Royal Netherlands Academy of Arts and Sciences
P.O. Box 19121, 1000 GC Amsterdam, the Netherlands

Proceedings of the workshop,
Amsterdam, 10 – 14 November 1991

Organization committee

Charles Molnar, Washington University, visiting at Eindhoven University of Technology
Martin Rem, Eindhoven University of Technology
Huub Schols, Eindhoven University of Technology

ISBN 0-444-85751-6

Koninklijke Nederlandse Akademie van Wetenschappen
Verhandelingen, Afd. Natuurkunde, Eerste Reeks, deel 37

Design and Implementation of Asynchronous Circuits

M. van der Korst, A. Peeters and H. Schols

North-Holland, Amsterdam/Oxford/New York/Tokyo, 1992

Contents

Acknowledgement 3

Preface 4

Opening Speech 6

1 Goals and Expectations 7
- 1.1 Asynchronous . . . 7
- 1.2 Challenges . . . 9
- 1.3 Design Style and Management . . . 9
- 1.4 University and Industry . . . 10
- 1.5 Theory . . . 10
- 1.6 Tools . . . 10

2 Physical and Abstract Models 11
- 2.1 Uniform Definitions . . . 11
- 2.2 Computational Models and the Real World . . . 12
- 2.3 Phase Diagrams and Transitions . . . 12

3 Formal Models 15
- 3.1 Time . . . 15
- 3.2 Specification Languages . . . 15
- 3.3 Specification Models . . . 16
- 3.4 Nonputs . . . 17
- 3.5 Liveness . . . 17
- 3.6 Aspects of Formal Models . . . 18
- 3.7 DI-Algebra . . . 18

4 Transformation Methods 21
- 4.1 Motivation . . . 21
- 4.2 Architecture . . . 22
- 4.3 Transparency . . . 23
- 4.4 Performance of the Tangram Compiler . . . 23
- 4.5 Discussion . . . 24

5 Synchronous and Asynchronous **26**
5.1 Pedagogical Tools 26
5.2 Closing the Gap 28
5.3 Design 28
5.4 Al Davis's Design Methodology 29
5.5 Software Tools 30
5.6 Testing 30

6 Science and Industry **32**
6.1 Cooperation 32
6.2 Educational Problem 32
6.3 Different Views 33
6.4 Contact 33
6.5 Uniform Notation 34

7 Concluding Remarks **35**
7.1 Present and Future 35
7.2 Conclusions from Session 2 35
7.3 Conclusions from Session 3 35
7.4 Conclusions from Session 4 36
7.5 Conclusions from Session 5 36
7.6 Conclusions from Session 6 37
7.7 Interaction between Research Groups 38
7.8 Publications 38
7.9 Technical Support 38

Bibliography **41**

A List of Participants **42**

B Asynchronous Bibliography **45**

Acknowledgement

We would like to acknowledge the financial and organizational support of the Royal Netherlands Academy of Arts and Sciences and the Department of Mathematics and Computing Science of Eindhoven University of Technology. Especially, the effort put forward by Manita Kooy and Franka van Neerven constituted a firm basis on which this workshop has been built. Furthermore, we are grateful to Michiel van der Korst and Ad Peeters, who did a great job by preparing the major part of these proceedings in a short time.

On behalf of the organization committee,

Huub Schols

Preface

This workshop is the fifth in an informal series of workshops held in Geldrop, Netherlands (1984), Eureka, Missouri (1985), Paterswolde, Netherlands (1986), and La Jolla, California (1988). It was planned to recognize and further stimulate the evolution of understanding of formal and technical issues in asynchronous system and circuit design towards a mature methodology for specification and design of high-performance information-processing systems. Toward this end, participants were chosen to represent a span of interests ranging from university researchers in abstract process algebras to industrial designers and builders of major asynchronous computer systems. An unusual feature of our workshop plan was to disallow the presentation of formal papers, that tend to emphasize results. Instead, a number of session topics focussed on issues, methods, and current challenges were chosen, and a discussion leader was selected and charged with initiating and provoking discussion of this topic. This approach, though regarded with concern and scepticism by some of the invitees, succeeded rather well. In the words of one of the sceptical participants in his report to his home organization:

> I was a little worried that the format would lead to lots of gyrations on minor points, which tends to happen with large, free-format meetings. I think the subject matter and seriousness of the participants made it work. The provocateurs were also well chosen, in that they were aware of the important issues in their areas and saw that they came up in the discussion.

The unusual approach in turn raised unusual problems in developing this report of the workshop. Since there were no formal presentations, they could not simply be collected and bound together. No formal mechanisms were used to generate and validate conclusions that could be presented as a formal consensus. Yet the sessions generated much intense discussion and most participants seemed to share at least a sense of where the important issues critical to progress in research and exploitation were to be found; in many cases they even agreed on the steps to be taken.

This proceedings should be looked at as a serious journalistic attempt to capture the flavor and major contents of the discussions that took place. Despite the occasional attribution of points and comments to individuals, this document should not be interpreted as a reliable indicator of the individual origin or advocacy of ideas. Nor should the anonymously reported discussions be taken as necessarily reflecting agreement among the participants. The report also contains some clarifications and extensions of workshop discussions outside of the formal workshop sessions.

Taken with these caveats, I find that the following report, as a whole, reflects very well the spirit of the workshop and the high enthusiasm of its participants about the future of the area. Most of the credit for this report should go to the designated reporters, Ad Peeters, Huub Schols and Michiel van der Korst, who were specifically charged with making detailed notes throughout the workshop sessions. The refinement of their initial drafts of session summaries was assisted by the comments from workshop participants in response to first and second drafts of the report. The reader should find the selective bibliography a particularly valuable complement to the reports on the session discussion.

The enthusiasm and flexibility of the Royal Netherlands Academy of Arts and Sciences and Eindhoven University of Technology in their sponsorship of this somewhat unusual program are hereby gratefully acknowledged, as is the willingness of the participants to engage in this workshop with its occasional 'free-for-all' exchanges of opinion.

Charles E. Molnar
co-organizer

Opening Speech

Opening speech by Professor dr. P. J. D. Drenth, president of the KNAW.

On behalf of the Royal Netherlands Academy of Arts and Sciences I extend a hearty welcome to all participants of the Workshop on the Design and Implementation of Asynchronous Circuits.

The subject of asynchronous circuits has been studied for almost as long as digital computers exist. Many of the participants to this workshop have been instrumental in shaping the subject into what it now is: a challenging and promising technology, but one that has thus far been hardly applied in commercial applications. Recently, however, we have witnessed a revived interest in this type of circuits. This was, for instance, very visible in Professor Sutherland's speech, which he delivered when he received the Turing Award in 1989. One of the questions that this workshop faces is how the international research community should react to this upsurge in attention for asynchronous circuits.

There are a number of research groups in the world that work on asynchronous circuits. I am very glad that most of these groups, from different continents, are represented at the workshop. I am also pleased that this workshop brings together researchers both from academic institutions and from industry. In that sense I see this workshop as a collaborative effort among the researchers in the world to push the subject of asynchronous circuits further towards maturity.

This workshop is the fifth in a sequence. The other four were held in Eindhoven, Saint Louis, Groningen, and La Jolla (California). They were all aimed at fostering the scientific development of asynchronous circuits. This fifth one is also aiming at giving new impulses to their industrial application. The time seems ripe for it.

I wish you all a productive workshop.

Session 1

Goals and Expectations

Session chair: **Martin Rem**

Rationale, goals, and expectations of the workshop.
Assertion of positions, conjectures, and conclusions.

1.1 Asynchronous

The adjective *asynchronous* is used very loosely. Circuits have been called asynchronous (to distinguish from synchronous) whenever they didn't contain a global clock. As a consequence, the area to which the term *asynchronous* applies is very large; applications vary from interfaces between systems with different clocks to clock-free designs. Although it may be possible to use asynchronous circuits wherever clocked circuits have been used traditionally, in general, they will be employed in more specific, restricted areas.

It is hard to pinpoint the most promising aspect of asynchronous circuits. Probably, some aspects are more important in some cases, whereas other aspects are very interesting in other cases. We list the possible advantages and disadvantages of (the use of) asynchronous circuits compared to (the use of) synchronous circuits.

+ Rapid design.

+ Better composability of parts. There exists no implicit synchronization due to the absence of clocks. As a consequence, there are less constraints for the composition.

+ Reliability of design process. Recent experiences with the design of asynchronous circuits (especially delay-insensitive circuits) indicate that very few design errors are made. As a consequence, less debugging is needed.

+ Low power dissipation. A clock may dissipate up to 80% of the power of a circuit. Power consumption (dissipation of energy) tends to become much more expensive than area due to the costs of packaging for high energy dissipation.

+ Design by non-experts (silicon compilation).

+ Performance improvement easier to achieve. The performance of a system can be improved by performance improvement of a subsystem. When the global clock has to be speeded up, in general, all parts have to be adjusted to this.

+ Speed of the circuit. Asynchronous circuits tend to reflect average case behavior rather than worst case behavior (the only thing a clock can do is to slow down a computation). However, the choice of a particular design style may slow down the circuit. For example four-phase handshake protocols, which are frequently used in asynchronous design, usually yield a lower speed of the circuit than two-phase handshake protocols, due to the doubling of the communication protocol.

+ Robustness of design. Properly designed asynchronous circuits can be more robust, that is, less sensitive to variations in, for example, power supply and temperature. Mostly, such reports reflect the point that variations of timing due to variations of circuit parameters and conditions cause changes in system speed without loss of functional correctness.

o Concurrency. In asynchronous circuits it is easier to achieve concurrency due to the absence of implicit synchronization. On the other hand, reasoning about asynchronous concurrency is harder than reasoning about sequential designs, due to the absence of the guaranteed existence of a global state of an asynchronous circuit.

o Testability. Detection of fabrication errors tends to be easier, since an asynchronous circuit that contains a production error usually comes to an effective halt, viz. a deadlock.

– Area. Asynchronous circuits typically have more transistors than synchronous circuits. The amount of area needed for a circuit is mainly determined by the amount of wiring, not by the number of transistors. Nevertheless, since asynchronous circuits tend to be more irregular than synchronous circuits, they need more wiring.

With respect to testability we would like to make some more remarks. To do a scan test one needs to be able to stop a circuit. Without additional test-circuitry it is practically impossible to stop asynchronous circuits. Difficulty of testing also depends upon the encoding of data: dual rail allows for easy tests, whereas data bundling complicates testing considerably.

Synchronous circuits are state-based; asynchronous circuits are parallel transition oriented. From a state-based point of view, asynchronous components form a special case of the synchronous components: no state is a successor to itself. From a transition-based point of view, synchronous components form a special case of the asynchronous components: some transitions are clock signals. Statements like these probably are of academic interest only. They may say more about the background of the author than about the problem that is addressed.

We conclude that we need to establish a consensus between people that work in the area of asynchronous design about a definition of this area and its advantages. Such a consensus might enable us to advocate successfully the design of asynchronous circuits.

1.2 Challenges

In order to get asynchronous design techniques accepted by mainstream designers, the research community should elaborate its potential. Several suggestions have been made:

- Design an asynchronous chip that is pin-compatible with an existing clocked chip.
- Make something challenging to get people interested.
- Solve a yet unsolved problem in order to impress industry.
- Solve a hard problem within the constraint system of industry before a synchronous solution is found.

All suggestions seem very hard to achieve, since a lot of knowledge and experience has been accumulated in traditional clocked design. It seems to be more promising to predict the future needs and have solutions ready in time; industry will take care of today's needs.

We should not aim at providing an alternative for synchronous design: synchronous design will stay. We have to address the problems in synchronous design, for example the mixing clock problem. In this way the separation between asynchronous and synchronous designs will disappear. Mixed (asynchronous and synchronous) designs have the future. Asynchronous communication between locally synchronous parts is an example of such a mixed design, cf. Macro-Modules [CM74] and Self-Timed design [Sei80]. The goal is to establish one complete design space in which synchronous and asynchronous parts are integrated.

