A NEW RANDOM-NUMBER GENERATOR USES DIGITAL CIRCUITS TO STUMP THE SMARTEST HACKERS

By Greg Taylor & George Cox

Imagine that it's 1995 and you're about to make your very first online purchase. You open your Netscape browser, sipping coffee as the home page slowly loads. You then navigate to Amazon.com, a new online bookstore your friend told you about. As you proceed to make your purchase and enter your payment information, the address your browser points to changes from one starting with "http" to one that begins with "https." That signals that your computer has established an encrypted connection with Amazon's server. This allows you to send credit card information to the server without worrying that an identity thief will intercept the transmission.

Unfortunately, your first online transaction was doomed from the start: It will soon be discovered that the supposedly secure transfer protocol your browser just followed wasn't very secure after all.

The problem was that the secret keys Netscape was using weren't random enough. They were strings of only 40 bits, meaning there were around a trillion possible number combinations. That may seem like a lot, but hackers were able to break these codes, even with mid-1990s computer speeds, in about 30 hours. The nominally random number Netscape used to form a secret key was based on just three values—time of day, process identification number, and parent-process identification number—all of them predictable. This allowed the attackers to reduce the number of keys that they needed to try, and to find the right one much sooner than Netscape had anticipated.

Netscape's programmers would have loved to use a completely random number to form the encryption key, but they had a hard time figuring out how to come up with one. That's because digital computers are always in well-defined states, which change only when the programs they are running tell them to change. The best you can often do with a computer is to simulate randomness, generating what are called pseudorandom numbers by using some sort of mathematical procedure. A set of such numbers may at first glance look perfectly random, but somebody else using the same procedure could easily generate exactly the same set of numbers, which often makes them a poor choice for encryption.

Researchers have managed to devise pseudorandom-number generators that are considered cryptographically secure. But you must still start them off using a special seed value; otherwise, they'll always generate the same list of numbers. And for that seed, you really want something that's impossible to predict.

Fortunately, it's not hard to harvest truly unpredictable randomness by tapping the chaotic universe that surrounds a computer's orderly, deterministic world of 1s and 0s. But how exactly do you do that?

For several years, you could find an online source of random numbers, called Lavarand. It got its numbers from the pictures a computer took of the waxy blobs churning away inside lava lamps. More sophisticated hardware-based systems use quantum-mechanical phenomena, such as photons striking a half-silvered mirror, as a basis for generating random numbers. You can even get an ordinary unassisted computer to produce random numbers based on erratic events taking place within its own mundane hardware—the precise timing of keystrokes, for example. But to get many of these numbers, you'd need to hammer away at a lot of keys.

A WHOLE LOT OF LAVA Lavarand was developed in 1996 to generate randomness from lava lamps. Over a million people grabbed numbers from the Lavarand website.

We and our colleagues at Intel think this should be easier. That's why for more than a decade now, many of our company's chip sets have included an analog, hardware-based random-number generator. The problem is that its analog circuitry squanders power. Also, it's hard to keep that analog circuitry working properly as we improve our fabrication processes. That's why we have now developed a new and entirely digital system that allows a microprocessor to produce a copious stream of random values without those difficulties. Soon it will be coming to a processor near you.

INTEL'S FIRST ATTEMPT to help an average computer produce better random numbers came in 1999, when the company introduced the Firmware Hub chip-set component. The random-number generator in that chip is a ring-oscillator-based analog design that works by taking some of the thermal noise that's present in all resistors, amplifying it, then using the resulting jittery signal to change the period of a relatively slow-running clock. On each of the erratic ticks of that slow clock, the chip then samples the output of a second, fast-running clock, which cycles back and forth quite regularly between its two possible binary states: 0 and 1. That erratic sampling then produces a random sequence of 1s and 0s.

The rub here is that the analog circuitry needed to amplify thermal noise consumes lots of power—and that circuitry operates whether or not the computer actually needs random numbers for what it's doing at the time. Those analog circuits are also a nuisance when it comes time to improve the manufacturing technology used to make the processor. Every few years, chipmakers modify their fabrication lines to produce integrated circuits at a finer scale, allowing them to pack more transistors into the same area. Making these shifts is pretty straightforward for CMOS digital circuitry, but each new generation of analog circuitry requires careful reevaluation and testing—a major headache.

