Designing Self-Timed Systems

Alexandre Yakovlev, Leningrad Electrical Engineering Institute, Leningrad, USSR*

As devices are scaled down and become faster while chips grow larger, a number of technological problems involving yield, power dissipation, wire delays versus gate delays, and so on, are being solved. A related set of VLSI design problems involves design complexity, testing, and timing. Of concern here are self-timed systems, that is, systems whose structure is an interconnection of self-timed modules communicating through asynchronous protocols without the use of a common clock. The self-timed design approach is capable of tackling the problems of design complexity, testing, and timing.

This article aims at reviewing the following: several approaches to self-timed VLSI design, examples of self-timed blocks and systems aired so far, and proposals for furthur steps in this area. Apart from this, one of the most important objectives of the article is to draw the designer's and manufacturer's attention to a design strategy that is very promising, although it has probably not yielded even a handful of demonstrated chip design examples to date.

Conceptual Background

Design complexity can be tackled by means of hierarchical and modular design techniques and by regularity of implementations. In order to cope with multidimensionality and granularity, one should consider time metrics and geometry (space metrics) at different levels.

In this environment, testability becomes one of the most important features of design. With device scaling, the number of primary I/O pins per scaled unit area decreases. Since each pin in effect takes more responsibility for ''inland'' affairs, it becomes increasingly important to avoid having to generate exhaustive tests. Ideally, self-testable, self-checking, and self-repairing chips would be built according to organizational principles based on observability issues.

The problem of timing in this context refers to the changing roles of wire delays and gate delays. When all the dimensions on a chip are scaled down by a factor of α, the wire diffusion delay scales up by α^2 and the switching time of devices scales down by α (see, for example, Mead and Conway, 1980; or Seitz, 1979). The increased wire delay relative to device delay slows down a synchronous system that distributes a central clock throughout the system, because the clock rate must be reduced to compensate for the signal skew. One of the ways to solve (better to say ''palliate'') this problem is to geographically divide the system into several subsystems, each having its own clock and operating within the limits of a so-called *equichronic* region. However, we then face the no less significant problem of eliminating the arbitration anomalies and synchronization failures that arise wherever we attempt to interconnect such an ensemble of independently clocked zones.

A class of systems called self-timed systems has been proposed (better to say ''evolved'') to deal with the timing problem (Bryant, 1980; Seitz, 1979, 1980). However, if we consider these recent self-timed design methodologies, it becomes apparent that the other design problems (complexity and testability) can also be solved within the self-timed approach, in a much more structured and natural manner than using the classically accepted synchronous design approach.

A self-timed system is an interconnection of self-timed modules that communicate through asynchronous protocols. It does not require a global clock. Instead, system-level events and protocol states are ordered in time by the causal relationships among the modules. (Inside each of the modules may or may not be a local clock that is supposed to be started asynchronously by system protocol signals.)

Also, a self-timed system is able to provide a high degree of modularity, which enables a straightforward top-down VLSI design to be performed. The natural observability due to spatial (code) redundancy, the presence of completion signals, and the sequential mode of coordination between modules tend to make self-timed systems self-checking and fault-tolerant. It is worth noting that having a self-timed implementation may simplify the testing task, because a malfunction in a self-timed system exhibits itself in components getting hung up on missing acknowledge signals (Rem, 1981).

Definition of a Self-Timed System

Seitz (1980) gives a general definition of a self-timed system as either a self-timed element or a legal interconnection of self-timed systems. Such a highly abstract view has the advantage of being structured and recursive. The definition can be used as a generation (synthesis) rule, or as a recognition (verification) rule. The designer specifies in his own notation a self-timed element's behavior and a general rule for legal interconnections (or a set of all possible legal interconnections).

The main distinctive feature of self-timing lies in the absence of any explicit notion of time metrics at the level of system specification. Bear in mind, however, that this definition does not put any restrictions on the internal structure of a self-timed element, which can be defined at a lower level using an element's local clock. Such lower-level definitions

*On leave during 1984/1985 at the Computing Laboratory, University of Newcastle upon Tyne, NE1 7RU, England.

Ordering relations between signal events:

$x \leqslant y$	event x precedes event y
$x = y$	events x and y are identical (not simultaneous)
$x \$ y$	events x and y are concurrent

Properties of the operator $\leqslant$:

Reflexive:	$x \leqslant x$
Antisymmetric:	$x \leqslant y$ and $y \leqslant x$ implies $x = y$
Transitive:	$x \leqslant y$ and $y \leqslant z$ implies $x \leqslant z$

The weak conditions:

some input becomes defined	$\leqslant$	some output becomes defined
all inputs become defined	$\leqslant$	all outputs become defined
all outputs become defined	$\leqslant$	some input becomes undefined
some input becomes undefined	$\leqslant$	some output becomes undefined
all inputs become undefined	$\leqslant$	all outputs become undefined
all outputs become undefined	$\leqslant$	some input becomes defined

TABLE 1. The "weak conditions": A combinational element signaling scheme (Seitz, 1980).

are also up to the designer.

