Asynchronous Design Shows Promise

But Lack of Tools and Skilled Designers Keeps Circuits Small and Simple

by Peter Song

Tick, tick, tick. The continuous beat of the ubiquitous clock signal is a comforting sound to digital designers, who know their signals have to be valid when the clock ticks but not between ticks. A growing number of designers, however, are making the scary leap to designs that do not use a clock. These asynchronous designs consume little power because transistors switch only when needed, not every cycle, and can provide performance benefits as well.

Asynchronous circuits are energy efficient—especially in CMOS technology, which consumes power only when switching—since only the transistors involved in the current computation are switched. They are well suited for algorithms that exhibit large differences between worst-case and average-case computing times. Without a clock to periodically generate huge voltage swings, asynchronous chips produce lower levels of electromagnetic interference. They are naturally immune from clock skew, a growing problem in clocked circuits as clock periods shrink and die sizes increase.

Asynchronous circuits are, unfortunately, much more difficult to design than clocked, or synchronous, circuits. Designing synchronous circuits is made easier by assuming that all state-holding elements (latches and registers) change together and in discontinuous steps. Assuming otherwise—that state-holding elements change independently or continuously—increases the need for highly accurate timing models, greatly complicates the design process, and requires highly inefficient test methods. The few designers who have tried it have found that the lack of simple and well understood design methods and tools gets in the way of reaping the full benefits of being clock-free.

Class	State change		Delay	
	Individual	System	Gate	Wire
Synchronous	Discrete	Synch	Bounded	Bounded
Micropipelines	Discrete	Async	Bounded datapath, unbounded control	Bounded datapath, unbounded control
Burst AFSM*	Discrete	Async	Bounded	Bounded
SIC† AFSM	Continuous	Async	Bounded	Bounded
Speed insensitive	Continuous	Async	Unbounded	Bounded (negligible)
Quasi-delay insensitive	Continuous	Async	Unbounded	Unbounded isochronic‡
Delay insensitive	Continuous	Async	Unbounded	Unbounded

Table 1. Sequential circuits can be classified by their assumptions about whether state changes are continual or discrete and synchronized or not and whether gate and wire delays are bounded (known) or unbounded (unknown). *AFSM = asynchronous finite-state machine. ‡isochronic = all wires originating from the same output have the same amount of delay. †SIC (single input change) = only one input is to change at a time. (Source: [1]).

Clock-free chips are unlikely to displace clocked chips to any great extent in the foreseeable future, even in applications that demand the highest MIPS/W rating. Nevertheless, the pursuit of reliable and easy-to-use asynchronous design methods and tools continues, mostly in research environments. A few designs are starting to make the transition from research to products. Using an asynchronous ARM core, Amulet2e delivers 40 MIPS at 150mW and targets portable embedded applications. Sharp's DDMP (data-driven multimedia processor) uses eight CPUs—each clock-free CPU delivers 600 MOPS using only 40K gates and less than 60 mW—to compete against Chromatic's Mpact2, Philips's Trimedia, and TI's TMS320C6X. And Cogency (see MPR 10/6/97, p. 10) is now shipping a fully asynchronous DSP.

Eliminating Clock Has Benefits

Unlike many synchronous circuits that consume power every cycle everywhere, asynchronous circuits consume power only when and where it is needed. Although asynchronous circuits tend to use more transistors than synchronous circuits, the transistors not involved in computation do not switch. Philips designed a DCC (digital compact cassette) error-corrector chip that consumes 80% less power than a clocked design, despite using 70–100% more area for replacing a clock and registers with handshake components. Although many synchronous designs use power-saving techniques, such as not clocking idle functional units, these techniques are inherently part of asynchronous circuits.

Asynchronous circuits are well suited for implementing algorithms such as the ripple-carry addition, in which the worst-case computing delay—the carry rippling from the least significant bit of the sum to the most significant bit—rarely occurs. Although ripple-carry adders are smaller than other types of adders and produce fast results in most cases, they are not used in clocked designs in which results must be produced in the same number of cycles in all cases. Asynchronous designs can be optimized for average-case performance rather than for worst-case performance, resulting in simpler and smaller circuits without unduly compromising overall performance.