That's why in 2008 Intel set out to make a random-number generator that uses only digital hardware. Intel researchers based in Hillsboro, Ore., and at the company's Bangalore Design Lab in India started by tackling a key part of the problem—how to make a random source of bits without using analog circuitry.

PREVIOUS PAGES: TYPOGRAPHY: CARL DeTORRES; THIS PAGE: ISTOCK.COM

Ironically, the solution this team came up with breaks a cardinal rule of digital design: that a circuit should be in a well-defined state, outputting only logical 0s or logical 1s. A digital circuit element can, of course, spend short periods switching between these two possible states. But it should never remain teetering, uncertain of which way to go. That would introduce delays and could even produce system failures.

But in this random-bit generator, teetering is a feature, not a bug. The design includes a pair of inverters, circuit elements whose single output is the opposite of its single input. We connect the output of one inverter to the other inverter's input. If the first inverter's output is 0, so is the input to the second inverter, whose output is then 1. Or if the first inverter's output is 1, the second inverter's output will be 0.

This random-bit source adds two rather oddly placed transistors to this two-inverter circuit. Switching those transistors on forces the inputs and outputs of both inverters to the logical 1 state. The inverters have to be modified slightly to take this sort of abuse, but that's easy enough to do.

Interesting things can now happen when the added transistors are turned off. The two-inverter circuit doesn't like having all its inputs and outputs in the same state: It wants the output of one inverter to be 0 and the other 1. But which inverter should output which?

There are two possibilities, and for the briefest of moments, the circuit hovers between them. In a perfect world, it might linger like that forever. But in reality, even a small amount of thermal noise—random atomic vibrations—within the circuitry will send it racing toward one of its two stable states. It's the physically random properties of the thermal noise that determine the outcome of this otherwise indecisive circuit.

In this way, our simple digital circuit can easily harvest some of the ubiquitous randomness of nature. All we need to do is to connect those two extra transistors to a clock that regularly turns both of them on and off. Every time the clock cycles, the circuit generates one random bit.

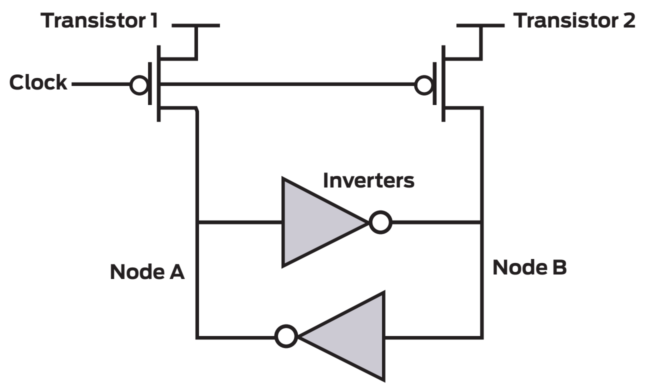
This digitized approach to random-bit generation would work fine if all inverter circuits were absolutely identical. But the messiness of the physical world never really allows that. In reality, no two inverters are exactly the same. Having subtle differences in the speed or strength of their responses might seem like a mild offense, but in this application, such differences could easily compromise the randomness we were trying to extract from the circuit.

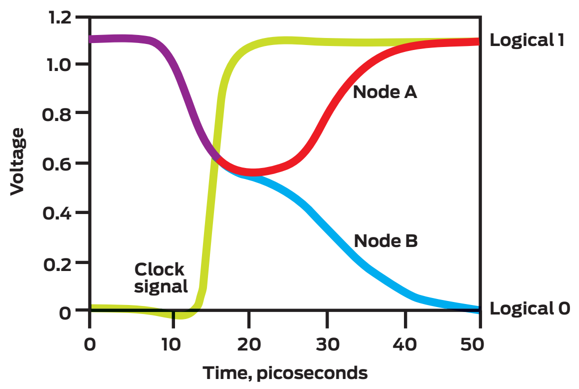
To keep the inverters in balance, we built a feedback loop into the new hardware. The circuitry in that loop performs some targeted fiddling until the two possible output values, 0 and 1, each occur roughly half the time. This helps our design satisfy one of the rules for statistical randomness: In a long stream of numbers, there should be roughly the same number of all possible digits. By adjusting the internal workings of each inverter on the fly, we can defend against the predictability that cryptologists so dread.