Recently, a compromised concept of self-timing with a degree of asynchronism has been accepted. This version does not demand that system elements be totally asynchronous. Instead, it requires that the system be asynchronous down to the level of self-timed elements. In order to retain a recursive definition, a self-timed element may not, then, be an interconnection of self-timed systems at the same abstraction level.

We can use one of two possible ways of establishing the legality of interconnections:

1. Provide legality (or correctness) by construction. Although functionally such a system is not very flexible, it is well-structured and, provided with a certain initial state, is globally well-behaved.
2. Build with only some local view of establishing well-behaved elements, and then verify global correctness. This approach can be reasonable for a system of limited complexity.

The traditional concept of correctness is that of providing *safety* and *liveness*. The safety condition can be guaranteed by the local asynchrony of the request-acknowledge signaling protocol used between modules. This being so, checking the liveness condition (freedom from deadlocks) becomes a crucial step in self-timed system analysis.

Seitz (1980) demonstrates an example of the specification of self-timed combinational logic (CL) using the definition introduced. Such a specification is expressed by the so-called *weak conditions* (Table 1) that establish the ordering constraints on signal changes on inputs (due to the environment, or *domain*) and outputs (due to the module, or *function*). The use of temporal logic for the formal specification of weak conditions is presented by Malachi and Owicki (1981).

Although Seitz makes no rigorous definition of what is meant by inputs and outputs, they are likely to be ternary variables having two defined states, zero (0) and one (1), and one undefined state, spacer (-). Such variables can be considered self-timed Boolean variables, i.e., as compostitions of logic and time metrics: Transitions between 0 and (-) and between 1 and (-) are allowed, but transitions between 0 and 1 are not. In order to encode each input (output), a two-rail code may be used, in which 00 represents the undefined state, and

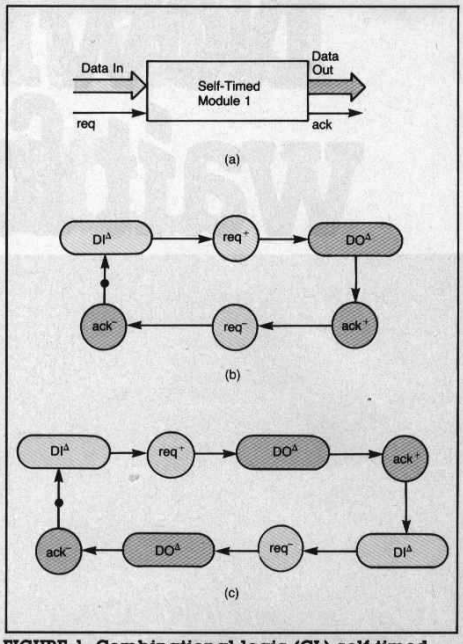

FIGURE 1. Combinational logic (CL) self-timed module (a); 4-cycle protocol (b); and 2-cycle protocol (c).

10 and 01 represent the presence of a zero and one, respectively.

An alternative representation of composite input (output) signaling is done with the use of explicit request-acknowledge wires supporting a single-rail data code. We illustrate such signaling with two signal graphs. [The signal graph (Rosenblum and Yakovlev, 1985) is a formal model obtained from a marked graph (Commoner et al., 1971) by labeling its vertices with the changes of binary or multivalued signals or the vectors of signals.] Figure 1(b) represents the 4-cycle, or return-to-zero (RZ) protocol, while Figure 1(c) represents the 2-cycle, or nonreturn-to-zero (NRZ) protocol. The notations x^+ and x^- denote the change of signal x from 0 to 1 and from 1 to 0, respectively. The changes of composite variables DI^Δ and DO^Δ denote some permutative changes of data input and data output (vector) signals, respectively.

