COMPUTERS WITHOUT CLOCKS

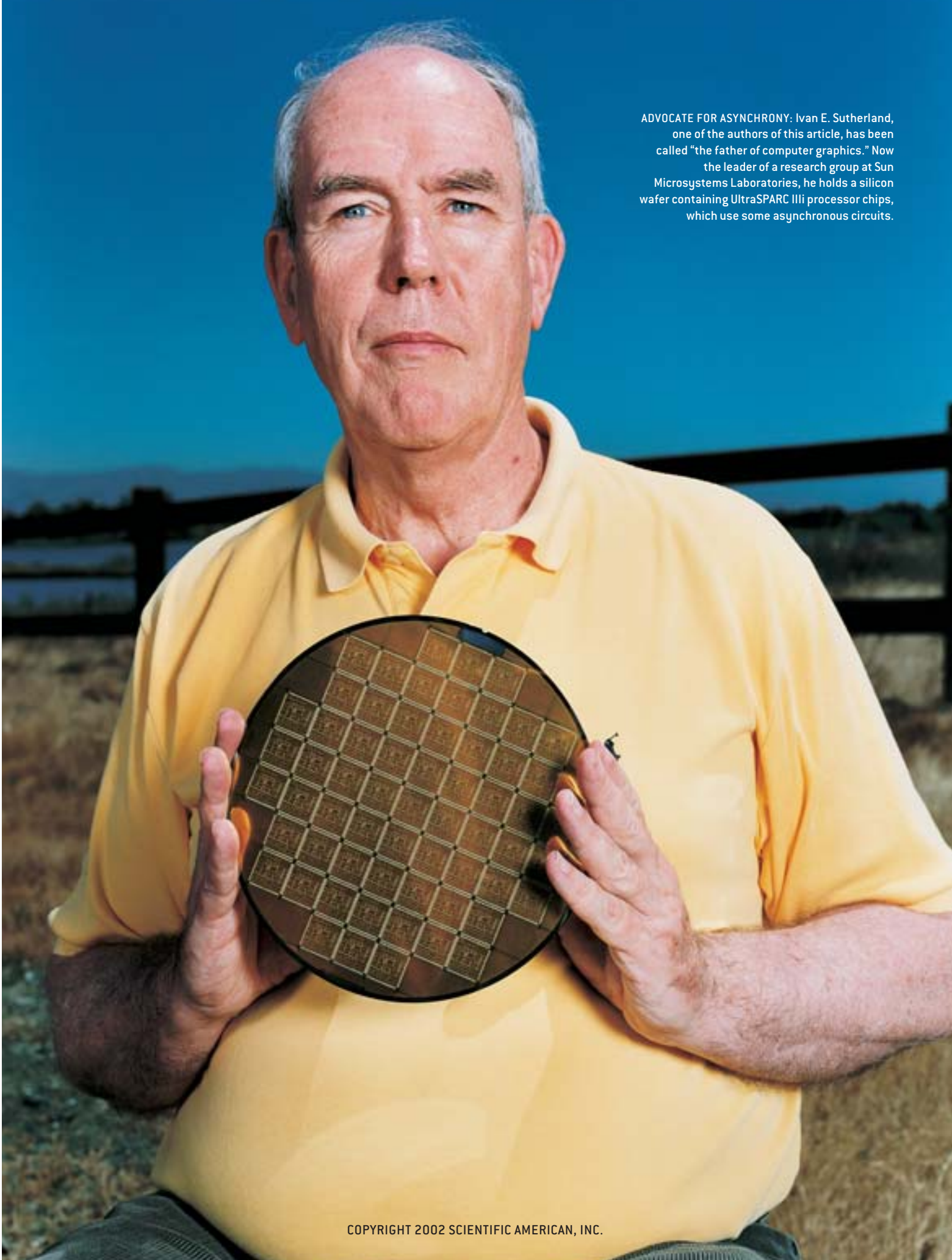
ASYNCHRONOUS CHIPS IMPROVE COMPUTER PERFORMANCE BY LETTING EACH CIRCUIT RUN AS FAST AS IT CAN

By Ivan E. Sutherland and Jo Ebergen

How fast is your personal computer?

