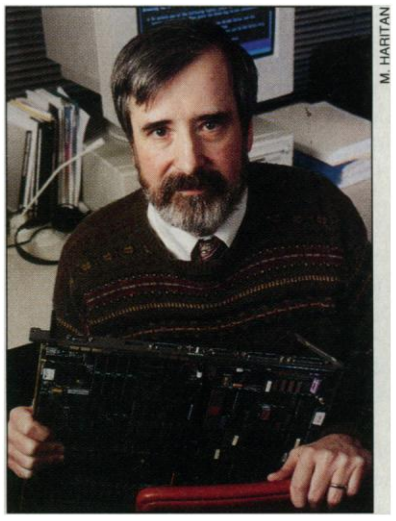
COMPUTER SCIENCE

Mathematical Logic Flushes Out the Bugs in Chip Designs

When the world's largest computer-chip manufacturer hardwires a simple arithmetic mistake into its lead product, the reverberations may not stop for months, even years. Much of the reaction to the news that Intel Corp.'s Pentium chip gives imprecise answers when dividing statistically rare combinations of numbers has focused on the bug's consequences for users and on Intel's heavily criticized initial handling of the situation. But Intel's misfortune may also have a more positive legacy—by giving a boost to researchers working to improve techniques for catching flaws in microprocessor designs before they get translated into hardware and become ruinously expensive to fix.



Faultfinder. Carnegie Mellon's Edmund Clarke.

Standard approaches to debugging chip designs rely heavily on "simulation"—running software mock-ups of a chip through as many operations as possible before the chip goes to market. But because software simulations often run millions of times slower than the chip itself does, their ability to nab complex errors can be severely hampered. In the case of the Pentium, says Carl Pixley, a researcher in Motorola's semiconductor products sector, "one thing is obvious: They simulated the hell out of it and didn't find [the error]." What might have found Pentium's flaw, at least in the view of some experts (see box), is an entirely different strategy called formal verification that has recently come into its own.

The underlying philosophy of this strategy is simple: Instead of testing the chip in as many ways as possible, scrutinize the logic of the design to prove, once and for all, that the chip will work as intended. The strategy itself, however, is more complex to put into practice—and there are several different approaches to executing it. One of them treats the logic of the entire chip as a giant "theorem" to be proven correct, by taking a mathematical model of the chip and stepping it through many different operations. A second approach focuses on specific chip behaviors that lie at the nexus of many computation paths and proves that their logic is ironclad. With this kind of formal verification, says Robert Kurshan, a researcher at AT&T Bell Laboratories, you ask the question "Is it ever possible in any [chip] lifetime to do something bad?"

Long considered a theoretical pursuit best left to academe, formal verification has seen "just tremendous advances in the last couple of years," says John Rushby, a computer scientist who specializes in formal verification at SRI International in Menlo Park, California. Among those advances are new ways to keep the symbolic expressions that describe the internal logic of million-transistor chips to a manageable size and split the problem of verifying an entire chip into manageable chunks. Researchers have also been aided by increases in computer power itself, harnessed in automated logic programs. With these new tools in hand, formal verification is racking up a growing list of successes, identifying errors in everything from specialized chips in aircraft to control units at chemical plants.

Lost in (state) space. What makes it so hard to check the design of large microprocessors is the vast "state space" available on these devices. Physically, a chip like the Pentium is made up of millions of transistors combined in a network of "latches" and "gates." At each cycle of the chip's internal clock, every latch is in a binary state of either 1 or 0 (which can be thought of as logical predicates "true" or "false"); each array of 1's and 0's constitutes one state of the chip.

But that's just the beginning. A chip with n latches has 2^n possible states—a number that, for a chip like the Pentium, is huge almost beyond reckoning. Simulation is, in effect, an informal rummage through some sequences of those states. As a result, "it might be that you could test a circuit until Kingdom come and not find the bug," says Ken McMillan, a specialist in formal verification at Cadence Berkeley Laboratories.