Clocked circuits must handle clock skew (the difference in arrival times of clock edges at different parts of the circuit) that bounds the minimum and maximum delay through combinational paths. Since wire delays do not scale linearly with decreasing clock periods, clock skew takes an increasingly larger portion of useful clock cycles as process geometries shrink. Furthermore, as more transistors are packed

into larger die, requiring larger clock-distribution trees or grids and stronger clock drivers, managing clock skew and power consumed by the clock becomes even more difficult. By replacing a global clock with handshake components that are local and operate independently, asynchronous circuits avoid these problems.

Assumptions on Time Classify Sequential Circuits

The fundamental distinction between synchronous and asynchronous circuits is whether states change together (synchronously) or independently. As Table 1 shows, synchronous circuits are the most restrictive class of sequential circuits, designed under the assumption that states change together and in discrete steps, or in clock cycles. While some asynchronous circuits are designed under the same assumption, others are designed under the assumption that states can change continuously, thereby requiring signals to be valid at all times. Different sets of assumptions yield varying tradeoffs in design difficulties, circuit sizes and complexities, and reliability.

Asynchronous finite-state machines (AFSMs) are structurally different from *finite-state machines* (FSMs) in that the clocked state-holding elements are now replaced with the delay elements, which are needed to avoid race conditions that can cause the machines to change multiple times on a single input change. The clocked elements in FSMs contain snapshots of the combinational logic, allowing the combinational logic to change freely between the clock edges. The delay elements in AFSMs, in contrast, continually represent the state of the combinational logic.

AFSMs suffer from two major problems that make them unsuitable for datapath or deeply pipelined designs. The first is that they require the inputs to operate in fundamental mode—an input change should not occur until the machine settles into a stable state—severely limiting the degree of concurrency generally needed in datapath designs. The other is that they suffer from the additive-skew problem. That is, changes to the first machine's inputs are restricted to meet the fundamental-mode assumption of not only the first machine but all other machines connected to it as well. The first machine's inputs must not change faster than the time needed for it and the other machines to become stable.

Since each machine has skew—the difference between the minimum and the maximum propagation delays through the combinational logic—then for two machines in series, the first machine's inputs must not change faster than the sum of the first machine's skew and the propagation delay through the second machine. Meeting the additive-skew requirement results in extremely poor throughput in designs with many pipeline stages.

Circuits, including AFSMs, that assume bounded delays must be designed to function without timing problems in widely varying physical conditions. Designing such circuits requires a correct-by-construction design method and accurate timing models in all phases of the design process; the simple but practical synchronous design method of encap-

Why Clocked Design Is Easier

Assuming that states change only on clock boundaries allows designers to solve separately the functional and timing problems of synchronous circuits. Since the Boolean logic is precise and timeless but propagation delays are imprecise and variable, being able to meet the two requirements separately simplifies both the functional and the timing design processes and makes the required tools more efficient.

Designers can largely ignore the imprecise nature of propagation delays during the functional design process. Propagation delays are inherently imprecise and variable because they are affected by many physical and environmental factors, such as process variations, packaging, temperature, frequency, and voltage. They are difficult to model accurately without detailed physical data. The most accurate physical data is belatedly available from layout when the design is nearly completed.

Functional simulation is simpler when propagation delays are not modeled. To select the right set of features from many design options, designers generally simulate cycle-accurate or near-cycle-accurate behaviors of the features. Such simulators are much faster than event-driven simulators that model delays between events. Even when using event-driven simulators, most designers do not model the individual gate and wire delays but only the signal delays needed to chronologically order the events, resulting in faster simulation.

Timing analysis is also more efficient when logic functions are not modeled. Static timing-analysis tools calculate the longest delay from each input to each output of a module without evaluating the logic between the input and the output. Since only propagation delays and not logic functions are used, the modules can be arbitrarily large and can be built hierarchically using smaller modules.