1.3 Design Style and Management

Design of asynchronous circuits is very well suited for a modular approach, in which a system is designed as an interconnection of modules. Reliable modules can be designed by hardware designers, who have detailed knowledge of the implementation technology. The system can be designed by persons who best understand the function of the system. This modular approach is facilitated by the asynchrony of the circuits: performance and correctness aspects of time are separated. The modular approach enables design of asynchonous circuits that are correct by construction.

The issues above actually are management issues. The design of a concurrent system can be hierarchically factorized into the concurrent design of modules. This requires an early specification of the interfaces between the modules. This will give rise to changes of specifications of interfaces. Design management is a key issue here. Concurrency is also a

management issue; overspecification is a problem, for example in unnecessary ordering of events (reducing concurrency).

The similarity between hardware design and software design is increasing every week. However, it is misleading to view circuits as programs, since circuits are more general than programs. In order to reason about parallel systems at a low level, a language with sufficient expressive power is needed. We might want to develop one language that can be used at all design stages.

1.4 University and Industry

There are several ways in which universities advocating asynchronous design can improve their contacts with industry. Joint projects form a direct way to do so. In general, universities should refrain from preaching revolution, since the turn-over is too expensive; furthermore, convincing people is too hard, since industry is way ahead of universities in solving today's problems. Finally, the large gap between algebraic formalisms (traces, lattices) and differential equations has to be bridged.

It is the responsibility of universities to educate. In this way, people will get to use new methods and concepts, and they will accept formal structures. Universities should not only present theories and models, but also put effort into interpreting the theoretical results and formalisms. The scope of the problems that are studied at universities has to be broadened. Industry is not interested in partial solutions: the combinatorics of integrating them always gets you.

1.5 Theory

We should find ways to convince ourselves —and others— of the integrity of our formal models. If there are fundamental properties that a formal model fails to capture, the predictions made on basis of such a model should be approached with caution (or suspicion). We should pinpoint the theoretical challenges (or obstacles) with respect to asynchronous circuits that are the most important at this moment. Maybe, the state of the art is such that the challenges and obstacles lie primarily in the exploitation of the theoretical results.

1.6 Tools

A lot of tools (for example RISC, SPICE) have been designed at universities. After they have become accepted, companies took them over and developed them further to the size they have now. Universities should build tools that support their ideas about asynchronous design. There is no need for developing a commercial tool; just a prototype will do, if it solves the problem. Once it is accepted by industry, industry will commercialize and standardize it.

Session 2

Physical and Abstract Models

Session chair: **Charles Molnar**

Relation between physical models and abstract models.
Motivation for playing formal games.
Interpretation of the results.

When a formal model is used for discussing asynchronous circuits, the question arises what the relation is between the formal model and the VLSI-circuit. What is the real world equivalent of a transition, and up to what level should one go? Should we go down from the gate level through the transistor to quantum physics?

The level of abstraction depends on the question that is to be answered. If one considers the problem of metastability, modelling a circuit as a dynamical system will be effective. In the design of systems one would, however, prefer to work with a model of circuits on the level of boolean functions. These higher levels of abstraction introduce the danger that the underlying physics are forgotten, and that problems like metastability are disregarded.

The concept of Delay-Insensitivity (DI) is more an engineering method than a real world model. It provides a way to separate concerns (especially timing problems). But DI does seem to have a meaning, in the sense that several different definitions turned out to be equivalent.

2.1 Uniform Definitions

In order to stimulate research and education in the field of asynchronous circuits consistent definitions should be formulated that are as precise as possible. Some basic concepts are:

- primitives:
  - components,
  - wires, and
  - events.

- speed independence: the functional correctness of the network is independent of the variations in the response time of the components.
- delay insensitivity: speed independent *and* independent of delays in the wires.

2.2 Computational Models and the Real World

When a computational model or formalism is used as an abstraction for real world objects the relation between these two must be understood thoroughly if conclusions about the model are to be reliable statements about the real world. In this relation dynamical systems can be used as an intermediate mathematical model:

$$\textit{Computational Models} \overset{M}{\longleftrightarrow} \textit{Dynamical Systems} \longleftrightarrow \ldots \longleftrightarrow \textit{Real World}$$

Relationship M would not solve the ultimate problem of relating the computational model and the real world, but could be a useful step. It is rather amazing that this has not already been done in an adequate, rigorous, general, and elegant way. This is a way of confessing that we do not know what the real world is; the relationship toward the center and the right hand side of the diagram have been the object of the study of physics for centuries. We should use that and not throw it away.

2.3 Phase Diagrams and Transitions

An example of a model/representation that provides a link between between dynamical systems and formal models is the phase diagram, as used in [Bro89]. This diagram captures a lot of issues concerning transitions in signals that carry the communication between a component and its environment.

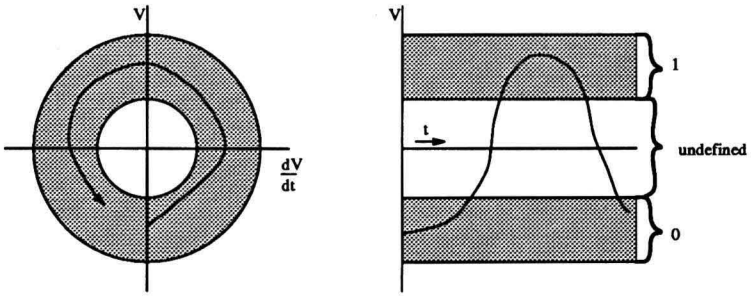

Figure 2.1: Phase diagram (left) and timing diagram (right) for a valid transition

In Figure 2.1 the phase diagram on the left represents the relation between the voltage level, V, and the first time derivative, $\frac{dV}{dt}$, of the voltage transition in the timing diagram

on the right. In the phase diagram the gray area represents a useful restriction on the behavior of the signal. When the voltage level is in one of the two voltage areas logically defined as 0 or 1, the voltage rate-of-change may be zero, whereas in the undefined area the voltage change must always be directed consistently towards one of the two voltage levels, out of the *undefined* area.

In such a phase diagram an *event* can be defined as the passing of the $\frac{dV}{dt}$-axis, when the voltage level is zero.

The definition of the valid area in the phase diagram captures a lot of issues concerning the integrity of transitions. Two of these issues are:

1. the transition must be monotonic, and

2. the derivative is bounded.

The inner border of the valid area of the phase diagram forces the transition to have a minimum speed when the voltage level is in the undefined area, thus forcing the transition to be monotonic. If an event is defined as the passing of the $\frac{dV}{dt}$-axis, the monotonicity restriction will keep the number of events per transition down to one. An example of an non-monotonic transition and the corresponding path in the phase diagram are shown in Figure 2.2.

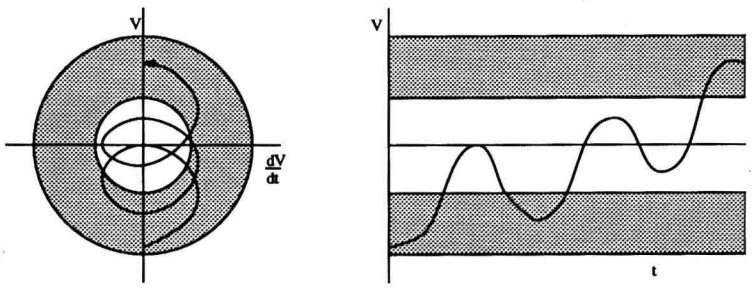

Figure 2.2: Non-monotonic transition

The borders of the valid area of the phase diagram define the upper and lower limit on the speed of transitions. An upper limit on the speed of transitions is needed, because when the transition is too steep this may lead to inappropriate reactions, such as non-monotonic transitions later in the network. A lower limit on the speed of transitions is also needed, because when a transition is too slow this may provide problems when different components use different thresholds, or when noise is present.

These cases are presented in Figures 2.3 and 2.4 respectively.

The phase diagram can be used for both synchronous and asynchronous circuits. In the case of synchronous circuits only discrete points in the trajectory, as defined by the clock, are considered. Synchronous and asynchronous circuits can be seen as different forms of abstraction of the picture. The phase diagram is a mathematical model in the sense that we are looking at sets of trajectories, that can be used to restrict the allowed behaviors in such

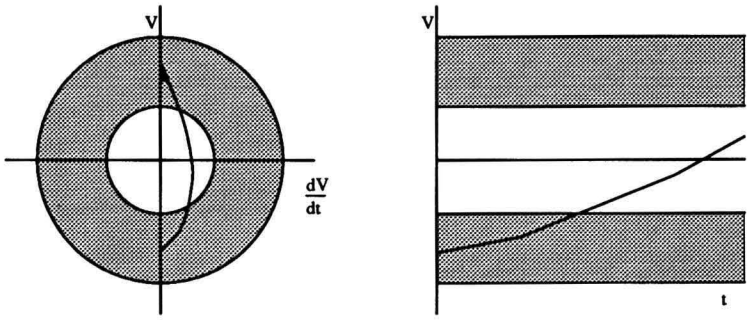

Figure 2.3: Transition too slow

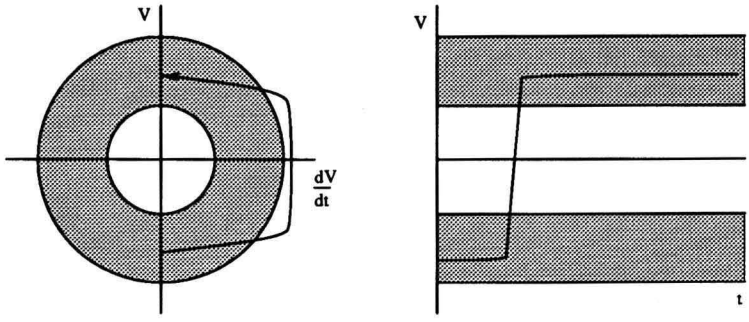

Figure 2.4: Transition too steep

a way that a rigorous definition of the relationship of continuous behavior and models that use abstractions such as events or discrete states can be unambiguously defined. Making such a relation exact and formal may help to illuminate the role of time in modeling at the more abstract levels.

Session 3

Formal Models

Session chair: **Jan Tijmen Udding**

Specification of asynchronous circuits.
Expressive power of formal models.
Distinction between models.

3.1 Time

With respect to *time* we can distinguish two concerns, viz. *correctness* and *performance*. In the interior of the modules one needs to be concerned with time in order to avoid hazards, critical races, etc. This is the correctness aspect of time, also called order and sequence. On the other hand, one can be concerned with performance (speed), which in a sense is also a correctness criterion.

In an asynchronous setting, one can make a network of modules faster by speeding up modules that are in the critical path. In a clocked circuit timing analysis of the network has to be done before the clock can be adjusted. This requires a vast amount of extra work. In asynchronous circuits, the performance and correctness aspects of time are separated by keeping timing problems inside the modules.

3.2 Specification Languages

Problems with inherent (asynchronous) concurrency are often specified by means of CSP-like programs. In contrast to CSP [Hoa85], which is a well-established term, the notion of CSP-like programs is confusing, since many variants with subtle differences exist. It would be good to have a standard notation like e.g. Occam, which in itself is a CSP-like language. Programs written in Occam can be easily understood and are better suited for specification purposes than, e.g., I-nets. It would be hard to express and understand one page of Occam text using I-nets. As a standard notation, however, Occam has some drawbacks, mainly caused by the close link to its target hardware: Transputers (see e.g. Section 3.4).

Some participants of the workshop argued that transformations and design-steps should be expressible inside one specification language. It is not yet possible to use one notation for the whole design trajectory. Although a uniform notation may be a nice goal, there are some things that plead against it. The design process involves decisions at different levels of abstraction, as far apart as system architecture and transistor lay-out. We do not expect to encompass everything in a single language, but want to have different languages for different levels, and a way of relating these levels. CSP and Petri-Net based notations are candidate languages for describing input/output behavior of circuits in terms of voltage-level transitions or handshakes.