Formal verification replaces that sampling strategy with an extension of the kind of logical reasoning familiar from mathematics. If, for example, you are told that $x + y = z$ and $x = y + z$ can't both be true if x, y, and z are integers greater than 0, you could try testing that statement by plugging in many different numbers; after repeating this "simulation" many times, you might conclude that it is correct. Or you could apply logic to prove it, once and for all, for any positive integers. Similarly, formal verification tries to provide general proofs of the "truth" or "falsehood" of the logic embodied in the circuit's hardware.

One way to do so, called theorem proving, is an effort to prove mathematically that the chip will perform a complex operation—division, for example—exactly as advertised no matter what the inputs. "The theorem is that the user level [say, the command to divide] and the gate level do the same thing," says Robert Boyer of Computational Logic Inc. in Austin, Texas, who along with colleague J. Strother Moore, also at CLI, is considered a pioneer of theorem proving. To verify the chip's division function, Boyer says he would translate the command into its equivalent in "the ugly sea of transistors." At that level, the variables x and y become strings of variables representing binary place-holders, arbitrary signs, and floating-point exponents.

Then he would formally step the chip (actually a mathematical model) through the needed number of clock cycles, allowing the gate logic to act on these new variables. Finally, he would retranslate the end point into the user-level language. The result, says Boyer, should be a theorem that the statement "This sucker divides" is true for all numbers accepted by the machine. To verify the overall working of a chip, he and his colleagues use their theorem-prover program to repeat the process for a number of carefully selected commands, each representing a critical function of the chip.

The second approach to formal verification, model checking, is an effort to simplify the problem by focusing not on command-level operations like division but on underlying chip behaviors, especially those that involve communication among chip components. After all, many of the most troublesome bugs are sequences of operations that lead to unwanted conditions such as "starvation," in which one component's request for information always takes precedence over another's, leaving the second component permanently idle.

Keeping track of time. To check that a chip handles these sequences properly, computer scientists Edmund Clarke and Allen Emerson, then at Harvard University, realized in 1981 that they needed a language for expressing time-related behaviors. They found it in a symbolic notation called temporal logic, devised in the 1940s by philosophers investigating the use of time in human language. Temporal operators provide precise, symbolic ways of representing sentences like "Irv will be in the kitchen until Helen

Second-Guessing Intel

Could the technique known as formal verification have caught the Pentium error before the chip went into production? Some advocates of this technique for tracking down errors without slogging through all of the chip's astronomical array of possible states (see main text) believe the method could have caught the bug. Others, citing the complexity of the floating-point division algorithm in which the error turned up, aren't certain. But just to be sure, Intel has already incorporated formal verification into the design effort for the Pentium's successor, the P6.

The Pentium bug can be thought of as a glitch in ordinary long division. In dividing, say, 627 by 28 (not the actual numbers that are a problem), you first need a trial divisor: in essence, a guess about how many times 28 goes into 62. Like people, computer chips rely on tables of multiples to generate their trial divisors, and the table built into the Pentium omitted several entries. Although the mistake is obvious—in hindsight—few researchers with experience at Intel ascribe the goof to a lack of vigilance.

Indeed, says Olivier Coudert, a pioneer in formal verification at Synopsys Inc. in Mountain View, California, verifying floating-point dividers and multipliers is "incredibly difficult" because of the vast array of different "states" such units can adopt. As a result, says Robert Kurshan of AT&T Bell Laboratories, "even if [Intel] had had the most sophisticated [formal-verification] tool then available, they probably could not have ... expected to find the bug." He adds, however, that recently developed tools combining two different approaches to formal verification might have succeeded. But in a letter to *Science*, Robert Boyer and J. Strother Moore of Computational Logic Inc. in Austin, Texas, a company that specializes in formal verification, assert that "the now-famous Pentium bug could have been found [at the time] by using well-known, formal ... techniques."

"Our experts are divided on whether one could find it through formal verification," responds Randy Steck, Intel's design manager for the P6. Nevertheless, says Steck, he and his team "will push [formal verification] as far and as fast as we can" on the P6 effort, and he adds that in limited areas the approach has already shown "really good results."

–J.G.

rings the front doorbell."

