ASYNCHRONOUS SEQUENTIAL LOGIC

In sequential logic (see ***Sequential Circuits***), output is a function of current input and the state stored in the system. In synchronous sequential logic circuits, time is quantized, making all actions and state changes take place at discrete intervals of time, determined by a regular source of pulses called a clock. For the other more general class, called asynchronous sequential circuits (*ASC*s), timing information is introduced without the use of a global clock; thus, events in signals and changes in states take place at any time. The use of ASCs may often bring some advantages when implementing digital control and processing structures. Such advantages come from the decentralization of the timing control in the operation of the system, with self-synchronization taking place at a local level.

This article introduces most of the fundamental issues related to ASCs: comparison with synchronous circuits and potential advantages of ASCs, major specification and design techniques, different implementation architectures, and performance characteristics.

An ASC can be simply defined as a sequential circuit whose internal states change only in response to changes in its inputs, with no common timing reference (see Ref. 1 for a complete introduction). While the reader can easily understand how a privileged signal—called the clock—can control the change of the state in a synchronous sequential circuit, the way to ensure correct operation in an ASC is not so clear. Thus, it is necessary to establish more precisely the operational procedure of the ASC, establishing suppositions about the delay models of components and interconnections in the system. Negligible, bounded, arbitrary, or unbounded delay models in gates and interconnections can be considered.

A simple ASC model is the Huffman circuit (Fig. 1), which is basically composed of a combinatorial circuit and a set of feedback lines, with a bounded (or zero) delay model for interconnections. For a Huffman ASC to work properly (hazard- and race-free operation), the input signals can only change once the internal state has been correctly settled (operation under fundamental mode and single-input change). There are some other operation modes, such as input–output, multiple, or unrestricted input change, with different constrains. A special operation mode, called *burst mode* operation, allows operation in the fundamental mode but on bursts of inputs rather than single inputs. For additional information, see Reading List.

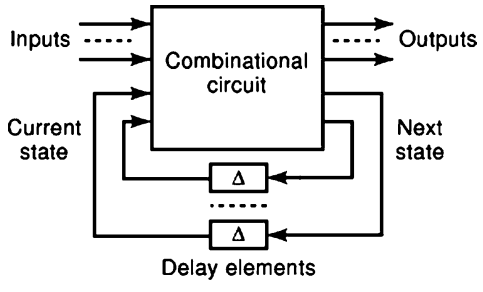

Figure 1. Huffman Circuit. Delay elements can be explicitly placed or being simply the delay in feedback lines.

More generally, an ASC is called *speed-independent* when it operates correctly (hazard-free) for any finite delay in gates. A subset of these circuits generates a completion signal indicating that its operation has finished. For correct behavior, changes in input signals are only allowed when a completion signal is activated. More restrictive is the delay-insensitive ASC, which works correctly for any finite delay in gates and interconnections. An intermediate category is the quasi-delay-insensitive ASC, which is a delay-insensitive ASC that considers isochronic forks. In this type of ASC, delay in interconnections is arbitrary except in forks, where the two branches have similar delays.

SELF-TIMED APPROACH

A self-timed circuit, also called a handshake circuit, is an ASC that is self-synchronized with its environment through a handshaking protocol (see Ref. 2 for a complete introduction). The behavior of components and elements in a self-timed system is conducted by events in their terminal ports: The beginning of the operation of the system is caused by a specific event in an input signal (request), and the end of the operation is indicated to the outside by another event in an output signal (acknowledgment). Thus, the time required to perform the computation or processing is determined by internal delays of gates and interconnections inside the circuit, corresponding to the time elapsed between the request and the acknowledgment events. A precedence relation exists between such events, in that initiation must take place prior to finishing, indicating a sequence of events.

A self-timed system can be defined either as (1) a self-timed circuit itself, or (2) a correct connection of self-timed circuits. Such a correct connection incorporates the restrictions in the communication between such elements, imposed by the handshaking protocol. In a simplified model, the handshaking protocol is verified by specific signals called *protocol signals.* In such signals at least two events are necessary to describe the self-timed operation (request and acknowledgment), and these events must alternate. Figure 2(a) shows a so-called two-phase, or no-return-to-zero (*NRZ*), handshaking protocol characterized by the events that occur at the edges of protocol signals. Thus, logic circuits operating under this protocol should be edge-triggered. On the other hand, Fig. 2(b) displays a so-called four-phase, or return-to-zero (*RZ*), handshaking protocol, which is level-sensitive. Systems incorporating such protocols will present different performance: The two-phase protocol is faster and, since it has less transitions, consumes less power. However, the four-phase protocol is easier to implement because it operates with less-costly level-sensitive hardware.

Potential advantages in the use of the self-timed approach are based on its higher efficiency in computing data, especially in those cases where processing time is strongly data-dependent; self-timed ASCs operate on an average-case basis, while synchronous circuits operate on a worst-case basis.