When people ask this question, they are typically referring to the frequency of a minuscule clock inside the computer, a crystal oscillator that sets the basic rhythm used throughout the machine. In a computer with a speed of one gigahertz, for example, the crystal "ticks" a billion times a second. Every action of the computer takes place in tiny steps, each a billionth of a second long. A simple transfer of data may take only one step; complex calculations may take many steps. All operations, however, must begin and end according to the clock's timing signals.

OLIVIER LAUDE



ADVOCATE FOR ASYNCHRONY: Ivan E. Sutherland, one of the authors of this article, has been called "the father of computer graphics." Now the leader of a research group at Sun Microsystems Laboratories, he holds a silicon wafer containing UltraSPARC IIIi processor chips, which use some asynchronous circuits.

Because most modern computers use a single rhythm, we call them synchronous. Inside the computer's microprocessor chip, a clock distribution system delivers the timing signals from the crystal oscillator to the various circuits, just as sound in air delivers the beat of a drum to soldiers to set their marching pace. Because all parts of the chip share the same rhythm, the output of any circuit from one step can serve as the input to any other circuit for the next step. The synchronization provided by the clock helps chip designers plan sequences of actions for the computer.

The use of a central clock also creates problems. As speeds have increased, distributing the timing signals has become more and more difficult. Present-day transistors can process data so quickly that they can accomplish several steps in the time that it takes a wire to carry a signal from one side of the chip to the other. Keeping the rhythm identical in all parts of a large chip requires careful design and a great deal of electrical power. Wouldn't it be nice to have an alternative?

Our research group at Sun Microsystems Laboratories seeks such alternatives. Along with several other groups worldwide, we are investigating ways to design computing systems in which each part can proceed at its own pace instead of depending on the rhythm of a central clock. We call such systems asynchronous. Each part of an asynchronous system may extend or shorten the timing of its steps when necessary, much as a hiker takes long or short steps when walking across rough terrain. Some of the pioneers of the computer age, such as mathematician Alan M. Turing, tried using asynchronous designs to build machines in the early 1950s. Engineers soon abandoned this approach in favor of synchronous computers because common timing made the design process so much easier.

Now asynchronous computing is experiencing a renaissance. Researchers at the University of Manchester in England, the University of Tokyo and the California Institute of Technology have demonstrated asynchronous microprocessors. Some asynchronous chips are already in commercial mass production. In the late 1990s Sharp, the Japanese electronics company, used asynchronous design to build a data-driven media processor—a chip for editing graphics, video and audio—and Philips Electronics produced an asynchronous microcontroller for two of its pagers. Asynchronous parts of otherwise synchronous systems are also beginning to appear; the UltraSPARC IIIi processor recently introduced by Sun includes some asynchronous circuits developed by our group. We believe that asynchronous systems will become ever more popular as researchers learn how to exploit their benefits and develop methods for simplifying their design. Asynchronous chip makers have achieved a good measure of technical success, but commercial success is still to come. We remain a long way from fulfilling the full promise of asynchrony.

Overview/*Clockless Systems*

- Most modern computers are synchronous: all their operations are coordinated by the timing signals of tiny crystal oscillators within the machines. Now researchers are designing asynchronous systems that can process data without the need for a governing clock.
- Asynchronous systems rely on local coordination circuits to ensure an orderly flow of data. The two most important coordination circuits are called the Rendezvous and the Arbiter.
- The potential benefits of asynchronous systems include faster speeds, lower power consumption and less radio interference. As integrated circuits become more complex, chip designers will need to learn asynchronous techniques.

Beat the Clock

WHAT ARE THE POTENTIAL BENEFITS of asynchronous systems? First, asynchrony may speed up computers. In a synchronous chip, the clock's rhythm must be slow enough to accommodate the slowest action in the chip's circuits. If it takes a billionth of a second for one circuit to complete its operation, the chip cannot run faster than one gigahertz. Even though many other circuits on that chip may be able to complete their operations in less time, these circuits must wait until the clock ticks again before proceeding to the next logical step. In contrast, each part of an asynchronous system takes as much or as little time for each action as it needs. Complex operations can take more time than average, and simple ones can take less. Actions can start as soon as the prerequisite actions are done, without waiting for the next tick of the clock. Thus, the system's speed depends on the average action time rather than the slowest action time.