IN PARALLEL WITH RESEARCH EFFORTS to refine a consistent digital source of random bits, other Intel engineers began developing the additional logic needed to deliver those bits effectively in a product. You might think that a processor could simply collect the output bits from basic random-number-generator hardware, feed them to an application, and call it a day. But the truth is that those bits aren't as random as you'd want them to be. The raw bit stream coming out of basic hardware, no matter how good, can have bias and correlation.

Our goal was to build a system that delivers random numbers compliant with and certified to meet common cryptographic standards, such as those set by the National Institute of Standards and Technology. To guarantee the quality of our random numbers, we designed a three-stage process—which uses digital circuitry, a conditioner, and a pseudorandom-number generator—now known by the code name Bull Mountain.

Our previous analog random-number generator was able to produce only a few hundred kilobits of random numbers a second, whereas our new generator spins them out at a rate of around 3 gigabits per second. It starts by collecting the mostly random output of the two inverters 512 bits at a time. Further circuitry then breaks each package of 512 bits into a pair of 256-bit numbers. Of course, if the original 512 bits aren't completely random, those 256-bit numbers won't be completely random either. But they can *Continued on page 56*

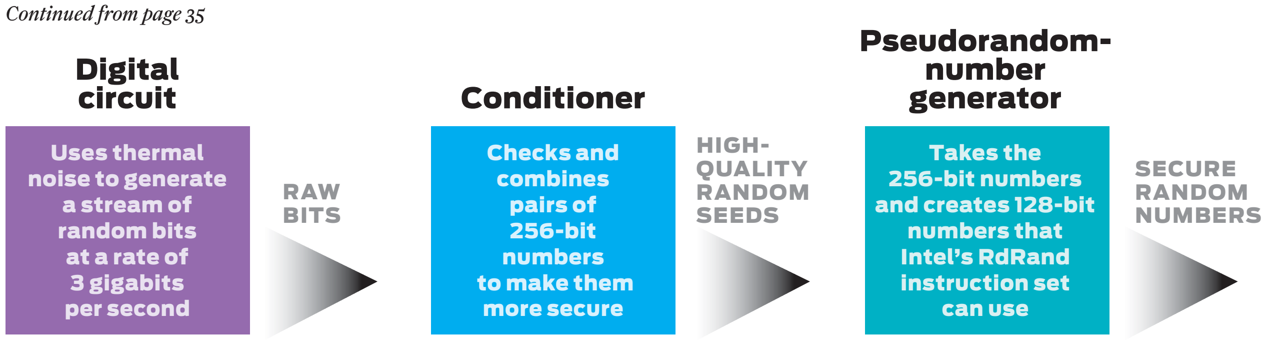




UNCERTAIN CIRCUITS: When transistor 1 and transistor 2 are switched on, a coupled pair of inverters force Node A and Node B into the same state [left]. When the clock pulse rises [yellow, right], these transistors are turned off. Initially the output of both inverters falls into an indeterminate state, but random thermal noise within the inverters soon jostles one node into the logical 1 state and the other goes to logical 0.

Digital Randomness

Continued from page 35


Digital circuit
Uses thermal noise to generate a stream of random bits at a rate of 3 gigabits per second
RAW BITS
Conditioner
Checks and combines pairs of 256-bit numbers to make them more secure
HIGH-QUALITY RANDOM SEEDS
Pseudorandom-number generator
Takes the 256-bit numbers and creates 128-bit numbers that Intel's RdRand instruction set can use
SECURE RANDOM NUMBERS

THREE-LAYER NUMBERS: Intel's Bull Mountain random-number generator prevents bias and correlation with a three-step process. First, a digital circuit generates a stream of raw random bits. Next, a conditioner generates healthy random seeds. Third, a pseudorandom-number generator readies the numbers for use in software.