Static timing analysis is so efficient that it can easily identify critical paths of multimillion-transistor circuits within hours. This is several orders of magnitude faster than dynamic timing-analysis tools, such as Spice, that use more detailed timing models. Although static timing tools are less accurate (most claim to be within 5–10% of the accuracy of Spice), their fast turnaround is indispensible for performing many iterations of logic changes and synthesis to meet timing goals of high-speed designs.

The penalty for this efficient design approach is that state changes can occur no faster than the longest combinational delay in the clocked circuit. If the clock is periodic—and it generally is, except during testing and debugging—its period must be longer than the longest combinational delay when computing the worst case. The penalty is an acceptable compromise for being able to design arbitrarily complex circuits.

sulating combinational logic with latches is utterly unreliable. Bounded-delay circuits also require precision test equipment to confirm the timing models. Most commercial CAD tools and test equipment, indispensable for designing and testing synchronous circuits, are inadequate for handling asynchronous circuits of any meaningful complexity.

Delay-Insensitive Circuits Avoid Hazards, Races

One way to avoid timing problems is to assume that gate and wire delays are unknown and design the circuits to be insensitive to their delays. These delay-insensitive circuits use a two-way handshake for passing each bit of information. Each receiver is required to acknowledge to the sender that it has received the data, but only after the data is no longer needed. The receiver does not know how long it will take to process the data, so each receiver must use a completion-sensing circuit to generate the acknowledge signal. The sender must wait for acknowledge signals from all the receivers before sending the next data item.

Delay-insensitive circuits use transition-based signaling for the handshakes, since there is no way to know when data is sent on a wire by judging its level. When a transition in either direction is sent on a wire, however, the receiver will eventually see the transition. Because a transition indicates when data arrives but not what that data is, dual-rail encoding is used in datapaths. Each signal is carried by two wires—one for sending the transitions for zeroes and the other for ones.

The bundled-data scheme is a more efficient alternative to dual-rail encoding. It uses a data wire for each data bit in a bus and one control wire for validating the entire bundle of data wires. Whereas only transitions are meaningful on the control wire, the logic level on a data wire represents its value. If the delay in the control wire is longer than the delay in the data wires, a transition sent on the control wire after the data wires are set to the data values will be seen by the receiver after the data values have arrived. Although the scheme assumes bounded wire delays, the assumptions are practical and can easily be implemented; since bundled wires are generally routed together, often manually, adding the control wire to the bundle and connecting it to a delay element should be manageable.

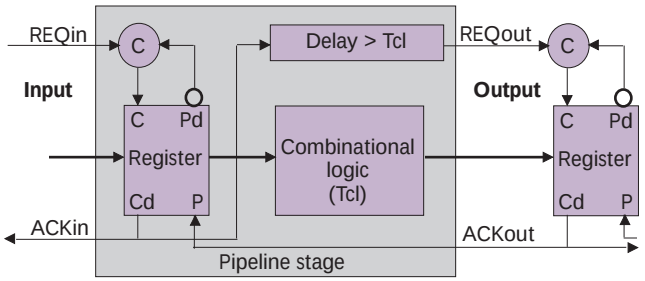

Figure 1. Micropipelines regulate the flow of data using a two-way handshake between adjacent pipeline stages. A clocked pipeline can be converted to a micropipeline by replacing the clocked latches with the micropipeline-control structure that consists of Muller C–elements, transition-based latches, and delay elements. Each delay element must have a longer propagation delay than does the combinational logic in the associated stage.

When fabricated without defects, delay-insensitive circuits will work with any amount of delay. While this property is appealing in theory, the kinds of gates that can be used and the complexity of the delay-insensitive circuits that can be designed are severely limited. In fact, the only single-output gates that can be used are single-input gates, such as buffers and inverters, and Muller C–elements. Muller C–elements are multiple-input, single-output gates whose output is 1 when all inputs are 1, 0 when all inputs are 0, and unchanged otherwise.