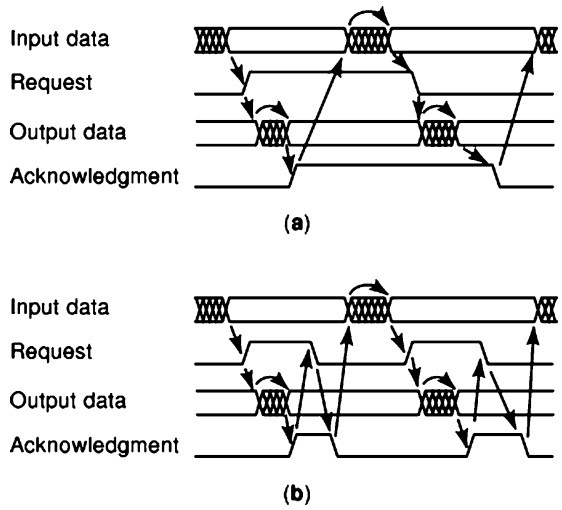

Figure 2. Handshaking protocol: (a) 2-phase or edge-sensitive; (b) 4-phase or level-sensitive.

GLOBALLY ASYNCHRONOUS LOCALLY SYNCHRONOUS APPROACH

A promising method between pure synchronous and self-timed circuits is the Globally Asynchronous Locally Synchronous (GALS) approach. Although the background for this technique was set in 1984 (3), recently has received a lot of attention because it offers advantages from both the synchronous and asynchronous domains. In GALS systems, the components are synchronous modules operating at their own clock speed, which allows the proven synchronous design methodologies to be used. The interface between the synchronous components is made with self-timed circuitry, generating the local clock signal under a request-acknowledgement basis. The main advantages in using the GALS technique are the elimination of problems related to the usage of a global clock (see next section) and the possibility of using classical synchronous cores, methodologies and CAD tools. The main drawback is the metastability problem when interacting synchronous and asynchronous signals, being this problem faced with design solutions (4).

LIMITATIONS OF SYNCHRONOUS CIRCUITS

Most of the problems and limitations of synchronous circuits stem from the existence of a global clock signal. The main problems are crosstalk noise and, especially, clock-skew and synchronization faults.

Clock-Skew Problems

In synchronous circuits, the clock signal must arrive simultaneously at the memory elements to avoid race problems. However, generating and distributing high-frequency clocks inside very large-scale (*VLSI*) integrated circuits (*IC*s) is a very difficult and expensive task, so that eliminating clock skew often limits system performance (5–7). Clock skew appears because the clock paths suffer different delays, causing synchronization failures in the system. The nature of clock skew is unpredictable for two main reasons: first, at a logic level, the designer cannot prevent the placement of memory elements in the layout, and second, the variations in the delays depend on various factors, such as operation temperature and technological process deviations. Thus, the designer cannot ensure clock-skew-free operation.

Figure 3 shows an example of the pernicious effects of clock skew. The correct operation requires that the first bistable store its input datum D_1, while the second bistable stores Q_1. However, if the delay in the clock signal is greater than the delay in the datum line ($\Delta_2 > \Delta_1$ + propagation delay of first bistable), D_1 may be stored instead in the second bistable, so that the Q_1 value is lost.

This problem is nowadays aggravated because, with current technologies, delays in interconnection paths are becoming comparable to delays in gates. Classical solutions to this problem, such as (1) routing of clock signals in the opposite direction to data flow and (2) use of nonoverlapping clocks, limit the overall performance of the system. The most effective solutions, such as identical buffering in each clock path or routing clock signals through H-tree networks, are much more expensive in design time and cost (5–7). Parameters that should be taken into account in generating and distributing clock signals are routing layers, clock network shape, clock generators, rise and fall times, and capacitive and resistive load in lines.

Synchronization Problems

Synchronization problems can have various causes, but the primary cause is metastable operation in bistables. The probability of failure increases with the complexity and operation speed of the system. In synchronous operation, some timing restrictions in bistables, concerning hold time, setup time, and pulse width, must be observed. Due to asynchronous interactions—a delay between data and clock signals, for instance—that cause a violation in such restrictions, a bistable may enter its metastable state, showing in its output an undetermined logic state for an indefinite time. If such output acts as the input of two parallel bistables, these may read different logic values upon the arrival of the next clock edge. In such a case, a system error occurs as a consequence of a synchronization fault.

In view of these problems, it is of interest to consider the design of ASCs, which provide solutions in that (1) ASCs do not need a global clock to synchronize the operation of the circuit, and (2) the handshaking protocol imposes restrictions on changes in the inputs of the memory elements.

DESCRIPTION AND REPRESENTATION TECHNIQUES OF ASYNCHRONOUS SEQUENTIAL CIRCUITS

Flow diagram and tables are the classic ways of describing ASC (1). States are stored in the feedback loops or in asynchronous latches. Correct operation is ensured with the assumption of operation in the fundamental mode and with single-input change. Once the flow table has been generated or obtained, minimization and assignment processes generate an asynchronous implementation, which should be race- and hazard-free.

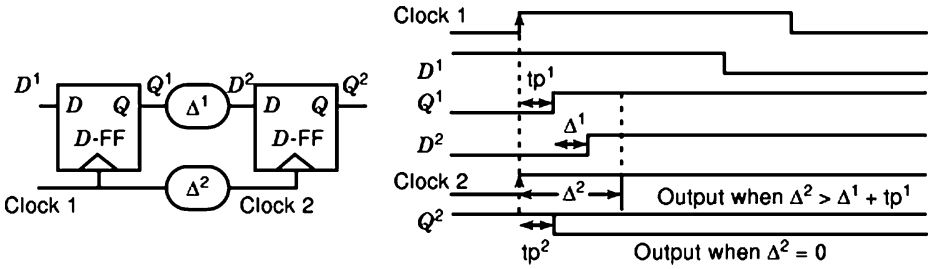

Figure 3. Illustrative example of negative effects of clock skew. Datum in Q_1 can be lost if delay in clock line is higher than delay in datum line plus propagation delay in first bistable.

