Posted 28 Feb 2018 | 16:00 GMT

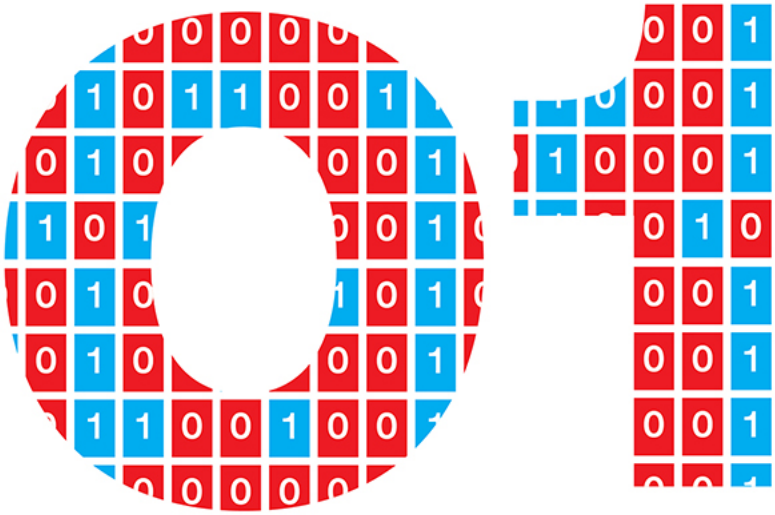

Illustration: Mark Montgomery

In electronics, the past half century has been a steady march away from analog and toward digital. Telephony, music recording and playback, cameras, and radio and television broadcasting have all followed the lead of computing, which had largely gone digital by the middle of the 20th century. Yet many of the signals that computers—and our brains—process are analog. And analog has some inherent advantages: If an analog signal contains small errors, it typically won't really matter. Nobody cares, for example, if a musical note in a recorded symphony is a smidgen louder or softer than it should actually be. Nor is anyone bothered if a bright area in an image is ever so slightly lighter than reality. Human hearing and vision aren't sensitive enough to register those subtle differences anyway.

In many instances, there's no fundamental need for electronic circuitry to first convert such analog quantities into binary numbers for processing in precise and perfectly repeatable ways. And if you could minimize those analog-to-digital conversions, you'd save a considerable amount of energy right there. If you could figure out how to process the analog signals in an energy-conserving fashion, you'll be even further ahead. This feature would be especially important for situations in which power is very scarce, such as for biomedical implants intended to restore hearing or eyesight.

Yet the benefits of digital over analog are undeniable, which is why you see digital computers so often used to process signals with much more exactitude—and using much more energy—than is really required. An interesting and unconventional compromise is a method called stochastic computing, which processes analog *probabilities* by means of digital circuits. This largely forgotten technique could significantly improve future retinal implants and machine-learning circuits—to give a couple of applications we've investigated—which is why we believe stochastic computing is set for a renaissance.

Stochastic computing begins with a counterintuitive premise—that you should first convert the numbers you need to process into long streams of random binary digits where the probability of finding a 1 in any given position equals the value you're encoding. Although these long streams are clearly digital, they mimic a key aspect of analog numbers: A minor error somewhere in the bitstream does not significantly affect the outcome. And, best of all, performing basic arithmetic operations on these bitstreams, long though they may be, is simple and highly energy efficient. It's also worth noting that the human nervous system transfers information by means of sequences of neural impulses that strongly resemble these stochastic bitstreams.

Consider a basic problem: Suppose you're designing a light dimmer with two separate controls, each of which outputs a digital value representing a fraction between 0 and 1. If one control is fully turned on but the other is at, say, 0.5, you want the light to be at 50 percent brightness. But if both controls are set to 0.5, you want the light to run at 25 percent brightness, and so forth. That is, you want the output to reflect the value of the two control settings multiplied together.