Coordinating asynchronous actions, however, also takes time and chip area. If the efforts required for local coordination are small, an asynchronous system may, on average, be faster than a clocked system. Asynchrony offers the most help to irregular chip designs in which slow actions occur infrequently.

Asynchronous design may also reduce a chip's power consumption. In the current generation of large, fast synchronous chips, the circuits that deliver the timing signals take up a good chunk of the chip's area. In addition, as much as 30 percent of the electrical power used by the chip must be devoted to the clock and its distribution system. Moreover, because the clock is always running, it generates heat whether or not the chip is doing anything useful.

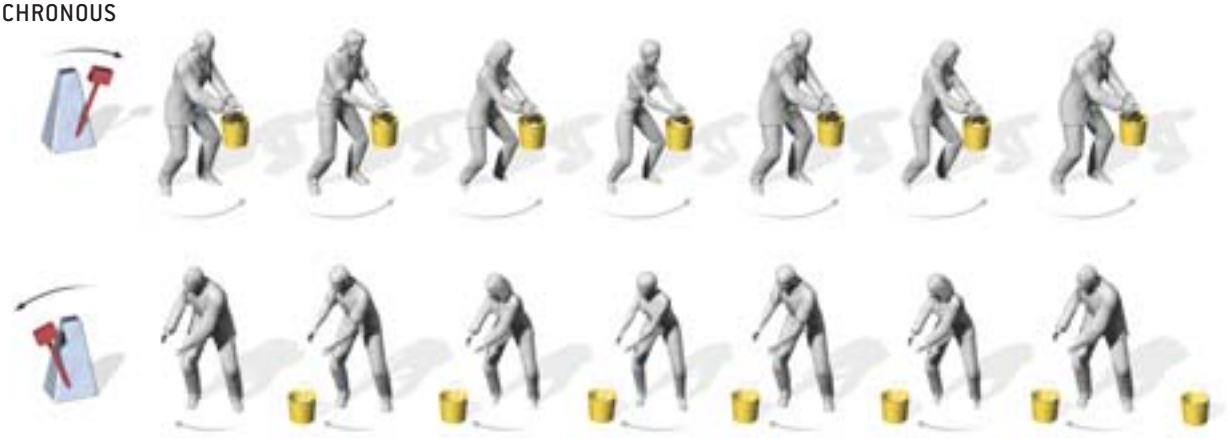
In asynchronous systems, idle parts of the chip consume negligible power. This feature is particularly valuable for battery-powered equipment, but it can also cut the cost of larger systems by reducing the need for cooling fans and air-conditioning to prevent them from overheating. The amount of power saved depends on the machine's pattern of activity. Systems with parts that act only occasionally benefit more than systems that act continuously. Most computers have components, such as the floating-point arithmetic unit, that often remain idle for long periods.

Furthermore, asynchronous systems produce less radio interference than synchronous machines do. Because a clocked system uses a fixed rhythm, it broadcasts a strong radio signal

BUCKET BRIGADES

THE METAPHOR of the bucket brigade can be used to describe the flow of data in a computer. A synchronous computer system is like a bucket brigade in which each person follows the ticktock rhythm of a clock. When the clock ticks, each person pushes a bucket forward to the next person down the line (*top*). When the clock tocks, each person grasps the bucket pushed forward by the preceding person (*middle*). An asynchronous system, in contrast, is like an ordinary bucket brigade: each person who holds a bucket can pass it down the line as soon as the next person's hands are free (*bottom*). —*I.E.S. and J.E.*

SYNCHRONOUS



ASYNCHRONOUS

at its operating frequency and at the harmonics of that frequency. Such signals can interfere with cellular phones, televisions and aircraft navigation systems that operate at the same frequencies. Asynchronous systems lack a fixed rhythm, so they spread their radiated energy broadly across the radio spectrum, emitting less at any one frequency.

Yet another benefit of asynchronous design is that it can be used to build bridges between clocked computers running at different speeds. Many computing clusters, for instance, link fast PCs with slower machines. These clusters can tackle complex problems by dividing the computational tasks among the PCs. Such a system is inherently asynchronous: different parts march to different beats. Moving data controlled by one clock to the control of another clock requires asynchronous bridges, because the data may be "out of sync" with the receiving clock.