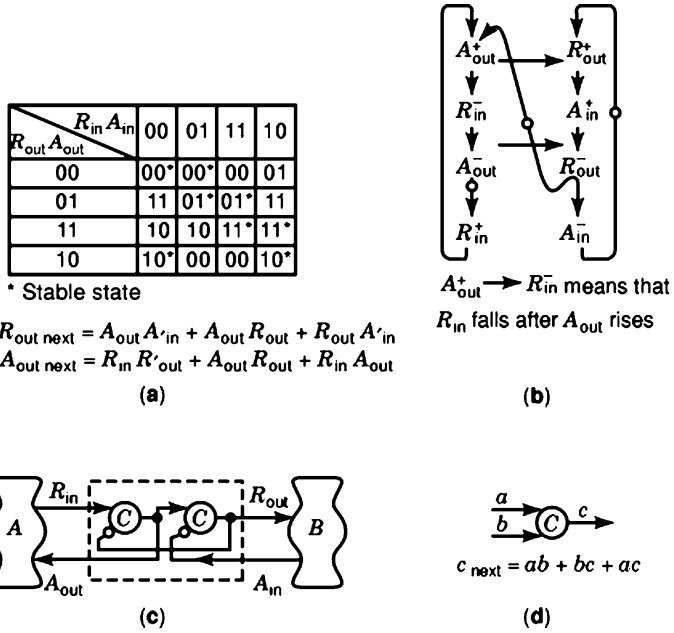

Figure 4. State table (a) and STG (b) corresponding to an asynchronous pipeline interface (c), working under a 4-phase handshaking protocol, and implemented with two two-input C-elements (d).

The characteristic of ASCs by which changes in inputs directly bring about changes in states and outputs makes graph-based descriptions suitable. Such representations easily describe precedence relations between events in signals and allow for parallelism, decisions, and conflict between processes as well. An important, but not unique, graph-based description is the signal transition graph (*STG*), an interpreted Petri net. Basically, STGs are formed by *places*—signal transitions, labeled + for rise transitions and—for fall transitions—and by *arcs* connecting places. If the STG satisfies some criteria of "good behavior" (liveness, safety, persistence, semimodularity, etc.), it may represent a hazard-free ASC (5). Other techniques based on Petri nets are change diagrams and I nets.

Figure 4 shows the table- and graph-based description and a possible implementation of a pipeline interface operating under a four-phase handshaking protocol. R_{in} performs a request from block A, acknowledged by A_{out} and transmitted to block B through the R_{out} signal, depending on the value of A_{in}, which indicates if B is ready to accept data. All signals are considered active high.

Starting from an initial state ($R_{out}\,A_{out}$ = 0 0) with no request stored, activation of a request (R_{in}^{+}) leads the system to the state ($R_{out}\,A_{out}$ = 0 1), indicating that the request has been acknowledged and considered. Only when A_{in} is disabled (A_{in}^{-}), indicating that the following cell is idle, does the system go to the state ($R_{out}\,A_{out}$ = 11), transferring the request from the present to the next stage. Disabling the input request (R_{in}^{-}) forces the falling of the acknowledge signal (A_{out}^{-}), leading the system to the state ($R_{out}\,A_{out}$ = 10), from which the system is returned to the initial state by disabling A_{in}.

The proposed race-free implementation uses as a memory cell a C-Muller element: a bistable that stores its input values when they are coincident. The characteristics of this bistable (i.e., the fact that it performs the AND operation on events) make it the recommended memory element for self-timed implementations, as suggested by Meng (8) and Martin (9).

Peculiarities of ASCs exclude classic synthesis tools used for synchronous sequential circuits. Different methodologies have been presented, which are focused on graph- and state-based descriptions. Also, VLSI programming and sintax-driven translation has recently received attention (9–11).

Synthesis tools using state representation (flow diagrams and tables) are oriented toward the implementation of race-free tables by using techniques of state minimization and efficient assignment. For instance, so-called "single-transition table" (STT) state assignments provide race-free VLSI circuits (see Reading List).

Synthesis tools using graph representations (STG or Petri nets) utilize as input an original graph that contains the desired behavior. This graph is transformed by using different techniques in such a way that the transformed graph verifies some properties of "good behavior," such as liveness, safety, or semimodularity. Some algorithms generate a state diagram and a hazard-free circuit is synthesized. The transformation of the original graph into the modified one takes place but adding arcs (causal relations) and signals in such a way that the transformed graph verifies the above-mentioned properties, and the resulting circuit is hazard-free (see Reading List).

VLSI programming allows the description of typical asynchronous VLSI processes such as concurrence and parallelism between processes. Synthesis tools based on VLSI programming, as for instance TANGRAM (10), BALSA (11) or the one used in (9), directly translate a high-level description into hardware through connections between handshake signals and circuits.

ARCHITECTURES OF ASYNCHRONOUS SEQUENTIAL CIRCUITS

The main parameter that characterizes the architecture of ASCs is the way that the self-synchronization is performed. In self-timed ASCs, the timing control is carried out by data themselves, assisted by specific protocol—*handshake*—signals. The way that this information is included depends on the data encoding and on the relationship between data and handshake signals. There are two main data signaling schemes used in self-timed ASCs: dual- and single-rail codification.