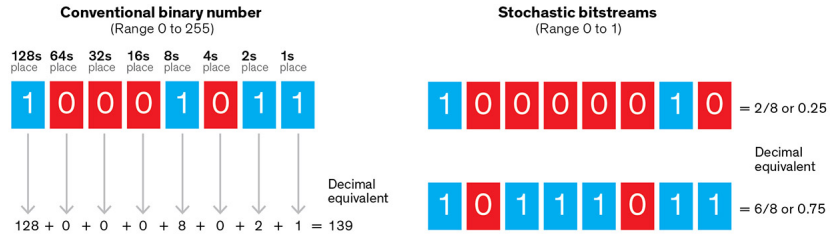

Illustration: David Schneider and Mark Montgomery

By the Numbers: Conventional binary numbers, just like the decimal numbers in everyday use, rely on the concept of place value [left]. Stochastic bitstreams don't use place value; the value they represent is determined by how often 1s appear [right].

You could, of course, achieve this using a microprocessor to carry out the multiplication. What if, instead, the output of your two controllers was transformed electronically into a random series of 0 or 1 values, where the probability of a 1 appearing at any given position in this stream of bits encodes the value at hand? For example, the number 0.5 can be represented by a bitstream in which a 1 appears 50 percent of the time, but at random points. Elsewhere in the stream, the bits have a value of 0.

Why go through the trouble of converting the number like this? Because basic arithmetic operations on such bitstreams are remarkably easy to accomplish.

Consider the multiplication you need to set the brightness of the light. One of the rules of probability theory states that the probability of two independent events occurring simultaneously is the product of the probabilities of the individual events. That just makes sense. If you flip a penny, the probability that it will land on heads is 50 percent (0.5). It's the same if you flip a dime. And if you flip both a penny and a dime at the same time, the probability that both will land on heads is the product of the individual probabilities, 0.5 x 0.5 or 0.25, which is to say 25 percent. Because of this property, you can multiply two numbers encoded into bitstreams as probabilities remarkably easily, using just an AND gate.

An AND gate is a digital circuit with two inputs and one output that gives a 1 only if both inputs are 1. It consists of just a few transistors and requires very little energy to operate. Being able to do multiplications with it—rather than, say, programming a microprocessor that contains thousands if not millions of transistors—results in enormous energy savings.

How about addition? Again, suppose we have two bitstreams representing two numbers. Let's call the probabilities of finding a 1 at any given point in those two bitstreams, respectively, p_1 and p_2. If one of these bitstreams has a value of 1 in 60 percent of the bit positions, for example, then the value it represents is 0.6. If the other has a value of 1 in 90 percent of the positions, the value it represents is 0.9. We want to generate a bitstream denoting the sum of those two values, $p_1 + p_2$. Remember that p_1 and p_2, like all probabilities, must always lie between 0 (something is impossible) and 1 (something is certain). But $p_1 + p_2$ could lie anywhere between 0 and 2, and anything greater than 1 can't be represented as a probability and thus can't be encoded as a bitstream.

To sidestep this obstacle, simply divide the quantity of interest ($p_1 + p_2$) by 2. That value can then be represented by a bitstream, one that is easy to compute: Each bit in it is just a random sample from the two input bitstreams. Half the time, a bit sampled from the first input is transferred to the output; otherwise a bit from the second input is used, effectively averaging the two inputs. The circuit that accomplishes this sampling is again a very rudimentary one, called a multiplexer. With it, addition becomes very easy.

Similarly simple circuits can carry out other arithmetic operations on these bitstreams. In contrast, conventional digital circuits require hundreds if not thousands of transistors to perform arithmetic, depending on the precision required of the results. So stochastic computing offers a way to do some quite involved mathematical manipulations using surprisingly little power.