3.3 Specification Models

In asynchronous circuit design, a lot of different specification models are used. Some (most?) of them are listed below. All specification techniques listed here are free of a time metric.

Petri nets

Petri nets occur in many different forms, which all have slightly different interpretations. Some established terms are I-nets, signal-transition graphs, and sequence nets. All these nets have in common that they are networks of places, transitions, and labels. In I-nets, labels represent transitions.

Petri nets are well suited for specification purposes, because the problem of combinatorial explosion is avoided by its inherent high degree of concurrency. Moreover, Petri nets can be simulated, which can be helpful in validating specifications, or at least in building confidence. As long as there is a gap between the formalisms used for specification and those used for design, we can only argue informally about the correctness of a design.

A trace set can be regarded as a set of firing sequences and thus as a canonical representation of nets. In this sense, a disadvantage of Petri nets is that representation is not unique. An important issue therefore is the relation between firing sequences and net equivalence.

Trace Sets

The term *trace sets* is used to denote a variety of models. Symbols in traces can be interpreted in different ways, ranging from voltage transitions to handshake events. There are also many different ways to define trace sets. Some people use commands, which are a generalization of regular expressions. Others base their syntax on DI-algebra (see Section 3.7), or on I-nets. The syntax that is used to define trace sets does not matter much, but subtle differences may occur in the expressive power.

Semantics based on trace sets differ considerably in expressive power. One of the simplest semantic domains is that of non-empty prefix-closed trace sets. In this domain,

safety properties can be expressed. Refusals can be added to deal properly with non-determinism. Other extensions include divergences (to deal with interference or infinite chatter), and complete and infinite traces (to deal with progress and fairness).

Tree-like

Alternative terms for tree-like techniques are state transition diagrams and state graphs. CCS [Mil80] is an example of a tree-like specification technique. CCS is not widely applicable in the design process. An example of a problem with CCS is the design of a 3-way four-phase signaling arbiter. In CCS, the 'well-known' implementation using three 2-way arbiters is not an implementation that satisfies the 3-way arbiter specification, because internal communications are retained as "tau" actions.

Other models

Other specification models that are used include partially ordered multisets and production rules.

3.4 Nonputs

Occam has no symmetric synchronization primitive. The main reason for this is to facilitate implementation on Transputer hardware. For the implementation of general symmetric synchronization, arbitration is needed. In Occam, arbitration is avoided by forcing the programmer to distinguish between input and output, thereby breaking the symmetry. The programmer thus indicates which party has the initiative in case choice is involved.

When Occam is used as a specification language for VLSI programs, specifications might have too high a degree of implementation detail. In data-communication, the distinction between input and output is important in order to express causality. Sometimes we also need a more symmetric form of pure synchronization, in which it does not make sense to break the symmetry. In that case we could call these channels nonputs.

3.5 Liveness

Fairness is rarely a correctness issue in circuit design. Circuits simply make certain types of progress. Primitive components do not deadlock. Deadlock may be introduced however by composing circuit components. Therefore, end results of designs should be analyzed or a proof should be given that deadlock will not occur.

In the asynchronous world, the main issue is avoiding interference, whereas in the synchronous world, deadlock is the main concern.

3.6 Aspects of Formal Models

- Expressiveness, including performance requirements.
- Full abstraction, refinement.
- Tool support.
- Tractability. (Suitability for (formal) manipulation.)
- Data representation.

Most formalisms that are used in asynchronous-circuit design are bad at data-abstraction. The main reason for this is that the theory in this field is still quite young, and up till now is focussed on concurrency control, which is a hard job in itself. Clearly, a smooth theory of data representation and abstraction is needed in a further formalization of the design process.

We can distinguish at least two kinds of tools, viz. tools that support transformations by a designer, and tools that verify properties of a design, for example correctness. The problem with tools for asynchronous formalisms is that tools are available for various formalisms, and support manipulations within that formalism, but these tools are not hooked up to each other in a smooth way.

With today's tools, we can verify certain design steps, but are far away from an automatic compilation from high-level specifications to silicon.

3.7 DI-Algebra

Joint work of Mark Josephs and Jan Tijmen Udding [JU91].

Example: C-element. In DI-algebra, a C-element can be specified as the least fix-point of the following equation:

$$C = a?; b?; c!; C$$

Characteristic for DI-algebra is the fact that there is only one unary operator: *prefixing*, denoted by a semicolon with a communication. For a process term P, $a?; P$ does not specify that this process will refuse inputs other than a. It specifies that it will not change into process P until input a is received; other inputs are 'buffered' so to speak.

The main difference between DI-algebra and e.g. Ebergen's commands [Ebe89] is in the way trace sets are used. In the semantics of DI-algebra, inputs are always allowed. So, $a\,a$ is in the trace set of the C-element. After $a\,a$, however, any behavior is possible. The process then changes into *chaos* (which is a constant in DI-algebra), expressing that the environment should not send two inputs in a row.

An example of a law in this algebra is that symbols of the same type commute. In the example of the C-element, this law results in:

$$a?; b?; c!; C = b?; a?; c!; C$$

On account of this property,

$$C = b?; a?; c!; C$$

defines the same C-element as the specification above.

Yet another specification of the C-element is the following

$$C = a?; [a? \rightarrow \bot \ \square \ b? \rightarrow c!; C],$$

in which $\bot$ represents chaos, that is, any possible behavior.

Values can also be included in the algebra. A process that communicates values might look like $a?x; P_x$. As an example we give the specification of an adder.

$$A = a?x; b?y; c!(x + y); A$$

In DI-algebra, combinatorial explosion because of parallelism is nicely avoided by leaving the parallelism *implicit*. Some people prefer *explicit* parallelism. When using commands, for example, for specifications, the specification of a C-element would look like

$$C = \mathbf{pref} * [(a?\ ,\ b?); c!]$$

Commands are a richer domain for specifying systems, because non-delay-insensitive elements can also be specified.

A disadvantage of the explicit parallelism in commands is illustrated by the following example. Consider $C \| F$, where

$$\begin{aligned} C &= a?; b?; c!; C \\ F &= c?; d!; e!; F \end{aligned}$$

We can specify all possible behaviors using commands, but this requires the use of at least one choice construct:

$$\mathbf{pref}(a?, b?\ ;\ *[(d!; a?), (e!; b?) \mid (e!; a?), (d!; b?)])$$

Note that the problem is that either $a?$ or $b?$ may occur as soon as either $d!$ or $e!$ has occurred. A typical trace that illustrates the problem is $a\,b\,c\,e$. After this trace either a or b is allowed, but not both. Specification of all possible behaviors is straightforward in DI-algebra. By using some laws from DI-algebra, we can easily deduce that

$$(C \| F) = a?; b?; d!; e!; (C \| F).$$

The specification of a C-element by means of I-nets can be depicted as follows.

This specification in a way corresponds to a specification with synchronization on common symbols, as for example the *weave* of two commands:

$$\mathbf{pref} * [a?; c!]\ ,\ \mathbf{pref} * [b?; c!]$$

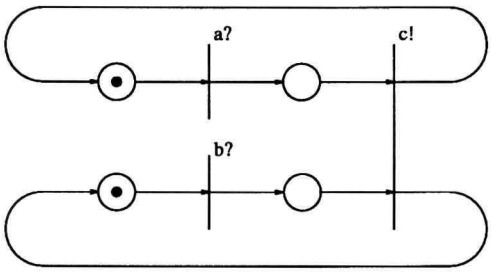

Figure 3.1: I-net specification of C-element

The C-element can also be specified by means of production rules. This specification reads as follows

$$\begin{aligned} a \wedge b &\longrightarrow c\uparrow \\ \neg a \wedge \neg b &\longrightarrow c\downarrow \end{aligned}$$

One might argue that commands are better suited for designing circuits, whereas DI-algebra seems better suited for analyzing designs. DI-algebra, however, actually suggests designs. It helps identify the important states of the system, for example. The differences in the approaches are, that (1) in the DI-algebra many behaviors are implicit but can be exposed by algebraic manipulation, whereas in commands all behaviors are explicit but we have to check for delay-insensitivity and, (2) in the DI-algebra designs can be verified by algebraic manipulation, whereas in commands one has to follow design heuristics that are known to be correct.

DI-algebra can be used for non-trivial designs. This was demonstrated in the design of Martin's D- and Q-elements, non-blocking arbiters, a routing chip [JMV91], and a delay-insensitive stack with constant response-time [JU90].

Session 4

Transformation Methods

Session chair: **Kees van Berkel**

Transformation of specifications to asynchronous circuits.
Transformation methods and heuristics.

4.1 Motivation

The subject of this session is the transformation of specifications to asynchronous circuits. The approach that is used as an example in this session is the Tangram compiler for automatic translation of specifications to asynchronous circuits, which is being developed at Philips Research Laboratories. The aim of the project is to provide a system for fast and cheap design of digital circuits.

Some challenging areas for the applications of asynchronous circuits lie in the field of consumer electronics, e.g.:

- Compact Disc, control & data processing;
- Digital Compact Cassette; both play and record, considerable data processing (10^6 transistors);
- Digital Audio Broadcast;
- Mobile Telephone;
- High Definition Television, data rate two orders of magnitude higher than audio applications.

For these applications it is not yet clear which architectures, algorithms and communication protocols are best suited, in contrast to, for example, micro-processors. Therefore the designer has more freedom, which opens the way for the application of asynchronous circuits. The two main motivations for using asynchronous circuits are:

1. ease of design, and
2. low power consumption.

Applications as listed above require fast development of new ICs with limited resources (i.e. fast and cheap). Especially when products are in the beginning of their life cycle, several generations of the product must be developed in a short time span. A silicon compiler can reduce the development time. The use of asynchronous components also makes the design of systems in this area easier, because they often contain components with different data rates.

The low power consumption of asynchronous circuits is very useful for the portable versions of the products. A current version of the portable CD-player can play only a few CDs on one set of batteries. For the DCC the power consumption is assumed to be even worse. A major part of the power is used by the ICs, so if the power consumption of the ICs can be reduced, the products will be more attractive. The need for reduction of power consumption has also led to an increasing interest in the use of analog circuits.

4.2 Architecture

One of Kees van Berkel's propositions is that in every silicon compilation approach three levels of abstraction can be distinguished, viz.:

1. VLSI programming language,
2. intermediate architecture, and
3. asynchronous circuits.

The VLSI programming language must provide primitives (communication, synchronization, control, data storage and computation), structuring mechanisms (procedures and functions), and a way to express sharing of resources. On this level the following topics are dealt with:

- functional correctness of the program (and hence of the compiled circuit);
- area, speed, and dissipation of the resulting circuit.

The intermediate architecture must provide an interface between the VLSI program and its implementation. This level must establish a separation of concerns between the two other levels.

On the circuit level the following topics are relevant:

- handshake protocol;
- data encoding;

- aggregation level of timing issues (DI/SI);
- technology (CMOS, MOS).

This level also introduces the problems of testing (diagnostic & production), adaptors (external communication, synchronous components) and initialization of the circuit.

Kees van Berkel uses *Tangram*, a language based on CSP (and Occam), as the VLSI-programming language. The intermediate architecture is represented by *handshake circuits*. For the implementation the following decisions were made:

- 4-phase handshake protocol;
- dual-rail encoding;
- delay-insensitive up to gate level and isochronic fork;
- static CMOS technology.

The language and the translation method are discussed in [vBS88] and [vBKR+91].

4.3 Transparency

An important principle in the Tangram compiler is transparency. The compilation is *transparent* in the sense that the cost/performance of the circuit corresponding to a program can easily be deduced from the program text. There is a simple relation between the cost/performance of the program and the cost/performance of the program primitives. This allows the designer to reason about the cost/performance of the result on the level of the program. The designer can experiment with variations that optimize some aspect (speed, area etc.) and adjust the program in order to meet some constraint. Current mainstream high-level synthesis lacks transparency, which results in poor control by the designer over the cost/performance of the results.

