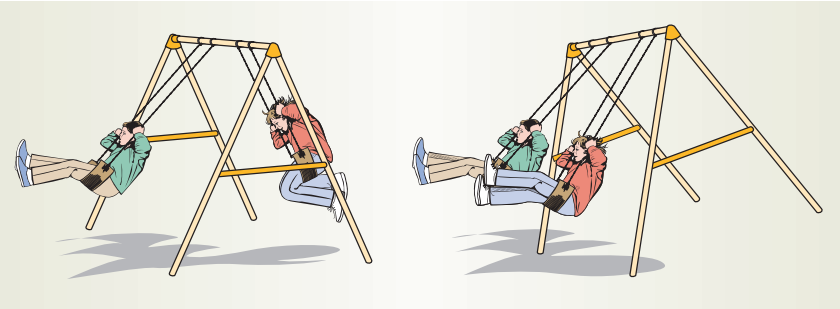


Figure 1. (left) Swinging out of phase demonstrates how out-of-phase outputs cancel, and there is no ground bounce. (right) Swinging in the same direction demonstrates in-phase outputs, adding up to ground bounce.

that length was adjacent to some other signal, which causes interesting cross-coupling effects with real energy transfer and consequences to speed and signal distortion. They even know that the real world is a tough place for an innocent signal to have to live (what would Dijkstra say about that phrase?), what with power supply noise, stray electric and magnetic fields, heating effects, and manufacturing tolerances.

Both sets of designers have rules of thumb that they know will yield good results most of the time. This base of folk wisdom saves them an enormous amount of analysis and effort, which they can then spend on the corner cases in which the rules of thumb break down.

The analog designers are used to the idea that the rules they used to good effect on the last chip are no longer correct on the next one. The same changes to silicon process technology that make a better chip possible necessarily impact electronics at the transistor level. But even digital designers can create systems using abstractions that get them into trouble: People's exhibit A—the Multiflow Trace development of the mid-1980s.

Making good things better

Think back to the 1980s. Microprocessors were slow, inflation was high, and the world was recovering from disco. Computer systems were built from medium-scale integration chips. One particularly popular family of digital logic was transistor-transistor-logic. TTL ran on 5 volts and was implemented in little rectangular "dual-inline" packages with electrical pins running down the long sides of the package. Small TTL chips had 14-pin packages (seven pins on each side); larger ones had 20 or 24 pins.

The TTL family had been around since 1965 and had grown to hundreds of different kinds of devices. You could get four 2-input NAND gates in a package—or eight buffers, a binary coded decimal counter, shift registers, flip-flops, or adders.

In a bid for more speed, TTL manufacturers first used Schottky diodes in their chips to avoid having their drive transistors "saturate," a condition that slows them down. Then, in the 1980s, they created some new family members called Advanced Schottky. And that is where Multiflow reenters our story.

Making better things much worse

In 1985, our task at Multiflow Computer Company was to design the hardware that would realize the vision of a very long instruction word (VLIW) computer, with 28 functional units executing in parallel on every clock cycle. Custom silicon was out of the question because we were a startup and didn't have the time or money for that approach. We could have designed with semicustom application-specific integrated circuits (ASICs), but that seemed risky to us compared with good old, inexpensive, reliable TTL, with which we were all experienced.

That is not how it turned out. Our abstraction for TTL had been compromised with the introduction of Advanced Schottky, but we did not learn that until it was far too late.

First, an educational diversion: When you were a kid, you sat on a swing and pumped back and forth. If it was a good swing set that had been properly secured to the ground, that is all you did. Your mental model of the swing set was validated and implicit. But if the swing set was constructed with an unsecured pair of steel legs at each end in the inverted-V configuration, swinging vigorously enough rudely disproved your "swing-set-mental-model"—one of the sets of steel legs could lift up off the ground and threaten to topple the whole structure. And if you're like me, you actually proved this conjecture to yourself, possibly more than once.

On an unsecured swing set, you also discovered that if a friend was in the swing next to you, you were both okay as long as you swung perfectly out of phase—he was swinging backward as you swung forward, and vice versa. As Figure 1 shows, if you got into phase, both swinging in the same direction at the same time, that swing set could definitely flip over.

Okay, let's return from the diversion. Much to our chagrin, Multiflow found that the Advanced Schottky TTL chips were not meeting their published specifications. Well, more precisely, sometimes they did and sometimes they didn't.

After many all-night debug sessions, the pattern that became clear to us was that if you only used, say, four buffers of a chip designed to have eight, those four would probably meet their spec. But if you attempted to use all eight, and all the signals moved in phase, they would destructively interfere with one another. In other words, the speed with which that octal buffer could send a new byte partly depended on the value of the previous byte it had sent. Moreover, it also depended on the actual value of the byte itself.