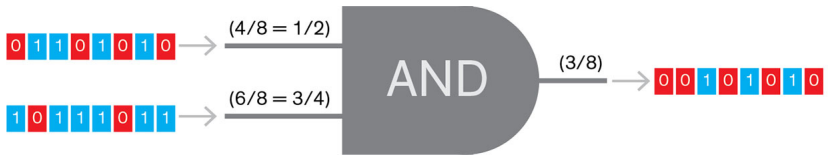

Illustration: David Schneider and Mark Montgomery

Many Times Better: Using stochastic bitstreams, multiplication can be carried out with a single AND gate. Here two bitstreams, representing 1/2 and 3/4, provide the inputs. The output has 1s in three of eight positions, meaning that it represents a value of 3/8 —the product of the two inputs.

Engineers welcomed stochastic computing when it was first developed in the 1960s because it allowed them to perform complicated mathematical functions with just a few transistors or logic gates, which at the time were rather costly. But transistors soon became much cheaper to make, and the attraction of stochastic computing quickly faded, as did solutions that involved just analog circuitry. The now-common forms of digital circuitry took off because they offered much better speed, performance, and flexibility.

But an important exception to that rule appeared in the mid-2000s, shortly after a new error-detection and error-correction scheme, low-density parity check (LDPC), started coming into widespread use. Discovered in the 1960s, LDPC codes are now used everywhere in communication systems, including Wi-Fi networks. Decoding LDPC codes can be a tricky business, however. But because the decoding involves probabilistic algorithms, it can be implemented using relatively simple stochastic computing circuits.

The success of stochastic circuits in that context, and the fact that controlling power use has now become one of the biggest challenges facing chip designers, prompted us and other researchers to revisit stochastic computing several years ago. We wanted to see what else it could do in the modern electronic era.

It turns out there is quite a lot. Apart from saving power, stochastic computing also offers a unique property known as progressive precision. That's because, with this technique, the precision of the calculations depends on the length of the bitstream you use. For example, suppose you're using 0110101010010111 to represent the fraction 9/16 (nine 1s in 16 possible bit positions). With stochastic computing, the leftmost digits are processed first, and all bits have equal significance or weight. If you look at the first eight bits of this example, 01101010, you get the number 4/8, which is a low-precision estimate of the value represented by the longer sequence.

The circuits that are used to process stochastic bitstreams act as though they are computing with the most significant digits of the number first. Conventional digital circuits—or paper-and-pencil calculations—work the other way, from the least to the most significant digits. When a normal computer adds two binary numbers together, the first bits computed don't provide any sort of early approximation of the overall result.

Stochastic computing circuits, on the other hand, do exactly that: Their progressive-precision property means that the answer is pretty good at the start and tends to get increasingly precise as more and more bits flow through the circuit. So a computation can be ended as soon as enough bits have emerged in the results, leading to significant energy savings.

How many bits is enough? That depends on the application, and those that demand high precision will, of course, require longer bitstreams—perhaps hundreds or even thousands of bits.

There are limits to the precision you can achieve in practice, though. That's because to represent an *n*-bit binary number, stochastic computing requires the length of the bitstream to be at least 2^n. Take the case of 8-bit numbers, of which there are 256 possible values. Suppose you wanted to represent the probability 1/256 with a bitstream. You'd need a bitstream that is at the very least 256 bits long—otherwise there wouldn't be a place for a lone 1 in a sea of 0s. Similarly, to represent 9-bit numbers, you'd need streams of at least 512 bits. For 10-bit numbers, the requirement would be for 1,024 bits, and so on. Clearly, the numbers get large fast. Achieving even what is known in computer programming circles as single precision (32 bits) would be nearly impossible, because it would require streams of *billions* of bits to be manipulated.