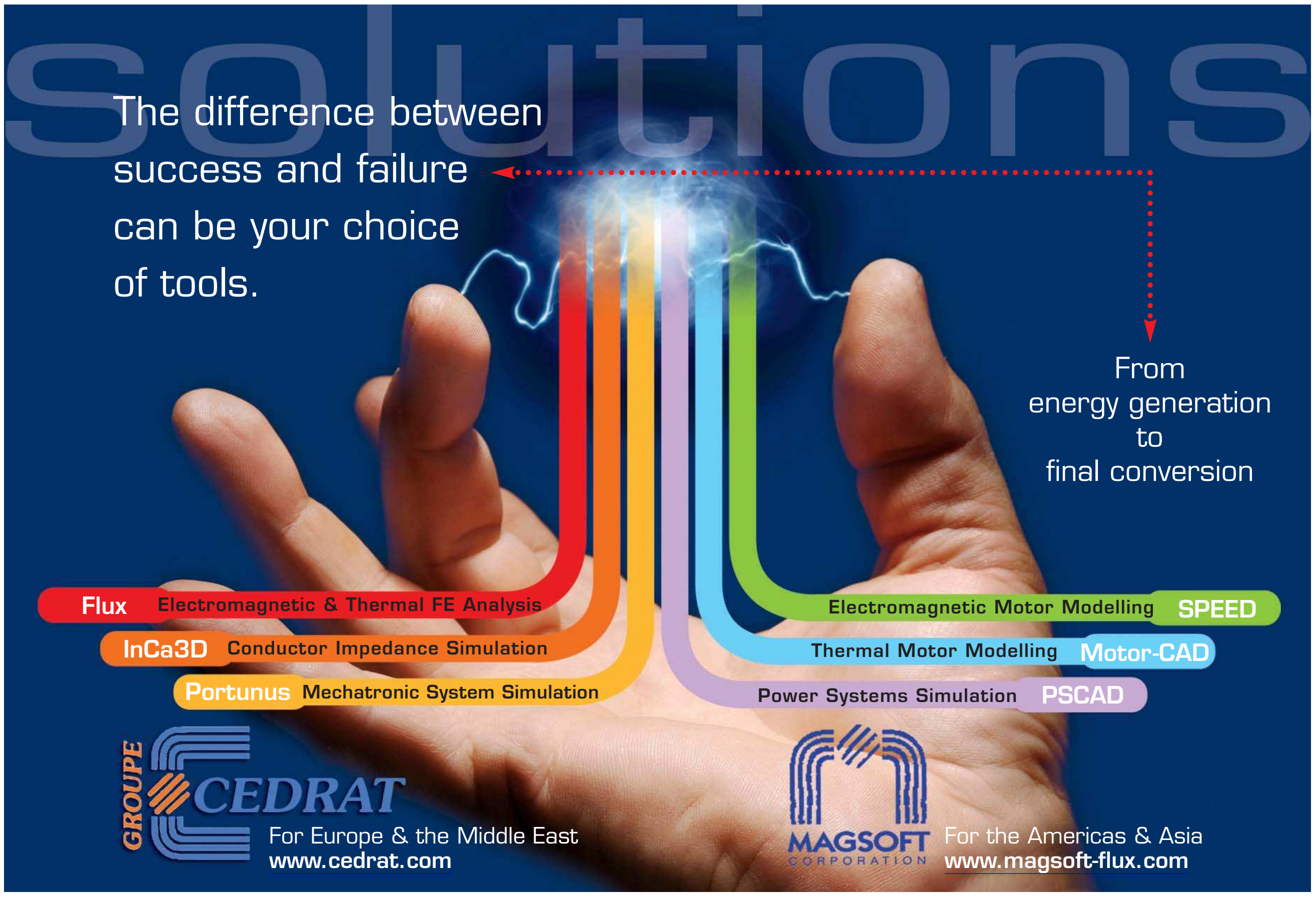
be mathematically combined or conditioned in such a way as to produce a 256-bit number that's closer to that ideal.

You can see this better with a simple illustration. Suppose for a moment that the random-bit generator spits out just 8 bits at a time, which is to say that it provides a stream of binary numbers that range in value from 0 to 255. Suppose further that those 8-bit numbers aren't completely random. Imagine, for example, that some subtle flaw in the circuitry tends to suppress values at the high end of the range. On casual examination, the random-number stream looks good, but if you plot millions of values, you'll notice that the high numbers come up a little less frequently than the low numbers.

One possible fix for that is simple: Always take two of these 8-bit values, multiply them together, and then throw out the upper 8 bits of the resultant 16-bit number. Such a procedure would largely eliminate the bias.

Bull Mountain doesn't work with 8-bit numbers: It works, as we said, with 256-bit numbers. And it doesn't just multiply two together—it does a more sophisticated cryptographic combination. But the basic idea is the same. You can think of this conditioning step as concentrating whatever degree of randomness the two-inverter circuit can provide.