Finally, although asynchronous design can be challenging, it can also be wonderfully flexible. Because the circuits of an asynchronous system need not share a common rhythm, designers have more freedom in choosing the system's parts and determining how they interact. Moreover, replacing any part with a faster version will improve the speed of the entire system. In contrast, increasing the speed of a clocked system usually requires upgrading every part.

Local Cooperation

TO DESCRIBE HOW ASYNCHRONOUS systems work, we often use the metaphor of the bucket brigade. A clocked system is like a bucket brigade in which each person must pass and receive buckets according to the ticktock rhythm of the clock. When the clock ticks, each person pushes a bucket forward to the next person down the line; when the clock tocks, each person grasps the bucket pushed forward by the preceding person [*see illustrations above*]. The rhythm of this brigade cannot go faster than the time it takes the slowest person to move the heaviest bucket. Even if most of the buckets are light, everyone in the line must wait for the clock to tick before passing the next bucket.

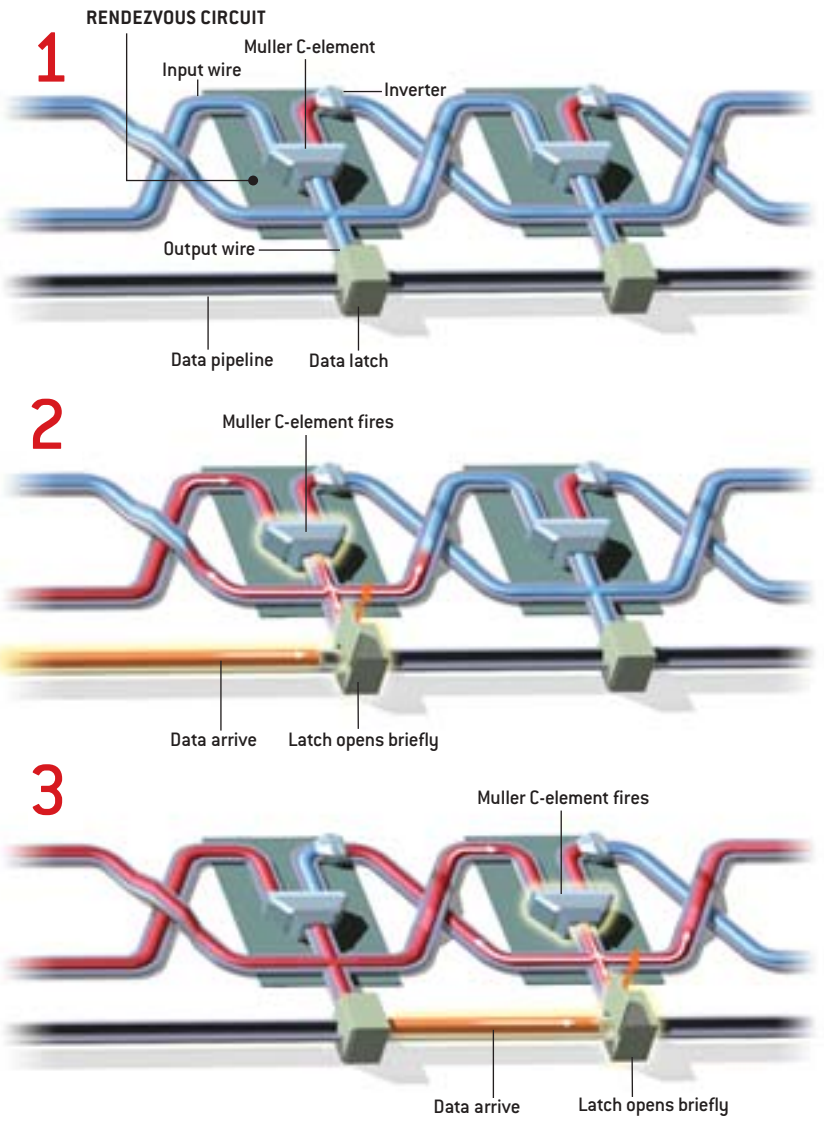
An asynchronous bucket brigade is governed by local cooperation rather than a common clock. Each person who holds a bucket can pass it to the next person down the line as soon as the next person's hands are free. Before each action, one person may have to wait until the other is ready. When most of the

BRYAN CHRISTIE DESIGN

HOW A RENDEZVOUS CIRCUIT WORKS

RENDEZVOUS CIRCUITS can coordinate the actions of an asynchronous system, allowing data to flow in an orderly fashion without the need for a central clock. Shown here is an electronic pipeline controlled by a chain of Muller C-elements, each of which allows data to pass down the line only when the preceding stage is "full"—indicating that data are ready to move—and the following stage is "empty."