4.4 Performance of the Tangram Compiler

The Tangram compiler is intended to be used by system designers who are not VLSI-engineers but experts in their application fields. These users must be able to produce relatively efficient ICs within a limited amount of time using limited resources. This means that users are willing to accept the system if the overhead in area and speed with respect to the existing methods is not too big. The current Tangram compiler yields circuits that are up to a factor two worse than synchronous circuits in both area and speed. However, if we look at a complete chip, the degradation need not be that bad. A typical chip consists of a processing part (30-40%), an I/O part (20-30%), and memory (30-50%). (See Figure 4.1.)

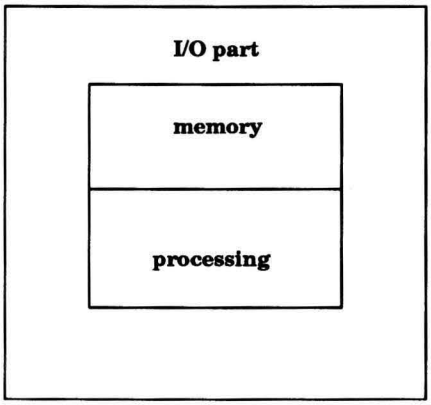

Figure 4.1: Generalized architecture of a chip

If the memory and I/O part are implemented with standard components the overall performance will be better. Once the degradation in size and speed has been reduced to 25% the Tangram compiler becomes a serious alternative for the existing methods.

It should be noted that asynchronous circuits stand at the beginning of the learning curve, in contrast to synchronous circuits with a respectable tradition of a few decades. As asynchronous circuits proceed along the learning curve, improvement of area efficiency may be expected.

4.5 Discussion

The discussion was started by the question whether sharing of resources is equivalent to asserting non-concurrency. In the Tangram compiler, sharing is indeed only allowed in the case of sequential use. Sharing can be profitable if:

- the size in hardware of the shared function is large enough,
- the same argument is used, which will reduce the required infrastructure, or
- the result is sent to the same variable (see the previous point).

In Tangram, definition of components that repeatedly compute a certain fixed function introduces seemingly unnecessary slack. Consider for example component F that repeatedly computes function f, as expressed by the following piece of Tangram text:

$$\#[a?x; b!f(x)]$$

In this case first the handshake along a is completed and then the communication along b is performed (see Figure 4.2.a). A more area-efficient solution (but not equivalent, thus

not necessarily better) would be to complete the handshake when the data transfer has been completed (see Figure 4.2.b). This could be denoted as follows:

$$\#[a?x \bullet b!f(x)]$$

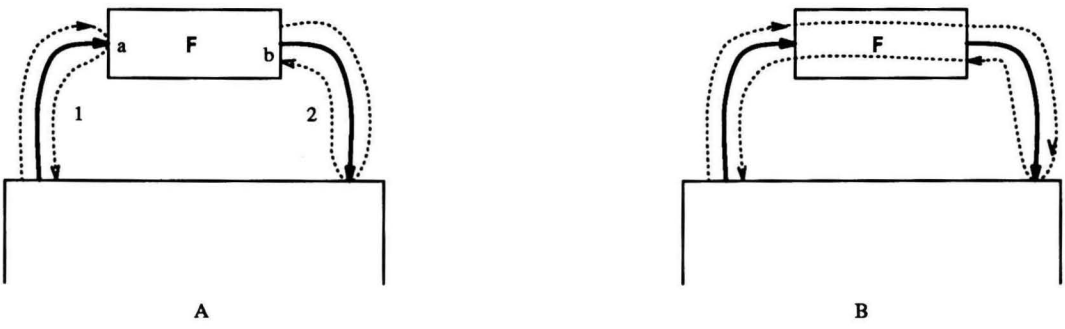

Figure 4.2: Two possible paths for completing handshake

The bullet, which denotes interwoven handshakes, would be an extension of the syntax. This resembles the reshuffling as used by Martin [Mar90]. It would, however, introduce implementation issues of the asynchronous circuit level in the language that lie outside the expressive power of the language. The problem is semantical, that is, the underlying model (CSP/trace theory) does not allow simultaneity of atomic actions.

Robert Sproull remarked that this problem does not exist when 2-phase signalling is used. His motivation for using 2-phase signalling is:

1. no meaningless transitions (economy);
2. natural for CMOS;
3. symmetry between up and down going transitions;
4. acknowledgement can be provided via the environment;
5. simple implementation of sequence of events by wiring together;
6. Ivan (Sutherland) loves it.

With 2-phase signalling, communication may be simpler than with 4-phase signalling, but in general the number of states per component is larger. Especially for data communication this can become inefficient. In the Tangram compiler both protocols could be used (2-phase for control and 4-phase for data). This, however, introduces the need for adaptors when the two protocols meet. Also the 2-phase is not always more efficient for control components. The sequencer is just a bundle of wires in 2-phase. But the mix-component becomes larger when 2-phase is used instead of 4-phase. The conclusion is that there is no simple rule for deciding between 2-phase and 4-phase handshake components. For the Tangram compiler the 4-phase handshake for all components seems to be a good decision, whereas for bundled data 2-phase may be better.

Session 5

Synchronous and Asynchronous

Session chair: **John Brzozowski**

Integration of synchronous and asynchronous design techniques, tools, and designs.

5.1 Pedagogical Tools

One of the objectives of this session is to discuss how the use of asynchronous circuits could be increased. One way to do this is through better education. In this connection it is useful to examine how asynchronous circuit theory is taught and how it should be taught, and how it relates to synchronous circuit theory.

Charles Molnar made the point that asynchronous circuits operating in fundamental mode are essentially synchronous. They do not have a clock, but it is possible to predict the sequence of states of such a circuit (if it is well behaved), because the environment waits long enough for the circuit to reach a stable state. Charles proposed the use of the term *sequenator* (cf. Section 7.5), meaning that a mechanism exists in fundamental-mode circuits that permits the states to be indexed sequentially, even though there is no clock.

It is also possible to have a somewhat different view, namely, that synchronous circuits are asynchronous circuits operated under some restrictions. John Brzozowski illustrated this point of view by some examples.

Consider two NOR gates, one with inputs S and Q and output $\overline{Q}$, and the other with inputs R and $\overline{Q}$ and output Q. The operation of the circuit constructed from two such NOR gates is governed by the following equations:

$$\begin{aligned} \overline{Q} &= (S+Q)' \\ Q &= (R+\overline{Q})' \end{aligned}$$

This circuit has three stable gate states, viz. $Q\overline{Q} = 00, 10, 01$. The table below shows the state transitions computed using the Boolean equations. The non-stable states of the circuit are labeled with an asterisk ($*$).

	SR			
$Q\overline{Q}$	00	01	10	11
01	01	01*	10	00*
10	10	01	10*	00*
00	??	01*	10*	00

When both inputs are made 1 and kept at that value for a sufficiently long time, the gates reach the stable state 00. When both inputs then change to 0, both gates become unstable, and there is a "race" between them. If the gate delays are unequal, stable state 01 or 10 may be reached. If they are equal, the circuit may oscillate in the states 00 and 11. Since it is unlikely that the gate responses remain perfectly matched for a long time, the circuit eventually reaches one of the two stable states 01 or 10, upon leaving the oscillation. In a very crude sense, this can be viewed as a representation of the metastable-state (glitch) phenomenon.

The behavior described above is normally considered undesirable since the final state after the completion of the transition cannot be uniquely predicted. To make sure that the circuit will not reach the state that causes this uncertainty, we restrict its mode of operation. Different modes of operation include:

- Unrestricted mode: unrestricted changes on inputs. This may result in unpredictable results, as above.
- Fundamental mode: the next input change may not occur until the circuit has settled after the previous input change. In a way, this mode of operation is adopted in synchronous circuit design, where the clock speed is adjusted to assure that the circuit settles properly.
- No double input changes, meaning that change of multiple inputs is not allowed. This is even more restricted than fundamental mode.
- $S = R = 1$ not allowed. This results in the behavior of the standard Set-Reset latch. Under this restriction, output $\overline{Q}$ is always equal to the negation of output Q; hence it can be removed.
- Input-output mode: an input change can occur only if the circuit has provided an appropriate output in response to a previous input change.
- According to some specification that restricts both the environment and the circuit. For example, one may use a trace structure to specify such a restriction. Input-output mode is a special case of this mode of operation.

Synchronous circuits can now be treated as special cases of asynchronous circuits. Consider for example the circuit depicted in Figure 5.1.

The non-stable states of this circuit are labeled with an asterisk (*) in the following table.

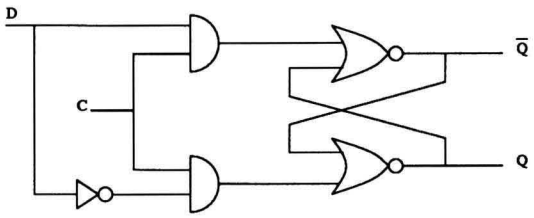

Figure 5.1: D flip-flop

	CD			
$Q\overline{Q}$	00	01	10	11
01	01	01	01	10*
10	10	10	01*	10

A safe way to operate this circuit is not to change D when $C = 1$. Clearly, C can be a clock input and can be eliminated from the description of the circuit if we adopt the above rule. Note furthermore that Q and $\overline{Q}$ are always complementary. Therefore, we might as well omit $\overline{Q}$ from the specification. We thus end up with the following diagram.

	D	
Q	0	1
0	0	1
1	0	1

Notice that in the above discussion we analyzed a clocked circuit in the same way as we have analyzed an unclocked, that is, asynchronous circuit.

5.2 Closing the Gap

One way to get designers of synchronous circuits to design asynchronous circuits is to offer them minimal change. One could try to design asynchronous circuits by applying synchronous methodologies down to the bottom of the design, and subsequently change to asynchronous methods at this lowest level. This would probably be appreciated by the engineers, since they do not want to do proofs about self-timed properties. However, some aspects of asynchronous circuit design differ significantly from synchronous circuit design.

5.3 Design

Does the classical design methodology that uses fundamental-mode flow tables and race and hazard theory still have a role to play in modern asynchronous design methodologies?

Small components can be designed using flow tables. Depending on the mode of operation we consider, very few circuits can be designed without adding delays. Small components

can be thought of as having a state, independent of whether the components are used in a synchronous or an asynchronous environment. For larger asynchronous components, however, the notion of state is very artificial, because of the high degree of concurrency. So, for larger systems, we do not think in terms of the state of the overall system, but in terms of the states of the parts (components) of the system. (See also the discussion in the next section. A number of people were rather surprised by the extent to which classical asynchronous design techniques were used in Al Davis' project.)

One can specify components by I-nets, rather than flow tables, and this type of specification has many advantages. However, in the final step, the concept of state and Boolean algebra are used in order to come up with a circuit, much as they are used in Huffman's method.

Process algebra can also be used to design small components. Starting with the specification of a C-element for example, we can derive a decomposition into a majority element and a feedback wire. While some participants liked this approach, there were others who found it rather involved.

Design of *small* components is necessary, but they do not represent the biggest challenge in asynchronous circuit design. The main problem is how to overcome the complexity of *large* components and systems.

5.4 Al Davis's Design Methodology

As an example, Al Davis discussed the design of a Post-Office Box (POB) for communication between process elements and a communication chip.

The first activity is trying to find out what has to be built. This is done by *behavioral modeling*, using Common Lisp as a specification language. There is no formal way of proving that the environment satisfies its specification, so all that can be done is to build confidence.

The second step is the transformation of the Common Lisp specification into a design expressed using I-nets. For the POB-chip, the I-net was enormous, accounting for 50 Mbyte of disk space. Since tools do not hook up nicely, there is no way to relate the I-nets to the Common Lisp specification. The I-nets can be simulated using some Meta Software Inc. tools.