Dual-Rail Codification

Using dual-rail code, also called self-synchronizing or delay-insensitive code, allows the inclusion of information about the validity of data by including redundancy of information. A simple code uses two signal bits (x_t and x_f) per data bit (x). Thus, we can express four possible values: when both x_t and x_f are inactive (low level, for instance), an "empty" or "spacer" state is defined, indicating that data are not valid. When either x_t or x_f is active (high level, for instance), the data are valid (true or false, respectively). By definition, x_t and x_f cannot be simultaneously active. Figure 5 presents a waveform diagram showing the synchronization scheme using dual-rail data, working with a four-phase handshaking protocol. Only one transition per bit takes place per operation cycle, while valid data and spacers are forced to alternate. Delay-insensitive operation may be ensured, since delay in interconnections would only bring about additional delays in transitions, but events occur in the right sequence.

Single-Rail Codification

This approach, also called *bundled data,* uses a specific handshake signal to validate data, in such a way that only one signal per data bit is needed. However, synchronization between the validation signal and data signal is required to ensure correct operation; thus, delay-insensitive operation is not guaranteed. To validate the output data of an ASC, it is necessary to generate a completion signal once the operation of the circuit is finished. This completion signal can be used as a request signal for other ASCs. The two most widely accepted mechanisms for generating completion signals are based on the use of matched delays and the use of differential circuits as computation or processing elements.

Matched Delays. This technique (2, 12) generates a completion signal by using a delay element that matches the worst-case delay of the combinational logic (Fig. 6). When the request is activated, input data are valid. Since the combinational logic takes less time to process data than the propagation time of the delay element, once the completion signal is activated, the output data are necessarily valid. This scheme has the advantage of simplicity, but its operation is always performed considering the worst-case delay. Furthermore, the correct calculation of propagation delays and implementation of delay elements requires exhaustive timing simulations (see ***Delay circuits***).

Differential Circuits. Using differential circuits as computation or processing blocks provides an efficient way of generating completion signals (8). These circuits, which are well suited for complementary metal–oxide–semiconductor (*CMOS*) implementations, generate both the true and the complemented outputs. However, dual-coded inputs are needed. Conversion of single-rail to dual-rail data can be performed at a local level. The generic schematic and waveform diagrams are shown in Fig. 7. In the precharge phase, outputs take the same value, while in the evaluation phase, the logic function is performed and the two outputs take complemented values. A simple logic gate detecting the complemented values can generate the completion signal. The main advantage is the adaptability to new operation conditions, but at the cost of expensive hardware resources.

Figure 8 shows an example of bundled-data architecture using differential circuits to generate completion signals. Synchronous memory elements (D flip-flops) are locally clocked by protocol signals (not shown in the figure) in such a way that data must be stored and stable while they are being processed. Single- to dual-rail data conversion takes place in the memory elements. Interconnection circuits are implemented with two C elements, as we can see in the ASC shown in Fig. 4. The R_{out} signal acts as a request signal for the differential circuit, while the completion signal is the R_{in} signal for the following interconnection circuit.

MACROMODULE-BASED CIRCUITS

Most current handshake circuits combine some of the above-mentioned characteristics: two- or four-phase handshaking protocol, matched delays or differential circuits, and single-rail or dual-rail codification. A common characteristic is their modularity, in the sense that we can interconnect several modules that work under the same handshaking protocol and codification schemes to build a complex self-timed ASC. Thus, an efficient approach to the development of handshake circuits is the use of a library of macromodules that, correctly interconnected, can perform any desired functionality.

With respect to interconnections between macromodules, although they can be used to design delay-insensitive control modules, their implementation is not delay-insensitive or even speed-independent.

Micropipelines

A very important macromodule-based approach, called micropipelines, was presented by Sutherland (12). It uses a two-phase handshaking protocol, single-rail codification, and matched delays, and its basic architecture is shown in Fig. 9. For the data path, it uses data pass–capture latches to store data in events of protocol signals. For control, it uses a library of event-sensitive macromodules, shown in Fig. 10. The XOR gate and the C element perform the OR and AND operation of events, respectively. The toggle cell transfers an event from its input to its two outputs alternately, starting with the dotted output. The select block allows a Boolean to direct the input event to the true or

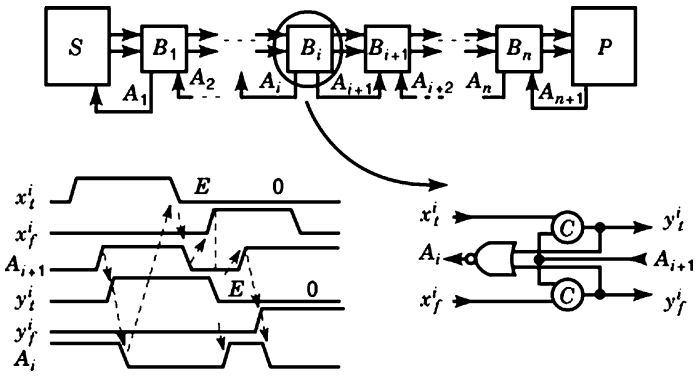

Figure 5. Speed-independent buffer as example of a dual-rail scheme for data signaling. A 4-phase handshaking protocol has been used. Empty (E) and Valid Data values are forced to alternate.