Each Muller C-element has two input wires and one output wire. The output changes to FALSE when both inputs are FALSE and back to TRUE when both inputs are TRUE. (In the diagram, TRUE signals are shown in blue and FALSE signals in red.) The inverter makes the initial inputs to the Muller C-element differ, setting all stages empty at the start. Let's assume that the left input is initially TRUE and the right input FALSE (*1*). A change in signal at the left input from TRUE to FALSE (*2*) indicates that the stage to the left is full—that is, some data have arrived. Because the inputs to the Muller C-element are now the same, its output changes to FALSE. This change in signal does three things: it moves data down the pipeline by briefly making the data latch transparent, it sends a FALSE signal back to the preceding C-element to make the left stage empty, and it sends a FALSE signal ahead to the next C-element to make the right stage full (*3*). —*I.E.S. and J.E.*



buckets are light, however, they can move down the line very quickly. Moreover, when there's no water to move, everyone can rest between buckets. A slow person will still hinder the performance of the entire brigade, but replacing the slowpoke will return the system to its best speed.

Bucket brigades in computers are called pipelines. A common pipeline executes the computer's instructions. Such a pipeline has half a dozen or so stages, each of which acts like a person in a bucket brigade. Instead of handling buckets of water, however, each stage performs one action of the instruction's execution. For example, a processor executing the instruction "ADD A B C" must fetch the instruction from memory, decode the instruction, get the numbers from addresses A and B in memory, do the addition and store the sum in memory address C. A clocked pipeline executes these actions in a rhythm independent of the operations performed or the size of the numbers. Adding one and one may take just as much time as adding two 30-digit numbers. In an asynchronous pipeline, though, the duration of each action may depend on the operation performed, the size of the numbers and the locations of the data in memory (just as in a bucket brigade, the amount of water in a bucket may determine how long it takes to pass it on).

Without a clock to govern its actions, an asynchronous system must rely on local coordination circuits instead. These circuits exchange completion signals to ensure that the actions at each stage begin only when the circuits have the data they need. The two most important coordination circuits are called the Rendezvous and the Arbiter.

BRYAN CHRISTIE DESIGN

A Rendezvous element indicates when the last of two or more signals has arrived at a particular stage. Asynchronous systems use these elements to wait until all the concurrent actions finish before starting the next action. For instance, an arithmetic division circuit must have both the dividend (say, 16) and the divisor (say, 2) before it can divide one by the other (to reach the answer 8).

One form of Rendezvous circuit is called the Muller C-element, named after David Muller, now retired from a professorship at the University of Illinois. A Muller C-element is a logic circuit with two inputs and one output [*see box on opposite page*]. When both inputs of a Muller C-element are TRUE, its output becomes TRUE. When both inputs are FALSE, its output becomes FALSE. Otherwise the output remains unchanged. For the Muller C-element to act as a Rendezvous circuit, its inputs must not change again until its output responds. A chain of Muller C-elements can control the flow of data down an electronic bucket brigade.

Our research group recently introduced a new kind of Rendezvous circuit called GasP [*see box on next page*]. GasP evolved from an earlier family of circuits designed by Charles E. Molnar, our late colleague at Sun Microsystems. Molnar dubbed his creation asP*, which stands for asynchronous symmetric pulse protocol (the asterisk indicates the double "P"). We added the "G" to the name because GasP is what you are supposed to do when you see how fast our new circuits go. We have found that GasP modules are as fast and as energy-efficient as Muller C-elements, fit better with ordinary data latches and offer much greater versatility in complex designs.

Without a clock to govern its actions, an asynchronous system must rely on local coordination circuits instead.