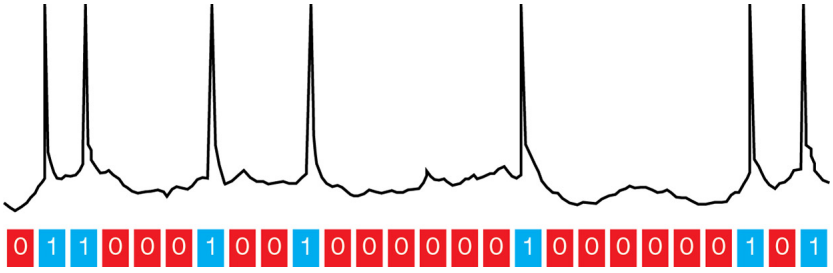

Illustration: David Schneider and Mark Montgomery

Bitstream Brains: Neural signals resemble bitstreams in that frequent spikes indicate high values of neural activity, just as frequent 1s in a bitstream indicate high values for the number that it represents.

Low in precision as it is, stochastic computing is curiously similar to what goes on inside our brains. Our neural pathways encode their signals primarily by the rate or frequency of sharp electrical pulses or "spikes." When those spikes are few and far between, the activity of the neural pathway is said to be low; when they occur frequently, the activity level is high. Similarly, when the 1s in a bitstream are few and far between, the stream corresponds to a low number; when they are common, it encodes a high number.

Also, stochastic computing circuits, like many biological systems, are resilient in the face of many kinds of disturbances. If, for example, a source of environmental noise causes some of the binary digits in a bitstream to flip, the number represented by that bitstream won't change significantly: Often there will be as many 1s that change to 0s as there are 0s that change to 1s, so the noise will just average out over time.

These similarities with biological systems weren't lost on us when we began our research. And we had them in mind when we began looking into an exciting new application for stochastic computing—processing signals in retinal implants.

Retinal implants are intended to restore sight to people with severe macular degeneration, retinitis pigmentosa, and other degenerative diseases of the retina. Although using electronics to restore lost vision is an old idea, the actual clinical use of retinal implants is less than a decade old, and it's been attempted with comparatively few patients because the technology remains so rudimentary.

Most retinal implants capture and process images outside the eye using a camera and a digital computer. That's pretty clunky. And it gives patients an odd sense when they move their eyes and find that the image projected on their retinas doesn't move in the way their brains expect. What you really want, of course, is for the image sensing and processing to take place *inside the eye*. One hurdle to accomplishing this is that there's little power available inside the eye to operate the electronics—the only power sources available are tiny inductive pick-up coils or photovoltaic cells. And you need relatively large amounts of power to sense and process images using conventional digital circuits. Even if a source of sufficient power were available, using it would still be problematic because excessive power dissipation can harm eye tissues, which can tolerate only a few degrees of temperature rise.

For these reasons, we figured that the simplicity and efficiency of stochastic computing could make a big difference. To test this idea, we conducted a little experiment. We designed several stochastic image-processing circuits, including one that detects edges in images. (Edge detection improves contrast, making objects easier to perceive.) Not surprisingly, the stochastic circuit we designed for this purpose is much simpler and more efficient in its use of power than the kinds of digital circuits typically used for edge detection.

Another biologically inspired application of stochastic computing is in artificial neural networks, which lie at the heart of many of today's smart systems. We explored this application recently using an image sensor connected to such a neural network, one configured to recognize digits after it has been trained to do so—meaning that its many adjustable parameters have been set at values that allow it to classify the images presented to it as a specific digit. Neural networks are arranged in a series of layers of artificial neurons, where the output of one layer serves as the input to the next. In our experiments, we replaced the first processing layer of our network with stochastic circuitry.

Although the stochastic circuitry sometimes gave inaccurate arithmetic results, it did not matter because neural networks can learn to tolerate such errors. So we just retrained our neural network to deal with the stochastic errors. In this way, we could reduce the energy used in the first layer of the network by a factor of 10, while pretty much preserving the original level of accuracy in digit classification.