A slightly more generalized method of specification is the self-timed pipeline module [Figure 2(a)], for which a CL module is only a particular case (where signals ack_1 and ack_2 are not used). We can also represent the pipeline module as a CL module with a self-timed rendezvous element (the C-element of David E. Muller) and inverter, as shown in Figure 2(b). (Recall that a C-element's output goes to 0 only after all of its inputs go to 0, and its output goes to 1 only after all of its inputs go to 1.) The signal graph of the corresponding protocol is shown in Figure 2(c). It is clearly seen from the

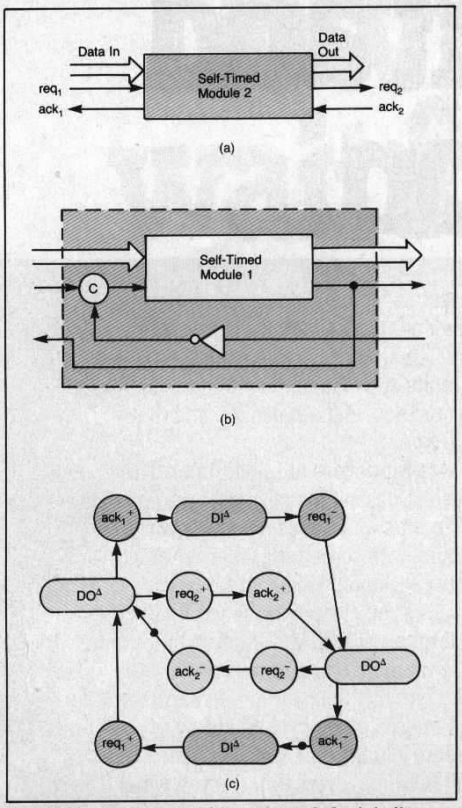

FIGURE 2. Pipeline self-timed module (a); its representation with CL module and C-element (b); and corresponding signal graph of 2-cycle protocol (c).

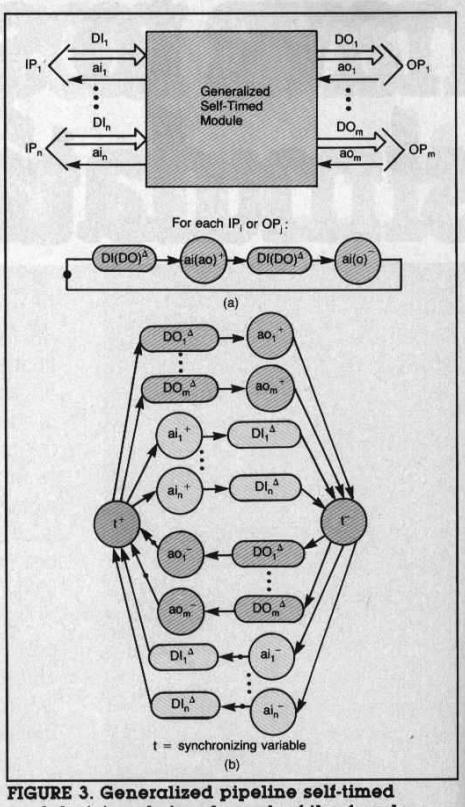

FIGURE 3. Generalized pipeline self-timed module (a) and signal graph of the 4-cycle protocol for the generalized CL module specification (b).

protocol that new input data can be accepted by the self-timed module only if its previous output data was acknowledged by the next module.

The general case of the pipeline self-timed module as an asynchronous process with a finite set of input and output data ports is a model where each port operates according to the 4-cycle protocol—that is, each port has its own data bus and acknowledge wire [Figure 3(a)]. The signal graph of the generalized pipeline CL module specification is given in Figure 3(b). Such a module may have as complicated an internal process behavior as the designer may wish. The behavior of the module can be specified, for example, in terms of CSP (Hoare, 1978) or asynchronous control structure (Jump and Thiagarajan, 1975). However, for this general case, the recursive specification is hardly possible, since the weak conditions are no longer fulfilled for each module, much less for the whole system. The conditions are satisfied for each particular port, and therefore they provide the safety condition between modules. Thus, in general, we may need to undertake the verification of liveness (Friedman and Menon, 1971).

Nevertheless, in this general case we can find some compromisingly restrictive module specification that will simplify the verification procedure. The interconnection of such modules to obtain a pipeline CL system is made using the following rules:

1. Each input port of a module is either driven from an output port of another module, or is an input port to the interconnection;
2. Each output port of a module either drives an input of another module or is an output port of the interconnection;
3. There are no closed port paths.

A pipeline sequential logic interconnection can be defined

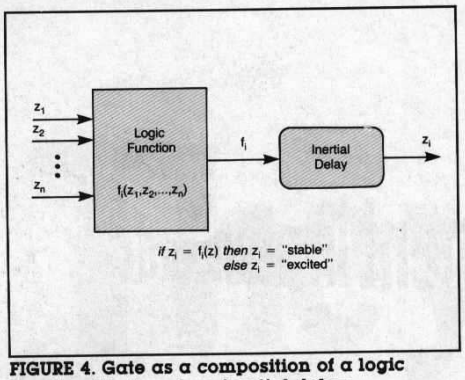

FIGURE 4. Gate as a composition of a logic function block and an inertial delay.