At the next level, the design is expressed as a data path controlled by some finite state machines (FSMs). At this level, as well as at later levels, the hierarchy in the I-net is pretty well reflected; there may be additional hierarchy. Designing the data path is very much like synchronous design. At this level we can deal with the delay analysis. For the POB-chip, about 200 FSMs were designed, each of about 100 states.

For communication and encoding of data, various forms are used. Bundled data, dual rail, handshakes, 2-phase, 4-phase: it is all there.

The FSM descriptions and equations are analyzed using a so-called MEAT tool (Most Excellent Asynchronous Tool). Designing the FSMs is a standard problem, though hard. It amounts to using flow tables (Unger's method), applying state minimization, combinatorial

functions, and covering redundancy to prevent races. The final design consists of a set of CMOS-functions. No latches or dynamic circuit elements are used. Of course, feedbacks are needed.

There are some very good models about how fast signals and transistors are. These models are very important in the final design stages.

A modified version of Dill's verifier is used to check the CMOS schematics produced by MEAT. Dill's verifier picked out some serious errors, also in designs that had already been completed and in which the potential error had never occurred (yet).

The reason why so many FSMs have to be designed is related to performance. The larger the modules, the better the final performance, because inside the modules the design can be fine-tuned. There is, however, a practical limit on the number of states for an FSM, because of the analysis tools that are being used and the computation time involved. Larger FSMs would require Crays to do the computations.

In the design, there are two levels of concurrency. One is the organizational concurrency between the FSMs, which is very high. The concurrency inside the FSMs is very modest.

There still is a lack of linkage between the different representations of the design. To date, there is no connection between Common Lisp, I-nets, and the architecture. It should be a doable job to connect these things. On November 1, 1991, a joint project with Stanford has been started.

5.5 Software Tools

Some available tools are listed below:

Asynchronous tools For Tangram, used by Kees van Berkel, there is a compiler and a software environment to analyze Tangram programs and circuits generated with Tangram.

Furthermore, Dill's verifier is available. This tool has often proved to be very valuable in tracing possible hazards in a design.

Al Davis uses MEAT, which has ingredients for minimizing states, doing state assignments, doing logic minimization on the functions, and for generating CMOS.

Synchronous tools FSM tools, static timing verifiers, automatic test pattern generators (ATPGs).

Both Spice, Petri-net simulators, Event simulators (switch level), algebraic manipulators.

5.6 Testing

Two test problems can be distinguished, viz. testing for fabrication errors and functional testing. An important problem is testing for fabrication errors. Testing asynchronous circuits is hard, because not every wire can be controlled.

To get asynchronous circuits accepted, Automatic Test-Pattern Generators will definitely be needed. Scan-path testing, as often applied in synchronous circuits, is not directly applicable to asynchronous circuits, since there is no clock that can be paused to scan out a state.

For testing, controllability is the main problem, because of the relative delays in different paths. Observability is not a main issue, since internal malfunctioning often leads to deadlock or premature external outputs. For testing asynchronous circuits a separate test mode is likely to be needed.

Session 6

Science and Industry

Session chair: **Robert Sproull**

Interaction between science and industry
in the field of asynchronous design.

6.1 Cooperation

Probably academics wish industry would simply send money, and industry wishes that academics would solve 'real' problems. The interest in asynchronous systems is growing, both in industry and academia, as is the cooperation among groups. Notable examples are the group at Stanford working with Hewlett-Packard and the cooperation among Philips and Eindhoven University.

One of the barriers to wider industrial acceptance of asynchronous design techniques is that of credibility: engineers need to understand how some of these circuits are designed before they believe they can design them too. To date, we lack a catalogue of design and engineering experience in asynchronous designs. A collection of 'real' designs and design experience would be a very useful tool to teach and explain to others what the asynchronous world has done. It is important to build experimental systems to grow the body of experience, and to report on it carefully.

A barrier for industry to cooperate with universities is that cooperation can be time consuming. Moreover, universities generally feel no need to use the newest available technologies. From a theoretic or scientific point of view, cooperation is not always interesting, because the problems from industry lie far aside their field of interest.

6.2 Educational Problem

The promotion of asynchronous design techniques requires a serious education project. For synchronous design, Mead and Conway did a very good job by the combination of their

book and some courses in VLSI design. In that way they expanded by orders of magnitude the number of people that well understood the basic topics involved in VLSI design.

It would be nice if industry and universities would organize a course together, and if a good text book or some good survey papers would be available. A problem for industry is that good people are often taken away from their project to manage larger and harder projects.

To get the work out to a lot of people, we need some consensus on terminology. It would therefore be good to accept a few conventions. In general, conventions will be set by whoever makes the best tools for it.

An interesting idea is to have summer workshops during which people do a design, going through each step at least once. For something like this good teachers are needed. To make sure that the knowledge gained during the course will not be lost, one also has to make sure that the tools available at the workshop are available afterwards, so participants can play with them when they come back home.

6.3 Different Views

In industry the correctness of designs is almost never proven. For most designs confidence is built by simulation. Formal proofs are not really interesting for designers. Something that would help a great deal is a synthesis tool that yields designs that are in some sense correct by construction. The formal proofs may help in proving the synthesis method correct.

There is quite a large gap between designers from industry and theorists from universities. Some people are worried about scientists who never did any engineering and who even seem to be quite happy about it. The other way round can also be a problem: designers who never have seen any formal framework for what they are doing. It might therefore be a good idea to have summer schools in which people from industry are taught some formalisms. On the other hand, PhD students should get their hands dirty on real designs in industry.

6.4 Contact

In order to establish more intense contact between industry and universities, a mailing list and a joint bibliography would be nice things to have. Another thing that might help would be some good survey papers in which one does not only advertise his own work.

We should moreover accept a few common problems and solve them. This should yield designs that are elaborated in all their details. A list of these problems could, for example, start with the distributed mutual exclusion element of Alain Martin and the nacking (non-blocking) arbiter of Al Davis.

6.5 Uniform Notation

In today's theory on asynchronous circuits a broad-scope notation for different levels of abstraction is lacking. Progressive refinements cannot be expressed in a single notation. The ultimate notation should also be able to deal with peephole optimizations, which are usually far away from the source notation.

In Tangram it is possible to prove equivalence of descriptions at various levels. This demonstration is done for Tangram and its compilation scheme. In this approach, however, various formalisms and notations are used.

Some desirable properties (requirements) of a uniform notation are formulated below.

- Granularity: control of grain size. Maybe we would like to specify C-elements and maybe even below that level. Events can be interpreted in different ways, for example as voltage changes and as handshakes.
- Data representation and format.
- Sequence constraints must be expressible. Examples of sequence constraints include before, after, mutual exclusive, and non-ordered.
- Functions (input/output relations), both on modules and environment.
- Performance.

One can distinguish four levels of abstraction, viz. (i) transistor, gate, wire, (ii) speed-independent, (iii) delay-insensitive, and (iv) CSP. The basic notion at all these levels of abstraction is that of an event, which may have a different interpretation at each level of abstraction. For all levels of abstraction, traces can be used. Different levels use different rules for composition etc. One way to go from one level to another is by means of action refinement.

In some cases, theorists like to deal with fairness. Most models deal adequately with safety, some models deal adequately with deadlock, but there are no models yet that deal adequately with fairness or livelock. According to designers, fairness is a non-issue in real designs. Analysis of (absence of) deadlock, however, is considered to be quite useful.

Session 7

Concluding Remarks

Session chair: **Martin Rem**

> Turning the problems that we face into a challenging plan of action for the future.

7.1 Present and Future

In this session a short review of the previous sessions has been given, see Sections 7.2 to 7.6. Furthermore, we have tried to outline the actions and interactions for the near future, see Sections 7.7 to 7.9.

7.2 Conclusions from Session 2

Whereas there seems to be little consensus about the interpretation of the notion asynchronous, the notions speed-independent, delay-insensitive, and self-timed seem to be agreed upon. Speed-independent is said to be defined by Muller, see [Mil65], and Dill, see [Dil89]. Udding has defined delay-insensitive, see [Udd84]. Self-timed has been introduced by Seitz, see [Sei80]. We need to set up formal definitions of the notions speed-independent, delay-insensitive, self-timed, and asynchronous. These definitions have to be given in one formalism. In this way we are able to compare and relate them. Such a formalism will probably be based upon some basic notions: e.g. component, wire, and event.

In relating the notion delay-insensitive to phase diagrams, see Chapter 2, Charles Molnar suggested that for delay-insensitive circuits the exact place of the boundary for fixing an event in a phase diagram is not important. In Figure 7.1 we illustrate Molnar's conjecture.

7.3 Conclusions from Session 3

Trace theory has been developed at Eindhoven University of Technology to describe the interaction of processes, see [vdS85] and [Kal86]. The central notion in trace theory is

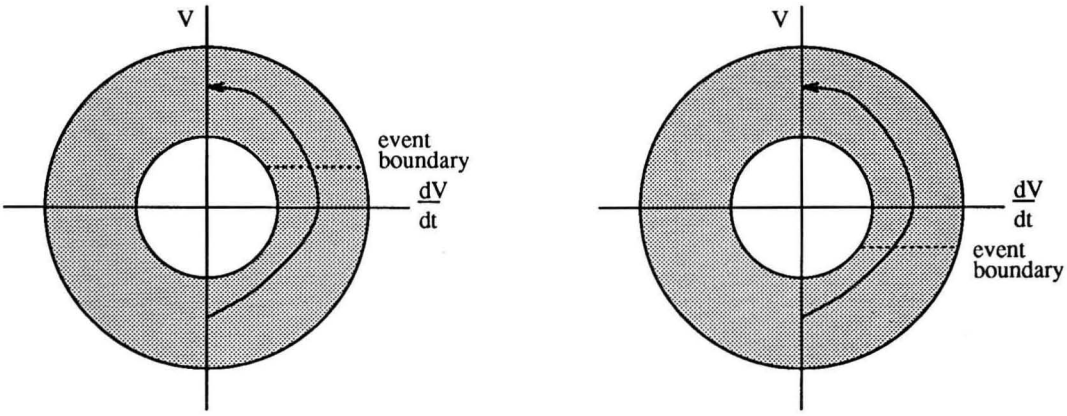

Figure 7.1: Two different boundaries in a phase diagram

the concept of a trace structure. Trace structures form a basis to which we can relate other models, e.g. CSP, trace theory programs, Petri-nets, I-nets, definitions of speed-independent and delay-insensitive. Trace structures may be part of the formalism mentioned in Section 7.2.

In order to prevent overspecification we need —in addition to the notions input and output— the notion nonput. A nonput models a symmetric synchronization between processes. The name nonput was coined by Kees van Berkel.

7.4 Conclusions from Session 4

In the development of tools for the translation of programs into circuits, transparency seems to be an important notion. It enables the designer to control at the program level general implementation aspects like area, performance, and energy dissipation. Nevertheless, the designer need not look at these aspects in all detail: he controls them, but he does not have to implement them.

When using handshake protocols we have the option to use either a 2-phase (transition-signaling) handshake protocol or a 4-phase (return-to-zero signaling) protocol. We need to study the merits of these protocols and the areas where they can be applied.

In many design strategies sub-processes are introduced to conquer the design task and enable parallelism. Sproull suggests to model the start and stop of sub-processes as two asynchronous actions.

7.5 Conclusions from Session 5

It is important to distinguish between an object (circuit) and its mode of operation. A particular circuit can be operated in many different ways, see Chapter 5. For this reason

it is very hard to design a component, when one does not make any assumptions with respect to its interaction with the environment. Therefore, one prefers to use as many assumptions about the component/environment interaction as possible. Of course, one has to be aware of the bottom line: putting too many constraints on the environment may lead to the non-existence of any interesting environment that is able to interact with the resulting circuit.

In comparing synchronous and asynchronous circuits, we noticed that in asynchronous circuits there exists a possibility to get rid of the probabilistic results that are due to races in which clock signals are involved. On the other hand, it is impossible to do an exhaustive test of asynchronous circuits, since there is no way to test for all possible delay-values – if it were even possible to control all delays at test-time.