We really like to sleep well at night, so we've built in additional circuitry that tests to make sure that the concentrat-


solutions
The difference between success and failure can be your choice of tools.
From energy generation to final conversion
Flux Electromagnetic & Thermal FE Analysis
InCa3D Conductor Impedance Simulation
Portunus Mechatronic System Simulation
Electromagnetic Motor Modelling SPEED
Thermal Motor Modelling Motor-CAD
Power Systems Simulation PSCAD
GROUPE CEDRAT
For Europe & the Middle East
www.cedrat.com
MAGSOFT CORPORATION
For the Americas & Asia
www.magsoft-flux.com

ing machinery is always working with bit streams that aren't too biased. If it detects a biased output, we tag it as unhealthy and refine it to meet our standards. This way, only healthy pairs of bit streams are combined.

Guaranteed randomness isn't all that useful if the random values aren't produced and vetted quickly enough to meet demand. While the hardware's circuitry generates random numbers from thermal noise much more quickly than its predecessors, it's still not fast enough for some of today's computing requirements. To allow Bull Mountain to spew out random numbers as quickly as software pseudorandom-number generators, and also maintain the high quality of the random numbers, we add yet another level of circuitry. It uses the 256-bit random numbers to seed a cryptographically secure pseudorandom-number generator that creates 128-bit numbers. From one 256-bit seed, the pseudorandom-number generator can spit out many pseudorandom numbers. With those seeds coming at a rate of 3 gigahertz, a healthy supply of these secret codes quickly builds up.

A new instruction, called RdRand, provides a way for software that needs random numbers to request them from the hardware that's producing them. Built for Intel's 64-bit microprocessors, RdRand is the key to using Bull Mountain. It retrieves a 16-, 32-, or 64-bit random value and makes it available in a software-accessible register. The RdRand instruction has been public for about a year, and the first Intel processor to provide it is known as Ivy Bridge. The new chip set performs 37 percent faster than its predecessor, and its smallest features have been reduced from roughly 32 nanometers to about 22. The overall increase in efficiency easily accommodates the demands of our random-number generator.

WHILE LAVA LAMPS MIGHT LOOK COOL, they don't go with every decor. We think our approach to random-number generation, on the other hand, will be the most versatile of its kind, able to work across the range of Intel silicon products.

As we mentioned, the timing of keystrokes has provided a convenient source of randomness for seeding random-number generators in the past. So have mouse movements and even the time it takes a disk drive to find some of the information stored on it. But such events don't always give you enough random bits, and to some extent the bits you get are predictable. Worse, because we now live in a world of servers with solid-state drives and virtualized workloads, these sources of physical randomness just aren't available to a lot of computers. Those machines need to be able to get random numbers from something other than random events taking place on the periphery. Bull Mountain provides a solution.

POST YOUR COMMENTS *online at* http://spectrum.ieee.org/random0911

So if you're a programmer, get ready for a prolific source of randomness to be put at your fingertips. And even if you don't want to part with a pseudorandom-number generator you've grown to love—whether for cryptography, scientific computing, even gaming—you'll now have Bull Mountain to produce the seeds for it. We expect to see those seeds scattered widely, with all sorts of wonderful software growing and blossoming as a result. ❑

The future of codeless test automation software

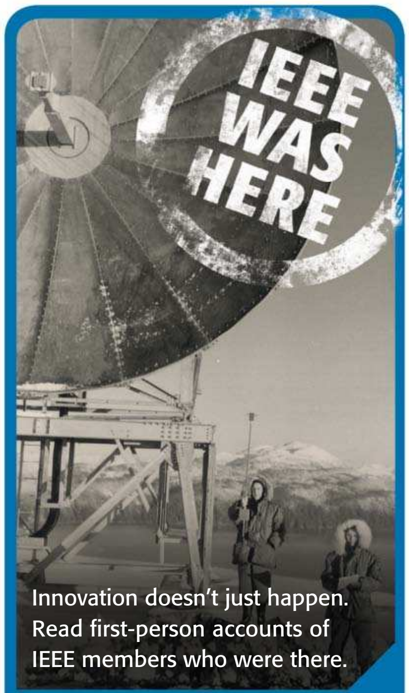
Testr3 Studio™

Establish a cost-effective solution
3 simple steps to fully automate
your functional tests.

Supports: GPIB - RS232 - TCP/IP - GPIB/LAN - LxI - CAN - SPI - I2C

www.versatyle.com

VERSATYLE® TEST CORPORATION



Innovation doesn't just happen.
Read first-person accounts of
IEEE members who were there.

IEEE Global History Network
www.ieeeghn.org

IEEE