Micropipelines Form Clock-Free Pipelines

Introduced by Ivan Sutherland in his Turing Award lecture in 1989 [2], micropipelines offer a conceptually simple way to transform clocked pipelines into asynchronous pipelines. Since registers that separate pipeline stages are controlled by delay-insensitive circuits consisting of inverters and Muller C–elements, they are viewed as having bounded-delay datapaths modulated by delay-insensitive control circuits.

Figure 1 shows a simple micropipeline. The flow of data through the pipeline is controlled by the input handshake signals, REQin and ACKin, and the output handshake signals, REQout and ACKout. The pipeline registers are controlled by two signals; a transition on the C (close) port closes them, and a transition on the P (propagate) port opens them, allowing data to propagate. The Cd port simply delays, and the Pd port delays with an inversion, the transitions on the C and P ports, respectively.

The major benefit of the micropipeline structure is that the registers filter out hazards, eliminating the additive-skew problem. Waiting longer than the worst-case propagation delay through the combinational logic before closing the registers allows the registers to contain snapshots of stable datapath outputs. They also hold the data stable until the registers in the next stage are closed. In this regard, the registers in both micropipelines and synchronous circuits provide the same function with the same benefit. In fact, any combinational logic structure used in clocked circuits can also be used in micropipelines, which means that micropipelines can be built from clocked pipelines simply by replacing the clocked latches with the micropipeline-control structure. Cogency has done exactly this with its first 16-bit asynchronous DSP chip (see MPR 10/6/97, p. 10).

The major drawback of using delay elements, however, is that they take away a key motivation for designing clock-free circuits—that is, using delay elements causes each stage to operate at the speed of its worst-case computing delay. Although different stages still operate at different speeds, the entire pipeline operates at the speed of its slowest stage. If the delay elements can be replaced with self-timed circuits that actually sense when computation is done, the stages can operate at their optimum speeds. Self-timed circuits, however, are

not easy to devise and may not be possible with every circuit structure.

Hal Computer's SPARC64 chip (see MPR 3/6/95, p. 1) uses an asynchronous divider that calculates double-precision results in 5–8 cycles. The divider uses micropipelines but replaces the registers with precharged gates that store partial results. Instead of delay elements, the divider uses completion-sensing circuits for the critical paths, allowing each divide iteration to execute at its full speed. Once a divide operation starts, the divider output is checked every cycle until the result is produced. Implemented in a 0.4-micron process, the divider occupies 5 mm^2. A traditional clocked implementation would take only 2 mm^2 in the same process, but it would also take 54 cycles (both in radix-2 designs).

Clock-Free Amulet2e Uses Micropipelines

For the past five years, a team of graduate students at the University of Manchester has been designing a family of asynchronous ARM microprocessors. Although their initial design, Amulet1, did not show clear advantages in power consumption or performance over synchronous ARM processors, it demonstrated the feasibility of designing a nontrivial (58,000-transistor) asynchronous processor. Amulet2e is an enhanced design that shows more promise.

As Figure 2 shows, the Amulet2e core consists of several micropipelines operating independently and communicating through bundled-data interfaces. Because it uses micropipelines, the core organization is not too different from that of conventional ARM cores. As an example, the memory interface consists of an output bundle for sending control, data, and an address for writing to memory and an input bundle for receiving a word from memory. The memory system may have arbitrary pipeline depths and delays but must return instructions and data in the requested order.

When an instruction is issued from the decoder to the execution pipeline, its destination register is locked, stalling subsequent reads until the result is written to the register. Access to the register file can be controlled by the conventional scoreboarding mechanism, but register forwarding cannot be done using the conventional mechanism without synchronizing across multiple pipeline stages. The solution is to use the LRR (last result register) and LLR (last load register) that hold the latest results of ALU and load operations, respectively, and forward from them to dependent instructions at the beginning of the execution stage. According to Steve Furber, who is directing the research, this seemingly obvious solution took his team a long time to devise—the salient point being that asynchronous design is conceptually different—and much effort is needed to devise the asynchronous equivalents of many performance-enhancing techniques widely used in today's high-performance designs.