Buridan's Ass

AN ARBITER CIRCUIT performs another task essential for asynchronous computers. An Arbiter is like a traffic officer at an intersection who decides which car may pass through next. Given only one request, an Arbiter promptly permits the corresponding action, delaying any second request until the first action is completed. When an Arbiter gets two requests at once, it must decide which request to grant first. For example, when two processors request access to a shared memory at approximately the same time, the Arbiter puts the requests into a sequence, granting access to only one processor at a time. The Arbiter guarantees that there are never two actions under way at once, just as the traffic officer prevents accidents by ensuring that there are never two cars passing through the intersection on a collision course.

Although Arbiter circuits never grant more than one request at a time, there is no way to build an Arbiter that will always reach a decision within a fixed time limit. Present-day Arbiters reach decisions very quickly on average, usually within about a few hundred picoseconds. (A picosecond is a trillionth of a second.) When faced with close calls, however, the circuits may occasionally take twice as long, and in very rare cases the time needed to make a decision may be 10 times as long as normal.

The fundamental difficulty in making these decisions is nicely illustrated by the parable of Buridan's ass. Attributed to Buridan, a 14th-century French philosopher, this parable suggests that an ass placed exactly between two equal piles of hay might starve to death because it would be unable to choose which pile to eat. Similar minor dilemmas are familiar in everyday life. For example, two people approaching a doorway at the same time may pause before deciding who will go through first. They can go through in either order, and Buridan's ass can eat from either pile of hay. In both cases, all that is needed is a way to break the tie.

An Arbiter breaks ties. Like a flip-flop circuit, an Arbiter has two stable states corresponding to the two choices. One can think of these states as the Pacific Ocean and the Gulf of Mex-

THE AUTHORS

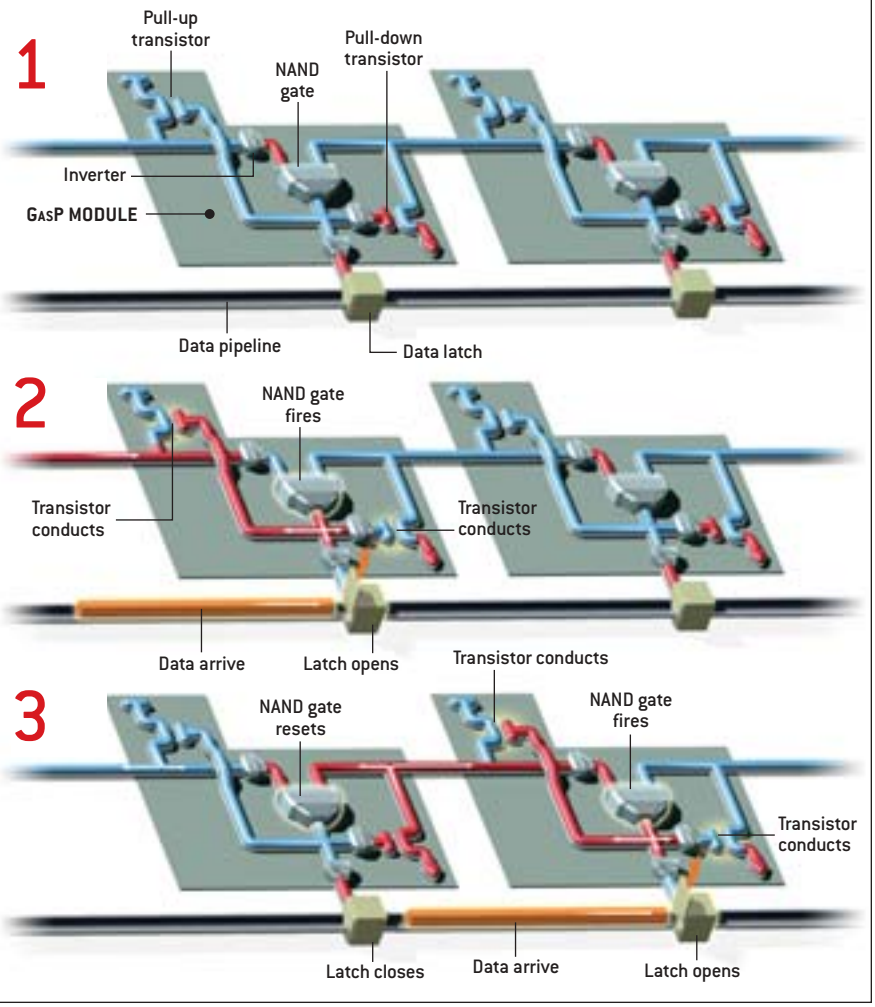
IVAN E. SUTHERLAND and *JO EBERGEN* are true believers in asynchronous computing. Although Sutherland (*middle left*) is best known as a pioneer of computer graphics—he invented the interactive graphics program Sketchpad in 1963—he became involved in asynchronous circuit design in the mid-1960s while building a graphics processor at Harvard University. He is now a vice president and fellow at Sun Microsystems, leading the Asynchronous Design Group at the company's laboratories. Ebergen (*middle right*) became fascinated by asynchronous circuit design 20 years ago during a three-month stint as a research assistant to Charles L. Seitz of Caltech. He subsequently taught at Eindhoven University of Technology in the Netherlands and the University of Waterloo in Canada before joining Sun's Asynchronous Design Group in the summer of 1996.