More relevant to computers, temporal operators provide a way of representing desirable circuit properties, such as the requirement that a component's request for data always be satisfied. Clarke, who is now at Carnegie Mellon University (CMU) in Pittsburgh, proposed that when a desired property of a circuit is expressed this way, a "model-checking" program could compare it with a model of the logic embodied in the actual hardware design. One strategy is to take the opposite of the desired behavior—a bug—and then drive the circuit model backward in time to find all chains of commands that could lead to that undesirable state. If, among those chains, you discover any states accessible to the chip in normal operation—for example, the state the chip is in after booting up—there is a flaw in the design.

Because the method takes into account only those sequences of states that affect the behavior of interest, it is more efficient than rambling through all of state space. At the same time, says Patrick Scaglia, vice president of Cadence Berkeley Laboratories, the method "goes through the entire set of possible states [leading to an error]." As a result, "it can identify sequences you never would have thought of." These long, improbable sequences are sometimes called "cold-sweat bugs," in reference to chip designers who are said to wake up in cold sweats fretting about them.

Still, Kurshan points out, model-checking routines can't guarantee chip designers a good night's sleep. The routines can still sometimes be "blown out of the water" by the number of states open to large, commercial chips. The huge, formal expressions generated by a chip's logic can also give theorem provers nightmares, sometimes limiting that technique's effectiveness.

But researchers are learning to cope with the complexity by enlisting aids such as binary decision diagrams (BDDs), logical structures that can serve as a proxy for large numbers of chip states, explains Randy Bryant, a computer scientist at CMU who has advanced the mathematical foundations of the method. The work of verifying a chip can also be divided into component parts of manageable size thanks to a technique known as "abstraction"—practiced notably by Kurshan—which allows researchers to focus on one part of a circuit without worrying about its interactions with some other part.

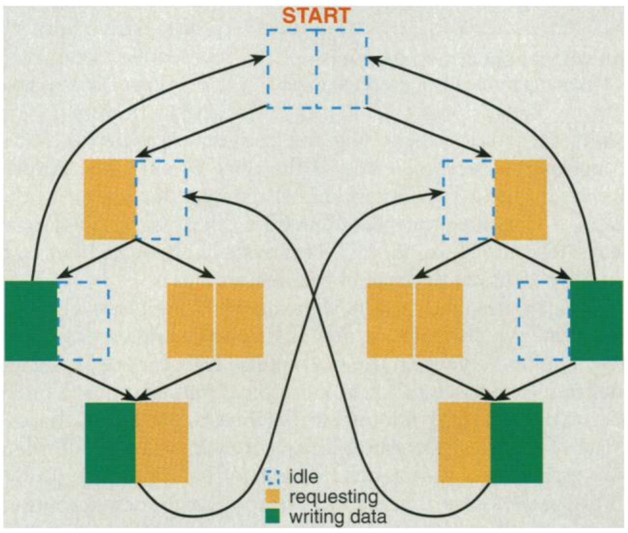
As a result of these advances, the list of formal verification's practical successes is growing rapidly. Rushby and colleagues at SRI and Collins Commercial Avionics in Cedar Rapids, Iowa, recently used a theorem prover developed at SRI to find bugs in a microprocessor called the AAMP5, which has half a million transistors and can be used to control cockpit displays in passenger aircraft, among other functions. Using model checking, Clarke, McMillan, and others found errors in a design standard for a hardware bus, which had been adopted by the U.S. Navy. Still other researchers have applied formal verification to uncover flaws in a chip at the core of a jet fighter's controller and in consumer microprocessor designs being developed at Motorola and Fujitsu.

Clarke is also extending the strategy to other complex technologies, ranging from communications protocols to railroad switching routines to the logic controllers of chemical factories. But it's the chip companies that will be looking for all the help they can get in the coming months, says David Dill, a formal-verification researcher at Stanford University, as they struggle to cope with their products' complexity and avoid a repeat of the Pentium calamity. "It was simply a matter of time before one of these consciousness-raising bugs hit," says Dill of the Pentium error. "This won't be the last."

–James Glanz

James Glanz is a free-lance science writer in Chicago.

SOURCE: EDMUND CLARKE



Good behavior? A logical model shows how a circuit schedules actions that can't be completed simultaneously—say, writing into a memory through two separate channels. To verify the circuit behavior, a "model-checking" program might test, for example, whether every request to write data is eventually satisfied.