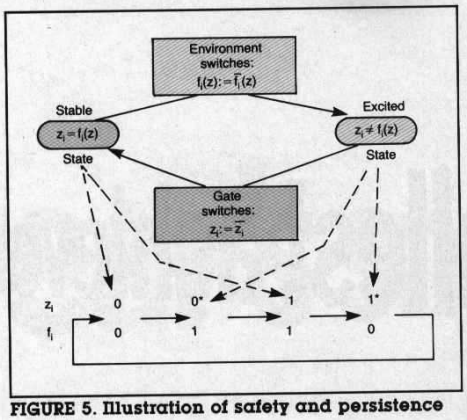

FIGURE 5. Illustration of safety and persistence conditions.

by a similar set of rules. Rules 1 and 2 are unchanged, whereas Rule 3 is restated such that each closed port path must have three or more modules in the loop. This condition is required, because if we have only two modules in a loop, they will always be in a blocking state: Either one of the corresponding ports has data, or both of them are empty.

Having taken into account these definitions and having provided the allowed initial state, we obtain a self-timed system that operates correctly.

The next section is a look back to the history of the self-timed approach. Special attention is paid to the speed-independent circuits having the "strongest" (down to the logic gate delay level) degree of self-timing.

Speed-Independent Circuits

Speed-independent circuits were proposed in the pioneering works of David E. Muller (see, for example, Muller and Bartkey, 1956; Muller, 1963; and Miller, 1965), who is obviously a magician of asynchrony with his magic wand, the C-element.

The main feature of speed-independent circuits is that their behavior does not depend on the real delay values of their gates. That is why gate delays can be unbounded (but finite). Even if they are infinite, the system, having been started, will eventually stop operating but will never operate incorrectly. The correct operation of speed-independent circuits was proved to be free of functional races and delay hazards. It seems important that the assumptions—that gate delays are unbounded and wire delays inside a module are negligible—make the speed-independent circuit model quite adequate for systems of small- and medium-scale integration.

Considering speed-independent circuits from the viewpoint of the contemporary self-timed approach, we notice that Muller's theory of semimodular circuits (the main subclass of speed-independent circuits that is formally convenient for analysis) gave design rules that in some sense were an example of Seitz's general definition.

The three main issues of speed-independent circuits that are relevant to the matter discussed here are the model of the circuit, semimodularity, and legal interconnections.

1. Model of the circuit. A circuit is formally defined as a set of Boolean equations:

$$z_i = f_i(z_1, z_2, \ldots, z_n)$$

where z_i is a logic variable corresponding to the output of a gate. A gate, or logic element, as a unit of the lower abstraction level of a circuit is defined as a sequential composition of a delay-free logic function block and an inertial delay, as shown in Figure 4. The gate is called stable if its state is equal to the value of its function; otherwise, it is excited, and its state is labeled with an asterisk (*).

2. Semimodularity. The circuit is called semimodular with respect to some initial state if for each gate the following operating sequence is carried out:

$$0 \longrightarrow 0^* \longrightarrow 1 \longrightarrow 1^* \longrightarrow 0$$

In other words, once an element has become excited, it may then become stable only by changing the value of its output signal to the complementary value.

On the one hand, this requirement is a safety condition, meaning that the previous excitation must be assimilated before the next change is put to the gate inputs. At the same time, it is also a persistence condition, meaning that the excitation must not be removed from a previously excited gate. Figure 5 illustrates this requirement graphically.

This definition of semimodularity is not an example of a "correct by construction" design rule. It is more accurately a verification criterion, and it has nothing to say about how to build a semimodular circuit from logic gates of a given set. However, Muller gave a theory and formal methods of the implementation of semimodular circuits from the initial specification of a circuit's behavior in terms of state diagrams and transition charts.

3. Legal interconnections. The third issue is a driving rule to the structured approach that has recently been used in self-timed methodologies. A legal interconnection of semimodular circuits is obtained by using a 4-cycle (RZ) request-acknowledge protocol. There are two such legal interconnections producing a composition that is also semimodular: sequential and parallel.

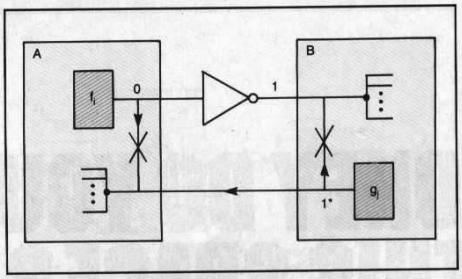

FIGURE 6. Sequential interconnection of semimodular circuits.