OLIVIER LAUDE

HOW A GASP MODULE WORKS

GASP MODULES can also act as Rendezvous elements for an asynchronous data pipeline. (The "P" in GasP is capitalized to acknowledge an earlier family of circuits.) Each GasP module has two wires connecting it to its neighbors and an output wire that drives a data latch. At the heart of the module is a NAND gate, which produces a FALSE output only when both inputs are TRUE. Otherwise the NAND produces a TRUE output.

The connection wires between modules represent the stages in the pipeline. At the start, all the signals in these wires are TRUE (*blue*), indicating that the stages are empty (*1*). The arrival of data in the pipeline (*2*) changes the incoming wire's signal to FALSE (*red*). An inverter flips the signal to TRUE and sends it to the NAND gate, which changes its output to FALSE. This makes the data latch transparent, allowing the data to move down the pipeline. The FALSE output also drives the incoming wire back to the TRUE state (which means empty) via a pull-up transistor and drives the outgoing wire to the FALSE state (which means full) via an inverter and a pull-down transistor. The NAND gate's output then returns to TRUE (*3*), making the latch opaque again. Meanwhile the FALSE signal in the outgoing wire triggers the same process in the next GasP module. —*I.E.S. and J.E.*



ico. Each request to an Arbiter pushes the circuit toward one stable state or the other, just as a hailstone that falls in the Rocky Mountains can roll downhill toward the Pacific or the Gulf. Between the two stable states, however, there must be a meta-stable line, which is equivalent to the Continental Divide. If a hailstone falls precisely on the divide, it may balance momentarily on that sharp mountain ridge before tipping toward the Pacific or the Gulf. Similarly, if two requests arrive at an Arbiter within a few picoseconds of each other, the circuit may pause in its meta-stable state before reaching one of its stable states to break the tie.

Novice Arbiter designers often seek to avoid even the occasional long delay by fashioning complicated circuits. A common error involves a circuit that notices the "hung" meta-stable state and pushes the Arbiter toward a particular decision. This is like training Buridan's ass to go left when it notices a hard choice. Such training, however, merely gives the ass three choices rather than two: go left, go right, or notice a hard choice and therefore go left. Even a trained ass will starve when unable to decide between the last two choices. Or, to use the geographic metaphor, you can move the Continental Divide with a shovel, but you cannot get rid of it. Although there is no way to eliminate meta-stability, simple, well-designed Arbiter circuits can ensure that virtually all delays are very brief. A typical contemporary Arbiter has a normal delay of 100 picoseconds and experiences a delay of 400 picoseconds less than once every 10 hours of operation. The probability of delays decreases exponentially with the length of the delay: an 800-picosecond pause occurs less than once every billion years of operation.

The Need for Speed

OUR GROUP AT SUN MICROSYSTEMS concentrates on designing fast asynchronous systems. We have found that speed often comes from simplicity. Our initial goal was to build a counterflow pipeline with two opposing data flows—like two parallel bucket brigades moving in opposite directions. We wanted the data from both flows to interact at each of the stages; the hard challenge was to ensure that every "northbound" data element would interact with every "southbound" data element. Arbitration turned out to be essential. At each

BRYAN CHRISTIE DESIGN

joint between successive stages, an Arbiter circuit permitted only one element at a time to pass. This project proved very useful as a research target; we learned a great deal about coordination and arbitration and built test chips to prove the reliability of our Arbiter circuits.