Implemented in VLSI's 0.5-micron triple-metal process, the Amulet2e consists of 454,000 transistors and occupies 41 mm^2. Using a 64-way 4K cache on chip, it delivers 40 MIPS while consuming 150 mW. It has a MIPS/W rating of 250, placing it higher than the ARM710's 192 and ARM810's 172 but significantly lower than the Digital SA-110's 745. The Amulet team now aims to deliver its third asynchronous processor, which they claim will match the Digital SA-110 in both performance and power efficiency.

Obstacles Abound for Clock-Free Designs

A major obstacle facing clock-free circuits is the lack of simple methods for designing arbitrarily complex circuits that can also be built reliably. Due to their fundamental mode of operation and resulting additive skew, asynchronous finite-state machines are ill suited for designing datapaths or pipelined circuits. Delay-insensitive circuits are robust and easy to fabricate, but they are extremely simple and therefore not too useful. Speed-insensitive and quasi-delay-insensitive circuits are under intense academic research, due to the somewhat insensitive delay models that make them synthesizable. Although many methods have been proposed, only a few have been tried with nontrivial designs.

Another major obstacle is the lack of design tools that support clock-free design. Most design tools take advantage of the fact that functional and timing designs are done largely in separate steps, making these tools unsuitable for asynchronous design. To handle millions of transistors efficiently, functional simulators generally ignore timing attributes, and static timing-analysis tools mostly ignore functional attributes. Synthesis tools first translate combinational logic into expression trees without regard to timing, then use heuristics to repeatedly partition the trees and map to available standard cells until timing goals are met (or time limits

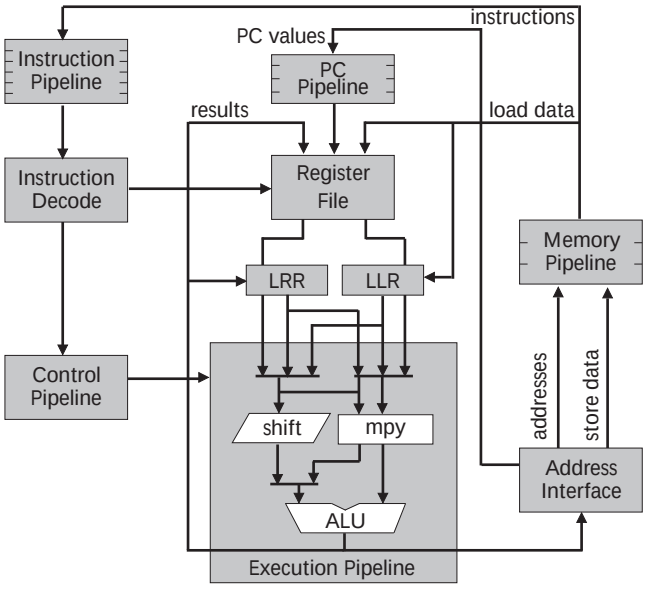

Figure 2. The Amulet2e core consists of several micropipelines operating independently and communicating via bundled-data interfaces. The core (excluding the memory pipeline from the figure) consists of 93,000 transistors.

References and Resources

[1] Hauck, S. "Asynchronous Design Methodologies: An Overview." In *Proceedings of the IEEE, Vol. 83, No. 1*, pp. 69–93, January 1995.

[2] Sutherland, I. "Micropipelines." In *Communications of the ACM*, Vol. 32, No. 6, pp. 720–738, June 1989.

[*] Davis, A., S. Nowick, eds. *Asynchronous Digital Circuit Design*, Springer-Verlag, 1994.

[*] Hulgaard, H., et al. "Testing Asynchronous Circuits: A Survey." University of Washington, Dept. of CSE International Report, TR # 94-03-06, 1991.

[*] *www.cs.man.ac.uk/amulet/async*

[*] *www.cs.utah.edu/projects/acs*

run out). Although some precision timing-analysis tools, such as Spice or EPIC's TimeMill, do treat functional and timing attributes as one indivisible entity, they are utterly inefficient as functional simulators that must deliver millions of simulation cycles for overnight regression.