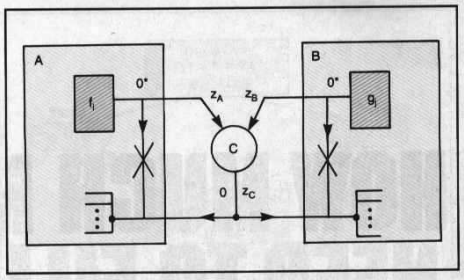

FIGURE 7. Parallel interconnection of semimodular circuits.

Let us consider two semimodular circuits, A and B (Figure 6). We can interconnect them sequentially if we break an output wire of one of the gates (say, f_i) at circuit A and an output wire of one of the gates (say, g_j) of circuit B. We have to make these breaks at points that are before any branching (fan-out) to inputs of subsequent gates. A and B are then interconnected through an inverter, as shown in Figure 6. Provided that the initial state of the gate f_i is stable and equal to 0, and that of the gate g_j is excited and equal to 1, this composition will be semimodular.

Using the same rule of breaking the output wires, we can obtain a parallel connection using the C-element, as shown in Figure 7. We must provide also that the initial states of the gates are both either 0 or 1.

In one of the concluding works by Muller, the general problem of the synthesis of speed-independent circuits was solved (Muller, 1967). However, it is notable that although he introduced a basic 4-cycle signaling scheme, his synthesis methods were concerned with "autonomous" control circuits, i.e., the circuits whose behavior was determined by their structure and the initial states of their gates. In the consideration of data inputs that can be changed from the environment, one may conclude that the circuit will operate correctly for one input discipline, i.e., it is semimodular, but for another discipline it will operate incorrectly.

Post-Muller Studies

Speed-independent circuits were later investigated in works by Armstrong et al. (1969), Dennis and Patil (1971), Keller (1974), Misunas (1973), and Varshavsky et al. (1976).

Armstrong et al. and Varshavsky et al. studied a general problem of the synthesis of asynchronous combinational and sequential circuits in order to solve the problem of open speed-independent circuits. The use of completion detection, which was necessitated by the assumption of unbounded gate delays, was said also to cause the circuits to stop for approximately half of all possible single faults, thus achieving a degree of self-checking.

The proper operation of such circuits required coding of the inputs and outputs. General *m/n* codes were sufficient for the purpose, although the use of such codes might lead to difficult problems concerning fan-in, fan-out, and complexity in the logic. It was rather expected that the use of the double-rail code would lead to much simpler realizations than did *m/n* codes. It was also shown that large systems could be realized by interconnecting (combinational or sequential) modules, each of which generates its own completion signals. Although the primary reason for designing speed-independent circuits was to obtain increased speed, it was established that the (necessary) introduction of completion detectors added roughly as much delay as was already present in the circuits they acted upon. This additional delay partially offset the anticipated speed gain, and it was not clear that a gain would have been achieved in all cases. Even if the speed gain was not significant, the circuits discussed might be useful because of their partial self-checking feature. A special subclass of asynchronous devices, aperiodic circuits, were later proved to be totally self-checking for stuck-at-0(1) faults (Varshavsky et al., 1982).

An indication of the complexity of combinational logic designs was obtained for the case of the double-rail code technique in the implementation of ripple-carry adders (Armstrong et al., 1969; Seitz, 1980). In terms of the number of gate inputs in the circuits (including sum- and carry-completion detectors) the complexity was slightly more than twice that of the uncoded adder. An evaluation in terms of PLA area could show more advantageous results, due to the separation of AND- and OR-plane areas from the input and output drivers, which would be present in any case.

A large contribution was made to the solution of the problem of modular synthesis of speed-independent control circuits based on the systematic "flowchart" design methods, in which a system can be designed and implemented almost directly from a flowchart with little regard for considerations such as semimodularity violations, races, hazards, fan-out, and so on. Such methods were said to be effective in reducing the complexity of the design automation procedure, which otherwise inevitably faces the NP-complete decomposition task. Some of these approaches and their practical applications (for example, ILLIAC I and II, and the PDP-16) were reviewed at length by Banning (1973).

Varshavsky (1985) solved the problem of analysis and synthesis of particular subclasses of speed-independent circuits. For example, he proposed several constructive techniques for implementing semimodular and distributive transition diagrams in circuits assembled from NAND and NOR gates having limited fan-in and fan-out. From the theoretical viewpoint, these results are very important, because they prove some basic homomorphisms between lattice-theoretical temporal specification and commercially available logical

packages.