More recently, we have chosen a fresh research target, a processing structure we call FLEET. The name refers not only to speed but also to the collection of computing elements, each of which we call a ship. Each ship does its task concurrently with the others; the FLEET system controls their actions by moving data among them through an asynchronous switching network. Our work on FLEET has led to many discoveries. We wanted speed, and that led us to create the basic GasP circuits. We wanted to steer data items from one pipeline to another, like cars at highway interchanges, and that led us to design a larger family of GasP circuits that can act like a dense system of freeways. These circuits can move data about twice as fast as a clocked system could. To gauge the speed of our switching networks, we often build rings on our test chips around which the data elements rush like race cars. We measure the time it takes a data element to complete a round-trip, which is a lot easier than measuring the much shorter time it takes data to advance through one stage.

Our designs are beginning to enter Sun's computer products. The UltraSPARC IIIi processor chip contains asynchronous data queues that accept information from memory chips [*see illustration on page 63*]. This asynchronous system is the simplest and fastest way to compensate for the differences in arrival time of signals from memory chips that lie at different distances from the processor. Sun's product designers are gaining confidence that they can build asynchronous parts, that the parts work and can be tested, and that asynchrony offers important advantages over clocked design. As their confidence grows, more and more commercial products will use asynchronous parts for greater speed and flexibility, better power efficiency and reduced radio interference.

Sun is by no means the only company investigating asynchronous circuits. A group at Philips Research in the Netherlands has developed an asynchronous error corrector for a digital compact cassette player and an asynchronous version of a popular microcontroller for portable devices. The asynchronous microcontroller has been incorporated into pagers sold by Philips. The success of the Philips research group comes from three factors. First, the team learned how to create products rapidly using a programming language called Tangram that simplifies hardware design. Second, the low power consumption of their asynchronous circuits enables a pager to operate longer on a battery charge. Third, the reduced radio interference of their asynchronous circuits allowed the inclusion of both a computer and a sensitive radio receiver in a tiny device.

Furthermore, the experiments at Manchester, Caltech and Philips demonstrate that asynchronous microprocessors can be compatible with their clocked counterparts. The asynchronous processors can connect to peripheral machines without special programs or interface circuitry.

A Challenging Time

ALTHOUGH THE ARCHITECTURAL freedom of asynchronous systems is a great benefit, it also poses a difficult challenge. Because each part sets its own pace, that pace may vary from time to time in any one system and may vary from system to system. If several actions are concurrent, they may finish in a large number of possible sequences. Enumerating all the possible sequences of actions in a complex asynchronous chip is as difficult as predicting the sequences of actions in a schoolyard full of children. This dilemma is called the state explosion problem. Can chip designers create order out of the potential chaos of concurrent actions?

The technological trend is inevitable: in the coming decades, asynchronous design will become prevalent.

Fortunately, researchers are developing theories for tackling this problem. Designers need not worry about all the possible sequences of actions if they can set certain limitations on the communication behavior of each circuit. To continue the schoolyard metaphor, a teacher can promote safe play by teaching each child how to avoid danger.

Another difficulty is that we lack mature design tools, accepted testing methods and widespread education in asynchronous design. A growing research community is making good progress, but the present total investment in clock-free computing pales in comparison with the investment in clocked design. Nevertheless, we are confident that the relentless advances in the speed and complexity of integrated circuits will force designers to learn asynchronous techniques. We do not know yet whether asynchronous systems will flourish first within large computer and electronics companies or within start-up companies eager to develop new ideas. The technological trend, however, is inevitable: in the coming decades, asynchronous design will become prevalent. Eventually there will no longer be an easy answer to the question, How fast is your personal computer? SA

MORE TO EXPLORE

Micropipelines. Ivan E. Sutherland: The Turing Award Lecture. *Communications of the ACM,* Vol. 32, No. 6, pages 720–738; June 1989.

Asynchronous Circuits and Systems. Special issue of *Proceedings of the IEEE;* February 1999.

Principles of Asynchronous Circuit Design: A Systems Perspective. Edited by Jens Sparsø and Steve Furber. Kluwer Academic Publishers, 2001.

The Asynchronous Logic Home Page is at **www.cs.man.ac.uk/async/**

The Asynchronous Bibliography is at **www.win.tue.nl/~wsinap/async.html**