The inability to suspend and advance state machines at will in asynchronous chips makes debugging and testing them difficult. Whereas the entire state of synchronous chips can be frozen, advanced, and even modified via scan operations, the state of asynchronous chips cannot. Without clocked registers, single-step operations are not possible: with asynchronous chips, a change in input pins propagates until the entire chip reaches a stable state. Asynchronous chips either run at full speed or remain halted; it is not possible to stop them until they halt naturally.

Existing manufacturing-test equipment is inadequate for testing asynchronous circuits. For circuits that assume bounded gate and wire delays, the widely used stuck-at fault model is inadequate and therefore requires path-delay tests for detecting delay faults. Path-delay tests involve the application of test vectors and the timely sampling of responses—each sample is taken after a time that allows a path to produce a correct output if it is defect-free but an incorrect output if it contains a delay fault. These tests are significantly more expensive to perform than stuck-at fault tests. In addition, adding extra logic to eliminate hazards during design directly conflicts with the need to eliminate redundant logic for testing, as it makes detecting faults difficult.

Circuits that use the continuous-time model are also more susceptible to noise than those that use the discrete-time model. Whereas state changes are designed to take place after noise has time to subside in the discrete-time model, circuits must be stable at all times in the continuous-time model. Noise and timing hazards can be dealt with by slowing down the clock in synchronous chips, but they generally cause malfunctions that cannot be fixed in asynchronous chips. Asynchronous circuits, however, generate lower levels of noise and EMI than clocked circuits.

Clock-Free Chips Won't Displace Clocked Chips

Clock-free circuits can have low power consumption and low levels of electromagnetic interference. They can be optimized for average-case, instead of the infrequent worst-case, computing times to simplify designs or yield smaller sizes without compromising overall performance. Due to difficulties in designing and testing asynchronous circuits, however, many of these potential benefits have yet to be turned into competitive advantages over synchronous circuits.

Asynchronous circuits cover a wide range of circuits and design methods. Although delay-insensitive circuits are less than practical, they introduce the ideas of using transition-based signaling for control and using either dual-rail encoding or the bundled-data scheme for passing data. Closest to synchronous circuits, and therefore not taking full advantage of being clock-free, micropipelines offer the most practical way to design asynchronous circuits. Micropipelines avoid additive skew, as well as hazards, by isolating adjacent stages with pipeline registers. Although not enough is known about applying micropipelines to more general circuit structures, large-scale design efforts, such as those undertaken by Cogency and the Amulet group, are providing more insight.

All the asynchronous design methods, however, are likely to be more complicated than the more familiar synchronous design methods. Although most designers learn about asynchronous design techniques in their introduction to logic design, there are no products from which to gain practical experience and further develop skill. This situation is unlikely to change in the foreseeable future unless a nontrivial asynchronous design can clearly demonstrate technical superiority over synchronous designs.

There is a circle of dependence that hinders wider acceptance of asynchronous circuits and more aggressive development of their design methods and tools. Given the enormous mismatch between the states of synchronous and asynchronous designs, asynchronous designs are unlikely to deliver higher performance than synchronous designs, even at comparable power-consumption levels. Without demonstrably clear market advantages, few may commit to asynchronous designs when easier solutions can be found. Many portable applications that are highly sensitive to EMI may find clock-free designs attractive, however.

Asynchronous circuits are more likely to be used within synchronous chips than by themselves. By themselves, they are less likely to have the levels of complexity needed to make them complete solutions in most applications. Due to accelerating advances in semiconductor processes, the minimum level of integration in a single chip is likely to grow faster than the complexity of the asynchronous chips that can be designed. By implementing clock-free only the modules that are well suited for it—datapaths that tend to have simple and repetitive logic, modules that have few interfaces and sporadic inputs, and algorithms that exhibit large computing delays between the worst case and average case—the difficulties of designing clock-free chips can be largely mitigated. M