Keller (1974) proposed a minimal set of speed-independent hardware modules without restricting his method to particular hardware implementations of the elementary gates. This set of primitives can implement sequential and parallel structures. Dennis and Patil (1971), Misunas (1973), and Varshavsky et al. (1976) gave the sets of speed-independent hardware building blocks for the implementation of subclasses of Petri nets and data-flow schemata.

The growth of large-scale integration gave rise to a number of works on asynchronous logic arrays for microprogrammable control. The most outstanding was that of Patil (1975), who proposed parallel asynchronous arrays of microcontrol specified by Petri nets. Each *place* and *transition* of a given Petri net was mapped onto a corresponding cell of the array structure.

Several projects on data-flow machines that are currently in progress exploit a great deal of self-timing and are likely to become VLSI designs.

Kinniment (1981) proposed a PLA realization of Petri net control, including arbitration circuits, and made an important conclusion concerning the complexity of asynchronous design: asynchronous control is reasonable for applications where the time and area overheads are small as a proportion of the system's total, evaluated at the "processor-memory-switch" level or above. As if following this recommendation, there are several works on asynchronous packet networks with speed-independent routers (Leung, 1979; Chu, 1983) designed for data-flow processors where all the blocks communicate in speed-independent fashion (Dennis and Misunas, 1975).

Some research has been carried out on speed-independent bus interfaces operating in "one-to-one" and "one-to-many" (broadcast) modes. Several data encoding and bus arbitration techniques were investigated (Banning, 1973; Sutherland et al., 1979; and Varshavsky, 1985).

Compromised Techniques

Many researchers in the area of self-timed systems have come to the conclusion that the speed-independent implementation of self-timed modules may lead to very high overheads. As a result, compromised techniques are acceptable, especially where the economy of area is more important than the advantages of pure speed-independence. So-called pseudo-asynchronous models, such as the delay-model combinational element and the internally clocked pipeline processor module, were proposed by Seitz (1980).

Ha and Reddy (1984) gave self-timed implementations for combinational and sequential modules with double-rail coding that are highly reliable in the speed-independent sense. They also used an advantageous principle of partitioning a combinational self-timed model into a functional part and a control part. They showed the appropriateness of a single-rail code approach for the implementation in PLAs, as the functions were expressed easily in sum-of-products form.

The use of "double-rail gates" may be suitable for a standard-cell approach. Modules realized by the proposed methods are efficient in area and in speed compared with other existing methods. The attractiveness of a testable design of combinational self-timed modules using PLAs was shown by Ha and Reddy, in that the design does not require extra circuitry to be testable. The test inputs are derived by a very efficient procedure and detect all single cross-points, stuck-at faults, and adjacent bridging faults. (It is obvious that since such a system is no longer speed-independent in Muller's sense and hence is not self-checking at the gate level, then it is required that test input sequences must be generated for error detection).

One of the most acute of today's problems is the problem of hierarchical self-timed design methodology. A solution was attempted by Lister and Alhelwani (1985), who developed a high-level language for the specification of self-timed systems and a formal method of transformation to a well-structured data-driven Petri net, which then could be mapped onto a set of hardware primitives, allowing the using of clocked circuits by specially embedding them into a self-timed control environment.

According to Seitz, "the greatest development in the theory of self-timed systems of the past several years" is the results obtained by Van de Snepscheut (1983). He proposed the very attractive method of deriving a circuit from a program. The resulting circuit consists of a number of smaller circuits interconnected by wires of unrestricted length. These smaller circuits consist of modules situated within isochronal regions. Although the formalism for this approach is based on rather sophisticated trace theory, in practice it gives very applicable results, especially with regard to complexity management and modularity issues.

Some other interesting examples of using formal techniques for self-timed system specification, validation, and design are by Malachi and Owicki (1981), Shostak (1983), and Barton (1981).

Practical Designs

Several projects on data-flow machines that are currently in progress exploit a great deal of self-timing and are likely to become VLSI designs. For example, at the Massachusetts Institute of Technology, a static data-flow machine built on the principle of a ring structure uses a speed-independent interconnection of separate blocks (Dennis and Misunas, 1975). It is believed that the self-timed design methods using basic asynchronous modules (Patil and Dennis, 1972) can lead to fewer design errors and greater confidence in correct translation of specifications into masks for chip manufacturing. This should also lead to a significant reduction in the number of design iterations required to perfect a device, and a lower investment for the manufacture of custom circuits. It seems likely that the layout of masks should be done by copying and interconnecting standard cells that are universal building blocks for asynchronous logic.

Another example is the data-driven machine (DDM1, DDM2) built at the University of Utah (Davis, 1978). This machine was based on several principles of recursive architecture proposed originally by Glushkov et al. (1974). An

additional principle was augmented by Davis, and it demanded that modules of recursively structured machines should function in a fully distributed asynchronous fashion. DDM1, as a version of a processor-store-element structure, was implemented and later was transformed into DDM2, which is a completely self-timed data-driven machine built with LSI nMOS using the PPL design system (Hayes, 1983).