In order to separate the sequencing aspects from the timing aspects of clocks Charles Molnar has coined the name *sequenator*. A sequenator is a mapping from a physical model to the set of integers, that preserves temporal ordering and justifies the use of sequential models to reason about the object represented by the physical model. Examples:

1. Consecutive positive transitions of a clock signal mapped onto consecutive integers;
2. Consecutive input changes (transitions) in a circuit used in a way that satisfies the fundamental mode conditions, mapped onto consecutive integers.

7.6 Conclusions from Session 6

At different design stages the designer has different concerns (different problems to solve). In order to go from one stage in a design to the next we need one notation for several stages in the design. Therefore we have to enrich the notation by creating bigger vocabularies. In this way, one deals within one notational framework with the object that is manipulated at all design stages. Transparency, cf. Chapter 4, will not be sufficient to deal with these problems.

In the design of circuits we may want to distinguish four levels of abstraction, viz.

- transistor, gate, wire,
- speed-independent,
- delay-insensitive, and
- CSP.

(See Chapter 6.)

7.7 Interaction between Research Groups

The participants of the workshop feel the need for a next workshop, to be held in 1993. From March 23 to 26, 1993, Steve Furber will organize a working conference on asynchronous logic in Manchester. This working conference will include technical papers, presentations, and open meetings.

The next workshop could be held in the fall of 1993. It may be helpful to distribute some problems to all participants to work on just before the workshop in order to be able to get into elaborate technical discussions and comparisons of methods.

The Amsterdam 1991 workshop consisted of 7 discussion sessions. We suggest that in a future workshop some (parts of) sessions are reserved for presentations. However, we strongly recommend to reserve plenty of time for discussions.

We feel the need for creating a joint public library in the area of asynchronous circuits and design. Martin Rem will create such a library at Eindhoven University of Technology. All communication concerning this library can be addressed to e-mail address

`async-bib@win.tue.nl`

David Dill already started an mailing list for the asynchronous community. The address of the mailing list is: `asynchronous@hohum.stanford.edu`. Requests for information or additions to the list can be sent to `asynchronous-request@hohum.stanford.edu`.

Al Davis (e-mail: `adavis@hplabs.hp.com`) will collect example problems that can be used to compare methods and their results. He will distribute some of these problems to the research community.

7.8 Publications

The research community interested in the design of asynchronous circuits seems not large enough (yet) to start a new journal. On the other hand, we should be able to prepare special issues of established journals, for example, by having a guest-editor for such a journal. Another suggestion is to appoint area editors for some journals, so that submission will be refereed by experts in asynchronous circuits. In this way it is possible to increase interest in asynchronous circuits and to avoid an isolated position of this field. In order to succeed we need to come to an agreement on terms and notation. Furthermore, we should publish a survey paper to present an overview of the research in this field.

7.9 Technical Support

We do not have tools available yet for all design stages. A good tool that *is* available is Dill's verifier, cf. [Dil89]; it can be used to detect flaws in both the circuit and the synthesis algorithm. Several tools are still needed:

- program to estimate area for wiring;

- static timing verifier;
- automatic test pattern generator.

Once we have a good toolkit available we may want to organize a summer school on the design of asynchronous circuits. We feel that it is very important that such a toolkit is not only used at the summer school, but that it is also readily available to the participants after they have returned to their research sites.

Bibliography

[Bro89] R. W. Brockett. Smooth dynamical systems which realize arithmetical and logical operations. In Hendrik Nijmeijer and Johannes M. Schumacher, editors, *Three Decades of Mathematical Systems Theory: A Collection of Surveys at the Occasion of the 50th Birthday of J. C. Willems*, volume 135 of *Lecture Notes in Control and Information Sciences*, pages 19–30. Springer-Verlag, 1989.

[CM74] Wesley A. Clark and Charles E. Molnar. Macromodular computer systems. In Ralph W. Stacy and Bruce D. Waxman, editors, *Computers in Biomedical Research*, volume IV, chapter 3. Academic Press, 1974.

[Dil89] David L. Dill. *Trace Theory for Automatic Hierachical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1989.

[Ebe89] Jo C. Ebergen. *Translating Programs into Delay-Insensitive Circuits*, volume 56 of *CWI Tract.* Centre for Mathemathics and Computer Science, 1989.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[JMV91] Mark B. Josephs, Rudolf H. Mak, and Tom Verhoeff. Asynchronous design of a router. In J. P. Veen, editor, *Proceedings of the IEEE/ProRISC Symposium on Circuits, Systems and Signal Processing*, pages 173–179, Utrecht, Netherlands, 1991. Stichting voor de Technische Wetenschappen.

[JU90] Mark B. Josephs and Jan Tijmen Udding. The design of a delay-insensitive stack. In G. Jones and M. Sheeran, editors, *Designing Correct Circuits*, pages 132–152. Springer-Verlag, 1990.

[JU91] Mark B. Josephs and Jan Tijmen Udding. An algebra for delay-insensitive circuits. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 3, pages 147–175. AMS-ACM, 1991.

[Kal86] Anne Kaldewaij. *A Formalism for Concurrent Processes.* PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1986.

[Mar90] Alain J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication.* Addison-Wesley, 1990. UT Year of Programming Institute on Concurrent Programming.

[Mil65] R. E. Miller. *Sequential Circuits and Machines*, volume 2 of *Switching Theory.* Wiley, 1965.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, 1980.

[Sei80] Charles L. Seitz. System timing. In Carver A. Mead and Lynn A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[Udd84] Jan Tijmen Udding. *Classification and Composition of Delay-Insensitive Circuits.* PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1984.

[vBKR+91] Kees van Berkel, Joep Kessels, Marly Roncken, Ronald Saeijs, and Frits Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proceedings of the European Design Automation Conference*, pages 384–389, 1991.

[vBS88] C. H. (Kees) van Berkel and Ronald W. J. J. Saeijs. Compilation of communicating processes into delay-insensitive circuits. In *Proc. of the 1988 IEEE Int. Conf. on Computer Design: VLSI in Computers & Processors*, pages 157–162, 1988.

[vdS85] Jan L. A. van de Snepscheut. *Trace Theory and VLSI Design*, volume 200 of *Lecture Notes in Computer Science.* Springer-Verlag, 1985.

Appendix A

List of Participants

Kees van Berkel
Philips Research Laboratories
P.O. Box 80.000
5600 JA Eindhoven
The Netherlands
berkel@prl.philips.nl

Walter Bosch
Institute for Biomedical Computing
Campus Box 8036
Washington University
School of Medicine
660 South Euclid Avenue
St. Louis MO 63110 USA
walter@wusbl.wustl.edu

Raymond T. Boute
Department of Computer
and Communications Systems
University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands
raymond@cs.kun.nl

John A. Brzozowski
Department of Computer Science
University of Waterloo
Waterloo
Ontario N2L 3G1
Canada
brzozo@watmath.waterloo.edu

Al Davis
HP Laboratories
Building 3L
P.O. Box 10490
Palo Alto
CA 94303-0969
USA
adavis@hplabs.hp.com

Jo Ebergen
Computer Science Department
University of Waterloo
Waterloo
Ontario N2L 3G1
Canada
jebergen@maytag.waterloo.edu

Ting-Pien Fang
Institute for Biomedical Computing
Campus Box 8036
Washington University
School of Medicine
660 South Euclid Avenue
St. Louis MO 63110 USA
ting@wuibc.wustl.edu

Steve Furber
Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
UK
sfurber@cs.mau.ac.uk

Ran Ginosar
Department of Electrical Engineering
Technion
Haifa 32000
Israel
ran@ee.technion.ac.il

Corry Huijs
Faculty of Computing Science
University of Twente
KGT Building H428
P.O. Box 217
7500 AE Enschede
The Netherlands
chuijs@cs.utwente.nl

Mark B. Josephs
Oxford University Computing Laboratory
Programming Research Group
11 Keble Road
Oxford OX1 3QD
UK
Mark.Josephs@comlab.ox.ac.uk

Michiel van der Korst
Department of Mathematics
and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands
wsinmvdk@win.tue.nl

Wilbert Körver
Department of Mathematics
and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

Steven Molnar
Department of Computer Science
Sitterson Hall C.B. # 3175
University of North Carolina
Chapel Hill
NC 27599-3175
USA
molnar@cs.unc.edu

Charles Molnar
Department of Mathematics
and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands
wsincem@win.tue.nl

Charles Molnar visiting from
Institute for Biomedical Computing
Campus Box 8036
Washington University
School of Medicine
660 South Euclid Avenue
St. Louis MO 63110 USA
cem@wuibc.wustl.edu

Ad Peeters
Department of Mathematics
and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands
wsinap@win.tue.nl

Martin Rem
Department of Mathematics
and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands
rem@win.tue.nl

Fred U. Rosenberger
Institute for Biomedical Computing
Campus Box 8036
Washington University
School of Medicine
660 South Euclid Avenue
St. Louis MO 63110 USA
fred@wuibc.wustl.edu

Huub Schols
Department of Mathematics
and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands
wsinhuub@win.tue.nl

Carl J. Seger
Department of Computer Science
University of British Columbia
6356 Agricultural Road
Vancouver
B.C. Canada V6T 1Z2
seger@cs.ubc.ca

Robert F. Sproull
Sun Microcystems Laboratories
2 Federal St.
Billerica
MA 01821 USA
rsproull@east.sun.com

Jan Tijmen Udding
Department of Computer Science
University of Groningen
P.O. Box 800
9700 AV Groningen
The Netherlands
jtu@cs.rug.nl

Tom Verhoeff
Department of Mathematics
and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands
wstomv@win.tue.nl

Appendix B

Asynchronous Bibliography

All participants of the workshop agreed upon the need for a joint public bibliography on asynchronous circuits in a broad sense. Immediately after the workshop, such a bibliography has been set up at Eindhoven University of Technology. In these proceedings, a first version of the bibliography is included.

We have chosen the BibTeX format for the data-base, mainly because of the widespread use of LaTeX, the corresponding document preparation system. A compressed version of the bib-file is available for anonymous ftp on Internet from `ftp.win.tue.nl` (address: `[131.155.70.100]`) as file `async.bib.Z` in directory `pub/tex`.

We have not defined precise criteria that items of the bibliography should satisfy. In our opinion, at least two requirements should be met. First, the items in the list should be related to asynchronous circuits. Second, the referenced material must be accessible, that is, obtainable from, for example, the author(s).

All communication concerning this library can be sent to the e-mail address:

`async-bib@win.tue.nl`

References

[1] Anonymous. Science and the citizen. *Scientific American*, 228:43–44, 1973.

[2] Douglas B. Armstrong, Arthur D. Friedman, and Premachandran R. Menon. Design of asynchronous circuits assuming unbounded gate delays. *IEEE Transactions on Computers*, 18(12):1110–1120, December 1969.

[3] J. C. M. Baeten. *Applications of Process Algebra.* Cambridge University Press, 1990.

[4] J. C. M. Baeten and F. W. Vaandrager. Specification and verification of a circuit in ACP (revised version). Report P8821, University of Amsterdam, Programming Research Group, October 1988.

[5] Marek A. Bednarczyk. *Categories of Asynchronous Systems.* PhD thesis, University of Sussex, October 1987.

[6] Peter Beerel and Teresa Meng. Semi-modularity and self-diagnostic asynchronous control circuits. In Carlo H. Séquin, editor, *Advanced Research in VLSI: Proceedings of the 1991 UC Santa Cruz Conference*, pages 103–117. MIT Press, 1991.

[7] J. A. Bergstra, J. W. Klop, and J. V. Tucker. Process algebra with asynchronous communication mechanisms. In G. Winskel, editor, *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 76–95. Springer-Verlag, 1985.

[8] Christian Berthet and Eduard Cerny. An algebraic model for asynchronous circuits verification. *IEEE Transactions on Computers*, 37(7):835–847, July 1988.