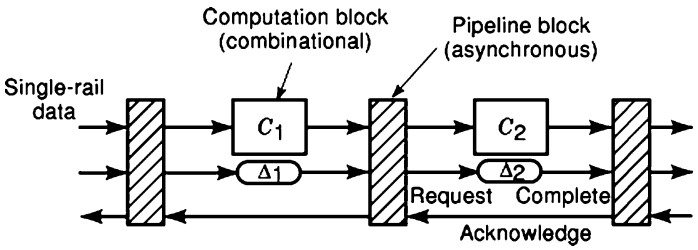

Figure 6. Bundled-data scheme using matched delays to generate complete signal. Δ_1 (Δ_2) matches the worst case delay of C_1 (C_2). Data signals must be validated by handshake signals.

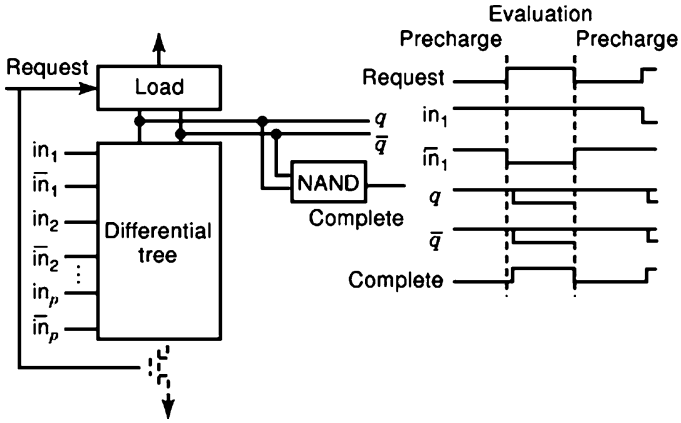

Figure 7. (a) Logic schematic of a generic differential logic block. LOAD block sets the precharge values and the differential tree generates both the true and the complemented logic output. (b) Waveform diagram showing how precharge and evaluation phases alternate.

false output. The call block allows two independent, mutually exclusive processes to share a common subprocess. The arbiter cell grants a common resource to only one of the elements that requested it. The mutually exclusive element (MUTEX) ensures that the resource can never be used simultaneously by the two elements that requested it.

Control and Data Handshake Circuits

One of the main approaches to the design of ASCs uses VLSI programming for direct translation of high-level descriptions into hardware (9–11). Control and data handshake circuits are the result of compiling the behavior description. A handshake circuit is a (quasi) delay-insensitive network of components connected by communication channels. A control handshake circuit communicates with other components through request/acknowledge signaling through the channels. Data handshake circuits also include data signaling. Following heuristic or systematic techniques, you can design more complex components based on simple handshake circuits.

As an example of an ASC built with handshake circuits (taken from Ref. 10), Fig. 11 shows the high-level description, symbol, and handshake-based implementation of one- and two-stage first-in, first-out (*FIFO*) memories. There is a direct translation from language (*forever do*) into a handshake circuit (repeater block, marked with $*$; and with a possible implementation is shown in Fig. 12). Blocks marked ";" are sequencers, which complete handshaking from the ASC's input to its outputs alternatively. The T and x blocks, called transferrers and handshake latches, respectively, are data handshake blocks, capable of transferring and storing data signals according to the handshake protocol. An open circle a block indicates that the request acts as input and the acknowledge as output (passive port), while a filled circle indicates an output request and input acknowledge (active port).

For the one-stage FIFO in Fig. 11 (see Fig. 4 for the same basic functionality), the statement ($a?x0$; $b!x1$) indicates that input a is loaded in variable x, which can be read through the b port. The action of the sequencer makes it possible for data to be written before being read, verifying the handshaking protocol. The two-stage FIFO is built