An example of a totally self-timed static RAM design was presented by Frank and Sproull (1983). Although the memory array uses a conventional six-transistor static cell, special completion signaling circuitry is provided for each column and each row to make the memory self-timed. The time losses on various completion detections are partly compensated for by a special sense amplifier. [The idea for the amplifier came from another interesting example of asynchronous design, Mostek's 38000, a 256-K ROM (Scherpenberg and Sheppard, 1982)]. This sensing scheme reportedly tolerates process variations, thereby increasing reliability and demonstrating a main advantage of the self-timing discipline. Quite encouraging results of area overhead for detection circuitry of only 5.2 percent of the total chip area were obtained for the RAM. Self-timed read and write cycles are 440 ns and 395 ns, respectively. This design has proved the concept of building systems with self-timed memory as integral on-chip components. The single-chip data-driven processor that is being constructed at Carnegie-Mellon University (Frank, 1982) includes such a memory and is also reportedly self-timed.

In the area of communications, self-timing can be successfully applied to conventional multiprocessors with a common switching network or with backplane buses...

In the area of communications, self-timing can be successfully applied in conventional multiprocessors with a common switching network or with backplane buses where communication between processes running concurrently in different processors occurs through shared variables and common access to one large address space (Futurebus, 1983). Self-timing can also be applied in message-passing multiprocessors where a network of point-to-point communications channels is used (Seitz, 1985).

Further Work

It is still necessary to deal with several research problems in hierarchical self-timed design, viz., high-level self-timed hardware description languages, verification tools, reasonable trade-offs between embedded self-checking and efficient augmented testing, and efficient testing methods at the system level. Still of concern are the problems in self-timed backplane bus structures, especially a design methodology for chip-level and board-level interfaces for a fault-tolerant asynchronous bus.

We hope that designers and manufacturers of the new generation of data-flow and conventional VLSI machines will pursue the benefits of self-timed design principles. □

References

Armstrong, D.B., et al. December 1969. "Design of Asynchronous Circuits Assuming Unbounded Gate Delays," *IEEE Transactions on Computers*.

Banning, J.P. January 1973. "Asynchronous Modular Systems," *TR-116*, Princeton University, Princeton, NJ.

Barton, E.E. 1981. "Non-metric Design Methodology for VLSI," *VLSI-81*, ed. by J.P. Gray, Academic Press, London.

Bryant, R.E. 1980. "Report on the Workshop on Self-timed Systems," *TM-166*, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA.

Chu, T.A. February 1983. "The Design Implementation and Testing of Self-Timed Two by Two Packet Router," *Memo 225*, Computation Structures Group, M.I.T.

Commoner, F., et al. October 1971. "Marked Directed Graphs," *Journal of Computer and System Sciences*.

Davis, A.L. April 1978. "The Architecture and System Method of DDM1: A Recursively Structured Data-Driven Machine," *5th Annual Symposium on Computer Architecture*, New York, NY.

Dennis, J.B., and S.S. Patil. 1971. "Speed-Independent Asynchronous Circuit," *4th Hawaii International Conference on System Sciences*.

Dennis, J.B., and D.P. Misunas. January 1975. "A Preliminary Architecture for a Basic Data Flow Processor," *2nd Symposium on Computer Architecture*, New York, NY.

Frank, E. H. October 1982. "The Fast-1: A Data-Driven Multiprocessor for Logic Simulation," Thesis Proposal, *VLSI Memo 122*, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.

Frank, E.H., and R.F. Sproull. March 1983. "A Self-Timed Static RAM," *3rd Caltech Conference on VLSI*, Pasadena, CA.

Friedman, A.D., and P.R. Mennon. January 1971. "Systems of Asynchronously Operating Modules," *IEEE Transactions on Computers*.

Futurebus. November 1983. "Specifications for Advanced Microcomputer Backplane Buses," *IEEE 896, Draft 6.2*.

Glushkov, V.M., et al. 1974. "Recursive Machines and Computing Technology," *IFIPS Proceedings*, North Holland, New York, NY.

Ha, D.S., and S.M. Reddy. October 1984. "On Testable Self-Timed Logic Circuits," *IEEE International Conference on Computer Design*, Port Chester, NY.

Hayes, A.B. March 1983. "Self-Timed IC Design with PPLs," *3rd Caltech Conference on VLSI*, Pasadena, CA.

Hoare, C.A.R. August 1978. "Communicating Sequential Processes," *Communications of the ACM*.