For the same reason that you and a friend could swing happily as long as you were out of phase, these Advanced Schottky octal buffers would meet their published specs if you sent [10101010] but not [11111111]. And you were doomed if you tried sending [11111111] and [00000000] in a repeating pattern. In fact, that pattern would cause eerie, hollow laughter to emanate from the machine. Or maybe that was just my boss. It's hard to tell.

The root cause of all these problems is a phenomenon called "ground bounce," a voltage oscillation between the ground pin on a component package and the ground reference level on the component die. Figure 2 shows how simultaneous switching of outputs induces a ground bounce glitch.

In electronics education, wires are assumed to be perfect conductors of electricity. Real-world wires exhibit parasitic capacitance to other wires, and they also exhibit inductance. Inductance is the propensity of a wire to create a magnetic field that accompanies any electrical current flowing through that wire.

Current flow creates magnetic fields, and collapsing magnetic fields induce current flow. This can be useful—it is why transformers work. But inside TTL packages, this inductance went from being an ignorable nuisance in the 1970s to an intolerable disaster in the 1980s.

Recall that Advanced Schottky did two things: improved the speed and put more functionality into longer packages with more pins. This meant that the corner pins—the power and ground connection—had the longest bond wires inside the device's package. Longer wires have more inductance, and faster chips drive much more current. As Figure 2 shows, the result was that the inductance of the internal bond wire could easily cause the internal ground reference point of the silicon to go from the desired zero volts to +1.5 V or –2 V for nontrivial amounts of time.

We Multiflow hardware designers eventually got advanced degrees in the ground bounce school of hard knocks and exorcised the ground bounce demons in the Trace VLIWs. We learned that dual-inline packaging was the worst; surface-mount devices were much better due to very small ground/power lead lengths in the packages.

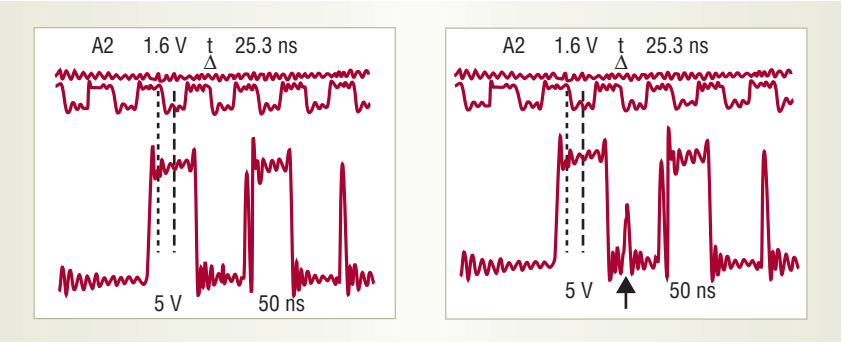


Figure 2. (left) Normal output switching. (right) Arrow indicates ground bounce-induced glitch from simultaneous switching of other outputs.

Edge rates of transitioning signals were far more important than DC loads on those signals because the size of the voltage spike induced in a chip was proportional to dI/dT. If you stop a clock, stop it high, not low—TTL has a much higher noise margin in the high state. Stop a clock low, and a transient ground bounce glitch will be seen as a new clock.

Beware tri-stating, a common TTL-era technique in which a chip would electrically remove its drivers from a wire so that another chip could drive it. Don't assume that devices with only a few outputs should be immune from ground bounce. I chased a transient error in a floating-point unit for a week once, only to discover that a four-bit adder chip was causing it. We even had chips in which ground bounce on the outputs was causing a big enough spike inside the chip that it clocked itself again.

LESSONS ABOUT ASSUMPTIONS

Which brings me back to the beginning. We chose TTL because we thought it would be a safe technology; we'd all had many years of experience with it. After suffering through a year of intensive debug and redesign surrounding the ground bounce affliction, well, in the immortal words of the Who, "We don't get fooled again." So we signed up with an ASIC vendor and switched to ECL, an older technology than TTL but intrinsically faster, and with a marked tendency to drive both Q and Q-bar at all times, which should largely cancel ground bounce.

But I'm a paranoid engineer. So I thought I'd just do a little checking before diving in, and I called the ASIC vendor. Here's how that phone call went.

Me: How many output lines can I drive simultaneously from one of your ASICs?

Vendor: 72.

Me: (Wow, 72, that's great, much better than TTL. I'm so glad those ugly days are behind us. On the other hand, our new design needs to drive about 72 outputs, so this had better work.) Just out of curiosity, how many can you drive without having ground bounce problems?

Vendor: Uh, well, we're not sure, actually, but we did have one customer who tried 73 and reported ground bounce issues.

And at that exact moment, I knew the true meaning of Charlie Brown's famous utterance: ARRRRGGGGHHHH! ■

***Bob Colwell** was Intel's chief IA32 architect through the Pentium II, III, and 4 microprocessors. He is now an independent consultant. Contact him at bob.colwell@attbi.com.*