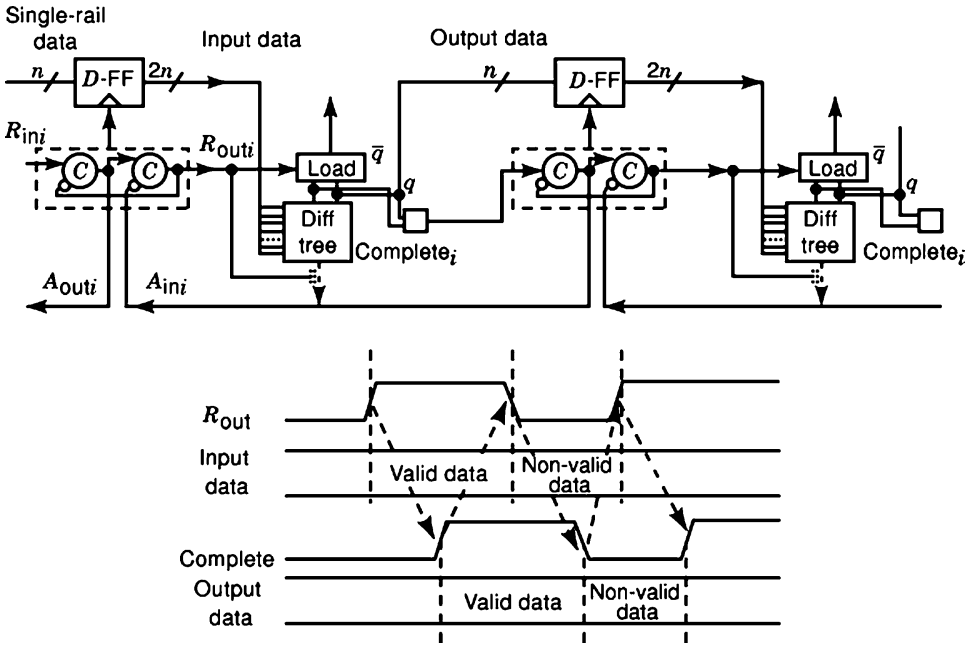

Figure 8. Example of bundled-data architecture using differential circuits as computation blocks. The generation of complete signals is quite straightforward by using a logic gate. A 4-phase handshaking protocol is used.

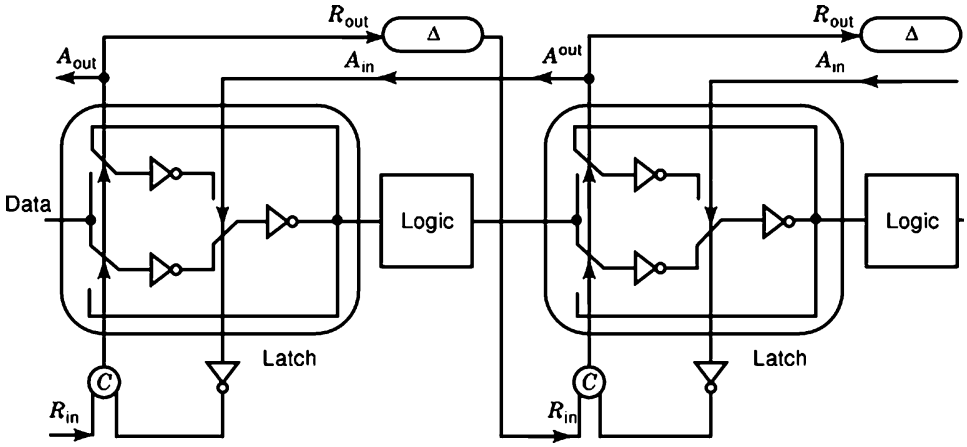

Figure 9. Micropipelined single-rail data architecture. The data path is implemented with combinatory logic to perform the logic function and Pass–Capture latches to store data. Control manages protocol signals and write/read operation in latches. Matched delays are used for completing handshaking protocol (see reference 12 for more complex examples).

with two cascaded one-stage FIFOs operating in parallel; it is marked with the symbol ‖ in the specification (Fig. 11). The final circuit can be synthesized by substituting each handshake component for its schematic and layout.

DISCUSSION OF CHARACTERISTICS AND PERFORMANCES

Current state-of-the art ASCs are more complex and, in general, more difficult to design than their synchronous counterparts. Many advantages of ASCs over synchronous circuits have been claimed, such as automatic adaptation to physical properties, better accommodation to asynchronous external inputs, better technology migration potential, more timing reliability, lower noise and electromagnetic emission, and higher modularity. However, the most prominent advantages of ASCs come from their special ability to exploit data dependence in operation time and their lower power consumption. Thus, there are some applications where ASCs can be recommended, such as digital signal processing and low-power applications. Some emergent applications are in the field of thermally-aware circuits, secure systems as smart cards, and the implementation of bio-inspired artificial vision systems, based on the asynchronous address-event-representation communication scheme. Advanced aspects, such as testability or verification, are still under development.