[9] Hans Bisseling, Henk Eemers, Michiel Kamps, and Ad Peeters. *Designing Delay-Insensitive Circuits.* IVO, Eindhoven University of Technology, September 1990.

[10] David L. Black. On the existence of delay-insensitive fair arbiters: Trace theory and its limitations. *Distributed Computing*, 1:205–225, 1986.

[11] Gregor v. Bochmann. Distributed synchronization and regularity. *Computer Networks*, 3:36–43, 1979.

[12] Gregor v. Bochmann. Delay-independent design for distributed systems. *IEEE Transactions on Software Engineering*, SE-14(8):1229–1237, August 1988.

[13] R. W. Brockett. Smooth dynamical systems which realize arithmetical and logical operations. In Hendrik Nijmeijer and Johannes M. Schumacher, editors, *Three Decades of Mathematical Systems Theory: A Collection of Surveys at the Occasion of the 50th Birthday of J. C. Willems*, volume 135 of *Lecture Notes in Control and Information Sciences*, pages 19–30. Springer-Verlag, 1989.

[14] Geoffrey M. Brown. Towards truly delay-insensitive circuit realizations of process algebras. In Geraint Jones and Mary Sheeran, editors, *Proceedings of the Workshop on Designing Correct Circuits*, pages 120–131. Springer-Verlag, 1990.

[15] E. K. Brunvand and M. Starkey. An integrated environment for the design and simulation of self timed systems. In A. Halaas and P. B. Denyer, editors, *VLSI 91*, page 4a.2, August 1991.

[16] Erik Brunvand. Parts-R-Us: A chip aparts... Technical Report CMU-CS-87-119, Carnegie Mellon University, May 1987.

[17] Erik Brunvand. A cell set for self-timed design using Actel FPGAs. Technical Report UUCS-91-013, Dept. of Comp. Science, Univ. of Utah, Salt Lake City, August 1991.

[18] Erik Brunvand. *Translating Concurrent Communicating Programs into Asynchronous Circuits.* PhD thesis, Carnegie Mellon University, 1991.

[19] Erik Brunvand and Robert F. Sproull. Translating concurrent programs into delay-insensitive circuits. In *Proceedings of ICCAD-89*, pages 262–265. IEEE Computer Society Press, November 1989.

[20] J. A. Brzozowski. Detection of timing problems in VLSI circuits. In *Congressus Numerantium, Vol. 56*, pages 7–18, 1986. Conference held in Winnipeg, Manitoba, October 1986.

[21] J. A. Brzozowski and J. C. Ebergen. Recent developments in the design of asynchronous circuits. In J. Csirik, J. Demetrovics, and F. Gécseg, editors, *Fundamentals of Computation Theory, FCT'89*, volume 380 of *Lecture Notes in Computer Science*, pages 78–94, FCT'89, Szeged, Hungary, 1989. Springer-Verlag.

[22] J. A. Brzozowski and J. C. Ebergen. On the delay-sensitivity of gate networks. Computing Science Notes 90/5, Dept. of Math. and C.S., Eindhoven Univ. of Technology, July 1990. To appear in IEEE Trans. on Computers.

[23] J. A. Brzozowski and C.-J. Seger. A characterization of ternary simulation of gate networks. *IEEE Transactions on Computers*, C-36(11):1318–1327, November 1987.

[24] J. A. Brzozowski and C.-J. Seger. A unified theory of asynchronous networks. Research Report CS-87-24, Computer Science Dept., Univ. of Waterloo, Cananda, March 1987.

[25] J. A. Brzozowski and C.-J. Seger. A unified framework for race analysis of asynchronous networks. *Journal of the ACM*, 36(1):20–45, January 1989.

[26] J. A. Brzozowski and C.-J. H. Seger. Advances in asynchronous circuit theory; part I: Gate and unbounded inertial delay models. *Bull. EATCS*, (42):198–249, October 1990.

[27] J. A. Brzozowski and C.-J. H. Seger. Advances in asynchronous circuit theory; part II: Bounded inertial delay models, MOS circuit design techniques. *Bull. EATCS*, (43):199–263, February 1991.

[28] J.A. Brzozowski and S. Singh. Definite asynchronous sequential circuits. *IEEE Trans. on Computers*, C-17(1):18–26, January 1968.

[29] J.A. Brzozowski and M. Yoeli. Practical approach to asynchronous gate networks. *Proc. IEE*, 123(6):495–498, June 1976.

[30] J.A. Brzozowski and M. Yoeli. On a ternary model of gate networks. *IEEE Trans. on Computers*, C-28(3):178–184, March 1979.

[31] Steven M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1991.

[32] Steven M. Burns and Alain J. Martin. Syntax-directed translation of concurrent programs into self-timed circuits. In J. Allen and F. Leighton, editors, *Proceedings of the Fifth MIT Conference on Advanced Research in VLSI*, pages 35–50. MIT Press, 1988.

[33] Steven M. Burns and Alain J. Martin. Synthesis of self-timed circuits by program transformation. In G. J. Milne, editor, *The Fusion of Hardware Design and Verification*, pages 99–116. Elsevier Science Publishers, 1988.

[34] Steven M. Burns and Alain J. Martin. Performance analysis and optimization of asynchronous circuits. In Carlo H. Séquin, editor, *Advanced Research in VLSI: Proceedings of the 1991 UC Santa Cruz Conference*, pages 71–86. MIT Press, 1991.

[35] J. Calvo, J. I Acha, and M. Valencia. Asynchronous modular arbiter. *IEEE Transactions on Computers*, C-37(1), January 1986.

[36] I. Catt. Time loss through gating of asynchronous logic signal pulses. *IEEE Transactions on Electronic Computers*, EC-15:108–111, February 1966.

[37] T. J. Chaney and C. E. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Transactions on Computers*, C-22(4):421–422, April 1973.

[38] Wei Chen, Jan Tijmen Udding, and Tom Verhoeff. Networks of communicating processes and their (de)-composition. In Jan L. A. van de Snepscheut, editor, *The Mathematics of Program Construction*, volume 375 of *Lecture Notes in Computer Science*, pages 174–196. Springer-Verlag, 1989.

[39] Tam-Anh Chu. On the models for designing VLSI asynchronous digital circuits. *Integration, the VLSI journal*, 4(2):99–113, June 1986.

[40] Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications.* PhD thesis, MIT Laboratory for Computer Science, June 1987.

[41] Tam-Anh Chu. Synthesis of self-timed VLSI circuits from graph-theoretic specifications. In *Proceedings ICCD '87*, pages 220–223. IEEE Computer Society Press, 1987.

[42] Y. H. Chuang. Transition logic circuits and a synthesis method. *IEEE Transactions on Computers*, C-18(2):154–168, February 1969.

[43] Wesley A. Clark. Macromodular computer systems. In *AFIPS Conference Proceedings: 1967 Spring Joint Computer Conference*, volume 30, pages 335–336, Atlantic City, NJ, 1967. Academic Press.

[44] Wesley A. Clark and Charles E. Molnar. Macromodular computer systems. In Ralph W. Stacy and Bruce D. Waxman, editors, *Computers in Biomedical Research*, volume IV, chapter 3. Academic Press, 1974.

[45] William J. Dally and Paul Song. Design of a self-timed VLSI multicomputer communication controller. In *Proceedings ICCD '87*, pages 230–234. IEEE Computer Society Press, 1987.

[46] Ilana David, Ran Ginosar, and Michael Yoeli. An efficient implementation of boolean functions and finite state machines as self-timed circuits. *ACM/Sigarch Computer Architecture News*, December 1989.

[47] Ilana David, Ran Ginosar, and Michael Yoeli. Self-timed FIFO buffer. Technical Report EE PUB No. 731, Department of Electrical Engineering, Technion, October 1989.

[48] Ilana David, Ran Ginosar, and Michael Yoeli. Self-timed reduced instruction set computer. Technical Report EE PUB No. 732, Department of Electrical Engineering, Technion, October 1989.

[49] Ilana David, Ran Ginosar, and Michael Yoeli. Self-timed is self-diagnostic. Technical Report EE PUB No. 758, Department of Electrical Engineering, Technion, November 1990.

[50] Ilana David, Ran Ginosar, and Michael Yoeli. An efficient implementation of boolean functions as self-timed circuits. *IEEE Transactions on Computers*, January 1992.

[51] Ilana David, Ran Ginosar, and Michael Yoeli. Implementing sequential machines as self-timed circuits. *IEEE Transactions on Computers*, January 1992.

[52] Mark Dean, Ted Williams, and David Dill. Efficient self-timing with level-encoded 2-phase dual-rail (LEDR). In Carlo H. Séquin, editor, *Advanced Research in VLSI: Proceedings of the 1991 UC Santa Cruz Conference*, pages 55–70. MIT Press, 1991.

[53] Peter J. Denning. The science of computing: The arbitration problem. *American Scientist*, 73:516–518, December 1985.

[54] David L. Dill. Trace theory for automatic hierarchical verification of speed-independent circuits. In Jonathan Allen and F. Thomson Leighton, editors, *Advanced Research in VLSI: Proceedings of the Fifth MIT Conference*, pages 51–65. MIT Press, 1988.

[55] David L. Dill. *Trace Theory for Automatic Hierachical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1989.

[56] David L. Dill and Edmund M. Clarke. Automatic verification of asynchronous circuits using temporal logic. In Henry Fuchs, editor, *1985 Chapel Hill Conference on VLSI*, pages 127–143. Computer Science Press, 1985.

[57] David L. Dill, Steven M. Nowick, and Robert F. Sproull. Automatic verification of speed-independent circuits with Petri net specifications. In *1989 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 212–216. IEEE Computer Society, 1989. (ICCD-89).

[58] Jo C. Ebergen. From functional specification to a delay-insensitive circuit. Technical Report CS-89-44, University of Waterloo, October 1989.

[59] Jo C. Ebergen. *Translating Programs into Delay-Insensitive Circuits*, volume 56 of *CWI Tract.* Centre for Mathemathics and Computer Science, 1989.

[60] Jo C. Ebergen. Arbiters: An exercise in specifying and decomposing asynchronously communicating components. Research Report CS-90-29, Computer Science Dept., Univ. of Waterloo, Canada, July 1990.

[61] Jo C. Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 5(3):107–119, 1991.

[62] Jo C. Ebergen. Parallel computations and delay-insensitive circuits. In Graham Birtwistle, editor, *IV Higher Order Workshop, Banff 1990*, pages 85–104. Springer-Verlag, 1991.

[63] Jo C. Ebergen and Ad M. G. Peeters. Modulo-N counters: Design and analysis of delay-insensitive circuits. In Jørgen Staunstrup and Robin Sharp, editors, *2nd Workshop on Designing Correct Circuits, Lyngby*, pages 27–46. Elsevier, North Holland, 1992.

[64] Ting-Pien Fang. An extension of Q-module realization. Technical Memorandum 317, Computer Systems Laboratory, Washington Univ., St. Louis, MO, November 1986.

[65] Ting-Pien Fang. On decomposition of delay-insensitive modules by factoring. Technical Memorandum 314, Computer Systems Laboratory, Washington Univ., St. Louis, MO, July 1986.

[66] Ting-Pien Fang and Charles E. Molnar. Synthesis of reliable speed-independent circuit modules: II. circuit and delay conditions to ensure operation free of problems from races and hazards. Technical Memorandum 298, Computer Systems Laboratory, Institute for Biomedical Computing, Washington Univ., St. Louis, MO, 1983.

[67] Edward H. Frank and Robert F. Sproull. A self-timed static RAM. In Randal Bryant, editor, *Third Caltech Conference on VLSI*, pages 275–285, 1983.

[68] Anders Gammelgaard. Implementation conditions for delay-insensitive circuits. In E. Odijk, M. Rem, and J.-C. Syre, editors, *PARLE '89: Parallel Architectures and Languages Europe*, volume 365 of *Lecture Notes in Computer Science*, pages 341–355. Springer-Verlag, 1989.