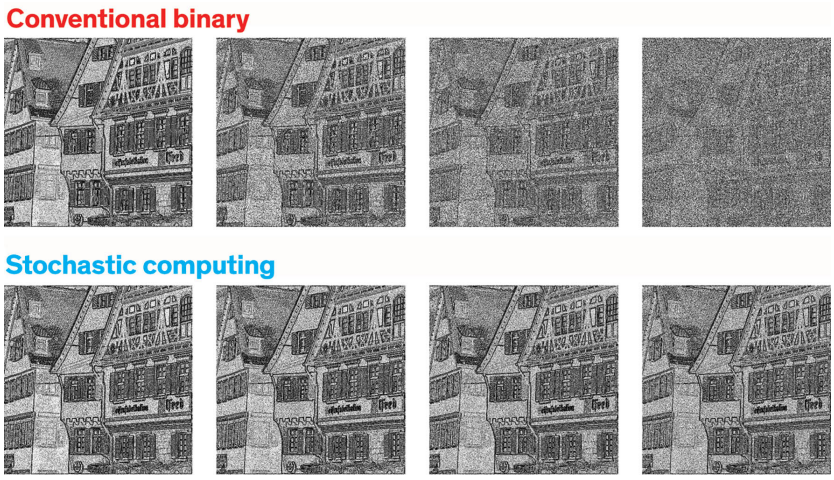

Image: Armin Alaghi

Always on Edge: Edge detection is commonly used in image processing. Here, an edge-detection algorithm that uses conventional binary numbers [top row] is compared with one that uses stochastic bitstreams [bottom row]. The stochastic results hold up much better as the bit-error rate is increased from 0.1 percent [far left] to 0.5 percent [middle left] to 1.0 percent [middle right] and finally to 2.0 percent [far right].

One of the things holding stochastic computing back has been the lack of any comprehensive design methodology. Sure, it's easy enough to design circuits for simple arithmetic operations such as multiplication and addition, but when the target function is more complicated, engineers have long been without a good road map.

A decade ago, Weikang Qian and Marc Riedel, of the University of Minnesota, devised a novel technique to solve this problem. Building on their work, we recently discovered another approach to designing stochastic computing circuits. It begins with the observation that a stochastic circuit corresponds to a Boolean function. AND, OR, NAND, and NOR are all examples of Boolean functions. More generally, they are defined as a mathematical function that takes some number of different inputs (each of which can be 0 or 1) and produces one output, which, depending on the input values, will be 0 or 1.

Suitable mathematical transformations applied to that Boolean function—ones similar to those used to determine, for example, the frequency content of audio signals—reveal how the stochastic circuit will operate on bitstreams, whether it will serve as a multiplier, say, or an adder. We found that you can go the other way, too. You can start with the desired function and perform those mathematical transformations in reverse to deduce the circuit needed.

Based on that observation, we developed a method that enabled us to design efficient stochastic computing circuits for image processing, including one that could carry out a common image-processing function called gamma correction. (Gamma correction is used to account for the insensitivity of the human eye to small differences in brightness in lighter areas of an image.) With this strategy, we were able to design a small (eight gate) circuit that implements the gamma-correction function.

Efficient as they are, stochastic circuits can be made even more so when combined with a power-reduction technique known as voltage scaling. That's basically a highfalutin way of saying that you dial the voltage way down to save energy at the cost of creating occasional errors. That's not much of a problem for stochastic circuits, which can work acceptably well at voltages that would be too low for conventional ones. For example, the gamma-correction circuit we built can tolerate a voltage reduction up to 40 percent, from 1 volt down to 0.6 V, with no loss of accuracy. And unlike conventional binary circuits, which fail catastrophically when the voltage scaling is too aggressive, stochastic circuits continue to operate, albeit with less precision, as the voltage is reduced.

While our examination of circuits for retinal implants and neural networks makes us very optimistic about the prospects for stochastic computing, we still haven't discovered the real killer app for this approach. It may be 50 years old, but stochastic computing, in our view, is still in its infancy.

This article appears in the March 2018 print issue as "Computing With Randomness."