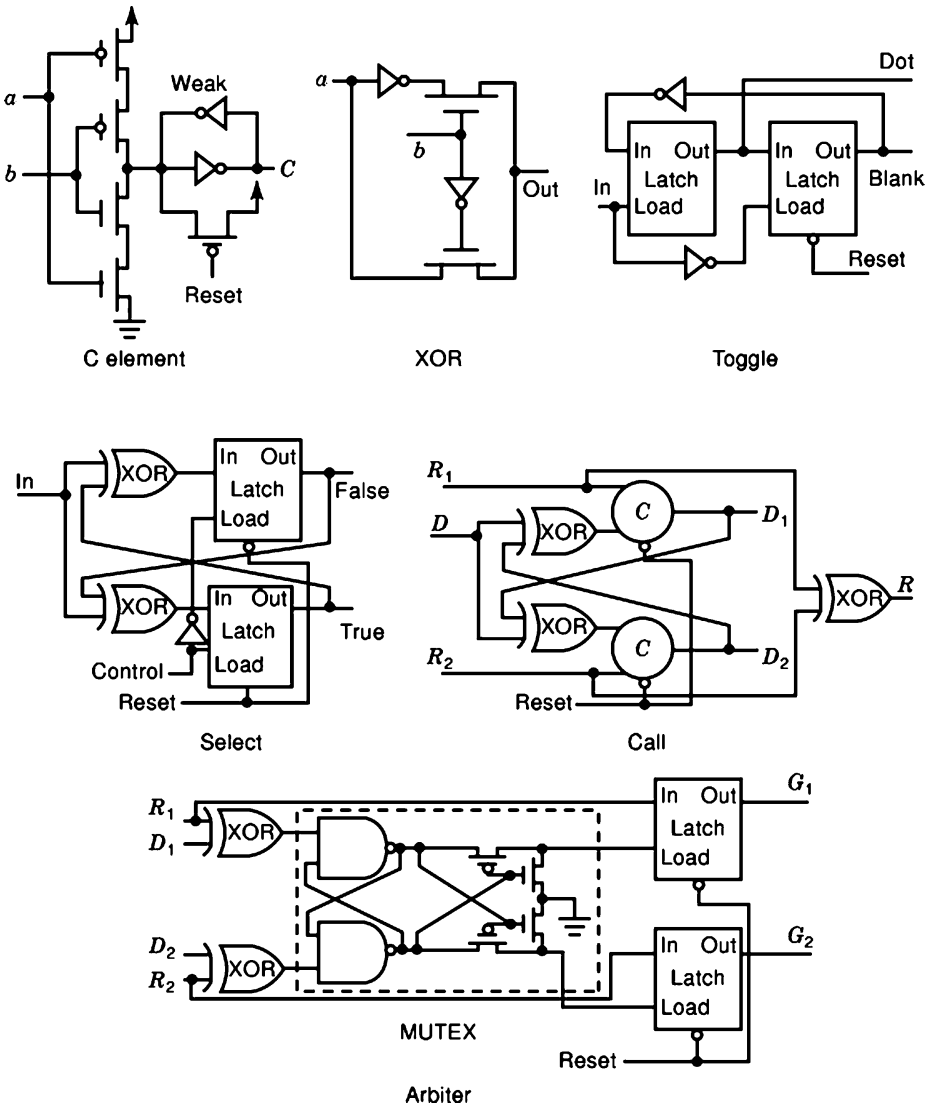

Figure 10. Event-sensitive macromodule library and a possible CMOS implementation of each cell.

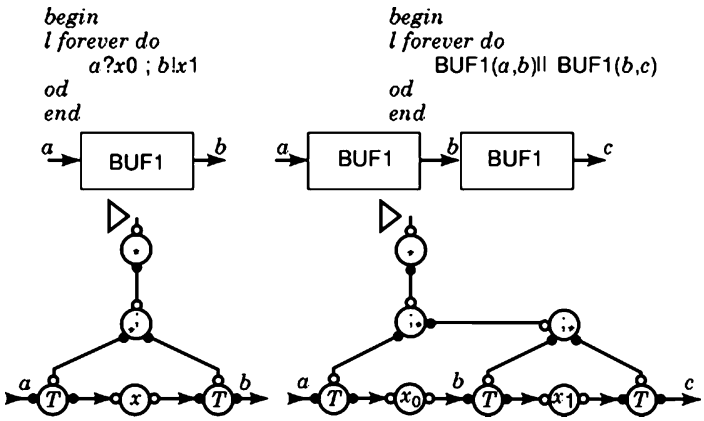

Figure 11. High-level description, symbol and module implementation of 1-stage and 2-stage FIFO memories.

To show the data dependence, let us consider (Fig. 13) a ripple carry adder (see ***Summing circuits***), where the time needed to perform the operation depends on the input words (2). The best cases correspond to direct generation of output carry of all cell bits, and this occurs when the added bits have the same value $(a_i, b_i, c_{i-1}) = (1, 1, x)$ or $(0, 0, x)$, giving as output carry 1 and 0, respectively, regardless of the value of the input carry. The worst case is given by the propagation of carry throughout the whole chain, whereby each cell needs the output carry of the previous cell to fin-

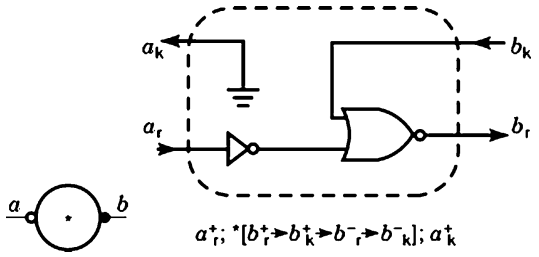

Figure 12. A possible implementation of a Repeater block. Its functionality is summarized as follows: a requests b (a_r^+); b is indefinitely executed ($b_r^+ \rightarrow b_k^+ \rightarrow b_r^- \rightarrow b_k^-$); a is released (a_k^+). A 4-phase handshaking protocol is supposed.

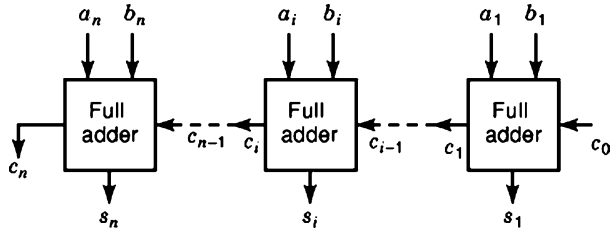

Figure 13. Ripple carry adder as an example (taken from reference 2) showing the dependence of processing time with input data. The operation performed by each full adder is $c_i = a_i\ b_i + a_i\ c_{i-1} + b_i\ c_{i-1}$; $s_i = a_i$ xor b_i xor c_{i-1}. If $a_n\ b_n = 1\ 1$, then $c_n = 1$; if $a_n\ b_n = 0\ 1$, then $c_n = a_{n-1}\ b_{n-1} + a_{n-1}\ c_{n-1} + b_{n-1}\ c_{n-1}$, depending recursively on the previous carry.

ish its processing. An input vector such as $(a_i, b_i, c_{i-1}) = (1, 0, x)$ will create such a situation. This example shows how the data themselves lead to very different time processing. While synchronous circuits must take into account the worst-case operation, ASC can operate on the average case.