Jump, J., and P.S. Thiagarajan. 1975. "On Interconnection of Asynchronous Control Structures," *Journal of the ACM*.

Keller, R.M. January 1974. "Toward a Theory of Universal Speed-Independent Modules," *IEEE Transactions on Computers*.

Kinniment, D.J. 1981. "Regular Programmable Control Structures," *VLSI-81*, ed. by J.P. Gray, Academic Press, London.

Leung, C.K.C. 1979. "On a Design Methodology for Packet Communication Architectures Based on Hardware Design Language," Computer Structures Group, Laboratory for Computer Science, M.I.T.

Lister, P.F., and A.M. Alhelwani. January 1985. "Design Methodology for Self-Timed VLSI Systems," *IEEE Proceedings*.

Malachi, Y., and S.S. Owicki. 1981. "Temporal Specifications of Self-Timed Systems," *VLSI Systems and Computations*, H.T. Kung et al., eds., Computer Science Press, Rockville, MD.

Mead, C., and L. Conway. 1980. *Introduction to VLSI Systems*. Addison-Wesley, Reading, MA.

Miller, R.E. 1965. *Switching Theory, Vol. 2*, Wiley, New York.

Misunas, D.P. August 1973. "Petri Nets and Speed-Independent Design," *Communications of the ACM*.

Muller, D.E., and W.C. Bartkey. 1956. "A Theory of Asynchronous Circuits," *Report No. 75*, the University of Illinois.

Muller, D.E. March 1963. "Asynchronous Logic and Applications to Information Processing," *Symposium on the Applications of Switching Theory in Space Technology*, Stanford University Press.

Muller, D.E. 1967. "The General Synthesis Problem for Asynchronous Digital Networks," *8th Symposium on Switching and Automata Theory*, New York, NY.

Patil, S.S., and J.B. Dennis. September 1972. "The Description and Realization of Digital Systems," *6th Annual IEEE Computer Society Conference*, San Francisco, CA.

Patil, S.S. 1975. "Micro-control for Parallel Asynchronous Computers," *Euromicro*, The Netherlands.

Rem, M. 1981. "The VLSI Challenge: Complexity Bridling," *VLSI-81*, ed. by J.P. Gray, Academic Press, London.

Rosenblum, L. Ya., and A.V. Yakovlev. July 1985. "Signal Graphs: From Self-Timed to Timed Ones," *International Workshop on Timed Petri Nets*, Torino, Italy, in print.

Scherpenberg, F.A., and D. Sheppard. June 2, 1982. "Asynchronous Circuits Accelerate Access to 256K Read-only Memory," *Electronics*.

Shostak, R.E. March 1983. "Verification of VLSI Designs," *3rd Caltech Conference on VLSI*, Pasadena, CA.

Seitz, C.L. January 1979. "Self-Timed VLSI Systems," *1st Caltech Conference on VLSI*, Pasadena, CA.

Seitz, C.L. 1980. "System Timing," *in Introduction to VLSI Systems*, Addison-Wesley, Reading, MA.

Seitz, C.L. January 1985. "The Cosmic Cube," *Communications of the ACM*.

Sutherland, I.E., et al. January 1979. "The TRIMOSBUS," *1st Caltech Conference on VLSI*, Pasadena, CA.

Van de Snepscheut, J.L.A. October 1983. "Trace Theory and VLSI Design," Ph.D. Thesis, Technological University of Eindhoven, The Netherlands.

Varshavsky, V.I., et al. 1976. "Aperiodic Automata," Moscow: Nauka (in Russian).

Varshavsky, V.I., and L.Y. Rosenblum. 1976. "Dead-beat Automata and Asynchronous Parallel Processes Control," *1st IFAC-IFIP Symposium*, SOCOCO-76, Tallinn, USSR.

Varshavsky, V.I., et al. 1982. "Totally Self-checking Asynchronous Combinational Circuits and Indicatability," *Automation and Remote Control*, Vol. 43.

Varshavsky, V.I. 1985. "Hardware Support of Parallel Asynchronous Processes," Lecture Notes, Helsinki University of Technology (in print).

About the Author

Alexandre Yakovlev is an assistant professor of computer science at the Leningrad Ulyanov Electrical Engineering Institute (LUEEI), where he is researching asynchronous control structures and fault-tolerant interface protocols. This year he was on leave to the Computing Laboratory at the University of Newcastle, England, where he worked with the university's VLSI design tools; self-timed systems; fault-tolerant wafer-scale architectures; and IEEE 896 (Futurebus) backplane-bus interf specification and design. He received his M.Sc. and Ph.D. degr in computer science from LUEEI.