Operation Speed

At a circuit level, ASCs show more tolerance for physical variations, such as deviation in the technological process and variations in supply voltage and temperature. This is mainly due to the action of the handshake and the generation of completion signals (indicating when the operation has been finished) and to their working at the maximum speed possible. At an algorithmic or architectural level, ASCs' ability to operate on an average case is helpful, especially when the worst and average cases are very different; and they are not limited by the slowest processing block (2, 8). However, verification of the handshaking protocol requires two processes: (1) monitoring the state, and (2) "wait or go" operation. Thus a tradeoff exists between the two approaches.

Power Consumption

In synchronous circuits, clock lines have to be toggled and circuit nodes charged and discharged even in unused parts or when the circuit is idle and there are no data to compute. Also, the generation of "good" clock signals (vertical edges) consumes a lot of power in each transition. Although ASCs often require more signal transitions in a given computation than do synchronous circuits, these transitions usually occur in areas involved in the current computation. Moreover, problems related to the generation of clocks are minimized. Figure 14 shows a generic representation of power consumption versus operations performed. Because a clock consumes power when the circuit is idle, depending on the input data rate, the ASC consumes less power. A good example of an ASC exhibiting less power consumption than its synchronous counterpart is found in Ref. 13.

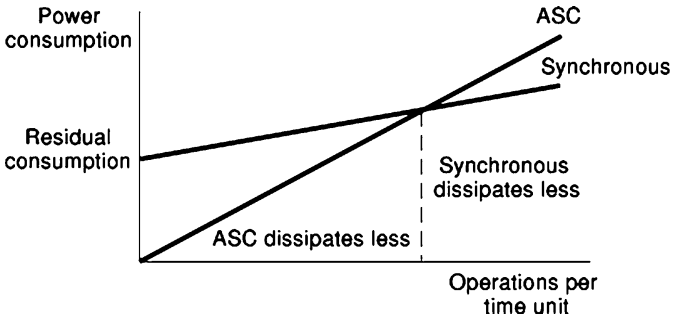

Figure 14. Representation of the power consumption vs operations performed. In the synchronous case, there is power consumption even if there are no data to compute. In a self-timed ASC, for relatively low input data rate, consumption is lower.

There are some interesting approaches combining the advantages of the synchronous and the asynchronous style. These structures are locally clocked and are based on the generation of a local clock that ensures correct operation of the circuit under asynchronous inputs. The most important are those based on burst-mode operation, metastable-insensitive Q-modules, and stoppable clocks (10).

BIBLIOGRAPHY

1. S. H. Unger, *Asynchronous Sequential Switching Circuits.*, New York: Wiley-Interscience, 1969.
2. C. L. Seitz, System timing, inC. A. Mead andL. Conway (eds.), *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley Pubs., 1980.
3. D. M. Chapiro, *Globally-asynchronous locally-synchronous*, PhD thesis, Stanford University, 1984.
4. D. Sokolov and A. Yakovlev, Clockless Circuits and System Synthesis, *IEE Proc. Computers and Digital Techniques*, **152** (3): 298–316, 2005.
5. H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*. Reading, MA: Addison-Wesley Pubs., 1990.
6. J. M. Rabaey, *Digital Integrated Circuits. A Design Perspective*. Englewood Cliffs, NJ: Prentice-Hall, 1996.
7. E. G.Friedman, Clock Distribution Networks in Synchronous Digital Integrated Circuits, *Proceedings of the IEEE*, **89** (5): 665–692, 2001.
8. T. H. Y. Meng, *Synchronization Design for Digital Systems*. Norwell, MA: Kluwer Academic Pubs., 1991.
9. A. J. Martin, Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, **1** (4): 226–234, 1986.
10. K. van Berkel, *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*. Cambridge University Press, 1993.

11. A. Bardsley,*The BALSA Asynchronous Synthesis Systems web pages*: http://www.cs.manchester.ac.uk/apt/projects/tools/balsa/

12. I. E. Sutherland, Micropipelines. *Commun. of the ACM*, **32** (6): 720–738, 1989.

13. K. van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalij, and A. Peeters, Asynchronous circuits for a low power: A DCC error corrector. *IEEE Design and Test of Computers*, **11** (2): 22–32, 1994.

Reading List

Most classical textbooks dedicated to digital logic design discuss asynchronous sequential logic. A summary of them is as follows:

A. E. A. Almaini, *Electronic Logic Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1994, chap. 5.

E. J. McCluskey, *Logic Design Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1986, chap. 9.

F. J. Hill and G. R. Peterson, *Computer Aided Logical Design with Emphasis on VLSI*. New York: John Wiley & Sons, 1993, chap. 14.

R. F. Tinder, *Digital Engineering Design: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall 1991, chap. 6.

S. H. Unger, *The Essence of Logic Circuits*. 2nd ed., Piscataway, NJ: IEEE Press, 1997, chap. 6.

ANTONIO J. ACOSTA-JIMÉNEZ
MANUEL J. BELLIDO-DÍAZ
ANGEL BARRIGA-BARROS
University of Seville, Seville, Spain