[69] Rodney M. Goodman and Anthony J. McAuley. An efficient asynchronous multiplier. In K. Bromley, S.-Y. Kung, and E. Swartzlander, editors, *Proceedings of the Second International Conference on Systolic Arrays*. Computer Society Press (IEEE), 1988.

[70] Ganesh Gopalakrishnan and Prabhat Jain. Some recent asynchronous system design methodologies. Technical Report UU-CS-TR-90-016, Dept. of C.S., Univ. of Utah, October 1990.

[71] M. R. Greenstreet, T. E. Williams, and J. Staunstrup. Self-timed iteration. In Carlo H. Séquin, editor, *VLSI '87. VLSI Design of Digital Systems*, pages 309–322. North–Holland, August 1987.

[72] Mark R. Greenstreet and Kenneth Steiglitz. Bubbles can make self-timed pipelines fast. *Journal of VLSI Signal Processing*, 2(3):139–148, November 1990.

[73] Alan B. Hayes. Stored state asynchronous sequential circuits. *IEEE Transactions on Computers*, 1981.

[74] Matthew Hennessy. Synchronous and asynchronous experiments on processes. *Inform. and Control*, 59:36–83, 1983.

[75] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[76] Lee A. Hollaar. Direct implementation of asynchronous control units. *IEEE Transactions on Computers*, C-31(12):1133–1141, December 1982.

[77] Jens U. Horstmann, Hans W. Eichel, and Robert L. Coates. Metastability behavior of CMOS ASIC flip-flops in theory and test. *IEEE Journal of Solid-State Circuits*, 24(1):146–157, February 1989.

[78] D. A. Huffman. The synthesis of sequential switching circuits. In E. F. Moore, editor, *Sequential Machines: Selected Papers*. Addison-Wesley, 1964.

[79] M. Hurtado and D. L. Elliott. Ambiguous behavior of logic bistable systems. In *Proceedings of the 13th Annual Allerton Conference on Circuit & System Theory*, pages 605–611, October 1975.

[80] Marco Hurtado. *Structure and Performance of Asymptotically Bistable Dynamical Systems*. PhD thesis, Sever Institute of Technology, Washington Univ., St. Louis, MO, 1975.

[81] G. Jacobs and R. Brodersen. Self-timed integrated circuits for digital signal processing applications. In *Proceedings of Third Workshop on VLSI Signal Processing, Monterey, California*, September 1988.

[82] Bengt Jonsson. A model and proof system for asynchronous networks. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing*, pages 49–58, 1985.

[83] Bengt Jonsson. A fully abstract trace model for dataflow networks. Research Report SICS R88016, Swedish Institute of Computer Science, November 1988.

[84] Mark B. Josephs. Receptive process theory. Computing Science Notes 90/8, Dept. of Math. and C.S., Eindhoven Univ. of Technology, September 1990.

[85] Mark B. Josephs, C. A. R. Hoare, and He Jifeng. A theory of asynchronous processes. *Journal of the ACM*, (submitted), 1989.

[86] Mark B. Josephs, Rudolf H. Mak, Jan Tijmen Udding, Tom Verhoeff, and Jelio T. Yantchev. High-level design of an asynchronous packet-routing chip. In Jørgen Staunstrup and Robin Sharp, editors, *2nd Workshop on Designing Correct Circuits, Lyngby*, pages 261–274. Elsevier, North Holland, 1992.

[87] Mark B. Josephs, Rudolf H. Mak, and Tom Verhoeff. Asynchronous design of a router. In J. P. Veen, editor, *Proceedings of the IEEE/ProRISC Symposium on Circuits, Systems and Signal Processing*, pages 173–179, Utrecht, Netherlands, 1991. Stichting voor de Technische Wetenschappen.

[88] Mark B. Josephs and Jan Tijmen Udding. Delay-insensitive circuits: An algebraic approach to their design. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 342–366. Springer-Verlag, August 1990.

[89] Mark B. Josephs and Jan Tijmen Udding. The design of a delay-insensitive stack. In G. Jones and M. Sheeran, editors, *Designing Correct Circuits*, pages 132–152. Springer-Verlag, 1990.

[90] Mark B. Josephs and Jan Tijmen Udding. An algebra for delay-insensitive circuits. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 3, pages 147–175. AMS-ACM, 1991.

[91] Anne Kaldewaij. *A Formalism for Concurrent Processes.* PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1986.

[92] Robert M. Keller. Towards a theory of universal speed-independent modules. *IEEE Transactions on Computers*, C-23(1):21–33, January 1974.

[93] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. Analysis and identification of self-timed circuits. In Jørgen Staunstrup and Robin Sharp, editors, *2nd Workshop on Designing Correct Circuits, Lyngby*, pages 275–287. Elsevier, North Holland, 1992.

[94] Lindsay Kleeman and Antonio Cantoni. Metastable behavior in digital systems. *IEEE Design & Test of Computers*, 4:4–19, December 1987.

[95] Lindsay Kleeman and Antonio Cantoni. On the unavoidability of metastable behavior in digital systems. *IEEE Transactions on Computers*, C-36(1):109–112, January 1987.

[96] Zvi Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, 1978.

[97] Shinji Kọmori, Hidehiro Takata, Toshiyuki Tamura, Fumiyasu Asai, Takio Ohno, Osamu Tomisawa, Tetsuo Yamasaki, Kenji Shima, Katsuhiko Asada, and Hiroaki Terada. An elastic pipeline mechanism by self-timed circuits. *IEEE Journal of Solid-State Circuits*, 23(1):111–117, February 1988.

[98] Shinji Komori, Hidehiro Takata, Toshiyuki Tamura, Fumiyasu Asai, Takio Ohno, Osamu Tomisawa, Tetsuo Yamasaki, Kenji Shima, Hiroaki Nishikawa, and Hiroaki Terada. A 40-MFLOPS 32-bit floating-point processor with elastic pipeline scheme. *IEEE Journal of Solid-State Circuits*, 24(5):1341–1347, October 1989.

[99] P. N. Lam and H. F. Li. Hierarchical design of delay-insensitive systems. *IEE Proceedings*, E-137(1), January 1990.

[100] Luciano Lavagno, Kurt Keutzer, and Alberto Sangiovanni-Vincentelli. Synthesis of verifiably hazard-free asynchronous control circuits. In Carlo H. Séquin, editor, *Advanced Research in VLSI: Proceedings of the 1991 UC Santa Cruz Conference*, pages 87–102. MIT Press, 1991.

[101] C. N. Liu. A state variable assignment method for asynchronous sequential switching circuits. *Journal of the ACM*, 10:209–216, 1963.

[102] S. Lubkin. Asynchronous circuits in digital computers. *Mathematical Tables and other Aids to Computation*, pages 238–241, October 1952.

[103] Yonatan Malachi and Susan S. Owicki. Temporal specifications of self-timed systems. In H. T. Kung, Bob Sproull, and Guy Steele, editors, *VLSI Systems and Computations*, pages 203–212. Computer Science Press, 1981.

[104] Leonard R. Marino. General theory of metastable operation. *IEEE Transactions on Computers*, C-30(2):107–115, February 1981.

[105] Alain J. Martin. A delay-insensitive fair arbiter. Technical Report 5193:TR:85, California Institute of Technology, 1985.

[106] Alain J. Martin. The design of a self-timed circuit for distributed mutual exclusion. In Henry Fuchs, editor, *Proceedings of the 1985 Chapel Hill Conference on VLSI*, pages 245–260. Computer Science Press, 1985.

[107] Alain J. Martin. The probe: An addition to communication primitives. *Information Processing Letters*, 20(3):125–130, 1985. Erratum: IPL 21(2):107, 1985.

[108] Alain J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.

[109] Alain J. Martin. On Seitz's arbiter. Technical Report 5212:TR:86, Caltech Computer Science, 1986.

[110] Alain J. Martin. A synthesis method for self-timed VLSI circuits. In *Proceedings ICCD '87*, pages 224–229, Rye Brook, NY, 1987. IEEE Computer Society Press.

[111] Alain J. Martin. The design of a delay-insensitive microprocessor: An example of circuit synthesis by program transformation. In M. Leeser and G. Brown, editors, *Hardware Specification, Verification and Synthesis: Mathematical Aspects*, volume 408 of *Lecture Notes in Computer Science*, pages 244–259. Springer-Verlag, 1989.

[112] Alain J. Martin. Formal program transformations for VLSI circuit synthesis. In Edsger W. Dijkstra, editor, *Formal Development of Programs and Proofs*, pages 59–80. Addison-Wesley, 1989.

[113] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, *Sixth MIT Conference on Advanced Research in VLSI*, pages 263–278. MIT Press, 1990.

[114] Alain J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*. Addison-Wesley, 1990. UT Year of Programming Institute on Concurrent Programming.

[115] Alain J. Martin. Synthesis of asynchronous VLSI circuits. In J. Straunstrup, editor, *Formal Methods for VLSI Design*, pages 237–283. North-Holland, 1990.

[116] Alain J. Martin. Synthesis of asynchronous VLSI circuits. Course Notes, VLSI 91, Edinburgh, August 1991.

[117] Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, and Pieter J. Hazewindus. The design of an asynchronous microprocessor. In Seitz [153], pages 351–373.

[118] Alain J. Martin and Pieter J. Hazewindus. Testing delay-insensitive circuits. In Carlo H. Séquin, editor, *Advanced Research in VLSI: Proceedings of the 1991 UC Santa Cruz Conference*, pages 118–132. MIT Press, 1991.

[119] Carver A. Mead and Lynn A. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[120] Teresa H.-Y. Meng. *Asynchronous Design for Digital Signal Processing Architectures.* PhD thesis, UC Berkely, 1988.

[121] Teresa H.-Y. Meng. *Synchronization Design for Digital Systems.* Kluwer Academic Publishers, 1991. Contributions by David Messerschmitt, Steven Nowick, and David Dill.

[122] Teresa H.-Y. Meng, Robert W. Brodersen, and David G. Messerschmitt. Automatic synthesis of asynchronous circuits from high-level specifications. *IEEE Trans. on CAD*, 8(11):1185–1205, November 1989.

[123] Teresa H.-Y. Meng, Robert W. Brodersen, and David G. Messerschmitt. A clock-free chip set for high-sampling rate adaptive filters. *Journal of VLSI Signal Processing*, 1(4):345–365, 1990.

[124] Teresa H.-Y. Meng, Robert W. Brodersen, and David G. Messerschmitt. Asynchronous design for programmable digital signal processors. *IEEE Transactions on Signal Processing*, 39(4):939–952, April 1991.

[125] R. E. Miller. *Combinational Circuits*, volume 1 of *Switching Theory.* Wiley, 1965.

[126] R. E. Miller. *Sequential Circuits and Machines*, volume 2 of *Switching Theory.* Wiley, 1965.

[127] Peter H. Mills and J. Dean Brock. A partial order characterization of delay-insensitive circuits. In D. A. Edwards, editor, *Proceedings of the IFIP TC-10 Conference on Design Methodologies for VLSI and Computer Architecture*, Pisa, Italy, Sept. 1988, 1989. North-Holland.

[128] Robin Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[129] David Misunas. Petri nets and speed independent design. *Communications of the ACM*, 16(8):474–481, August 1973.

[130] Charles E. Molnar. Introduction to asynchronous systems. In *Proceedings New Frontiers in Computer Science Conference*, pages 83–93, Santa Monica: Citicorp/TTI, March 1986.

[131] Charles E. Molnar and Ting-Pien Fang. Synthesis of reliable speed-independent circuit modules: I. general method for specification of module-environment interaction and derivation of a circuit realization. Technical Memorandum 297, Computer Systems Laboratory, Institute for Biomedical Computing, Washington Univ., St. Louis, MO, 1983.

[132] Charles E. Molnar, Ting-Pien Fang, and Frederick U. Rosenberger. Synthesis of delay-insensitive modules. In Henry Fuchs, editor, *1985 Chapel Hill Conference on Very Large Scale Integration*, pages 67–86. Computer Science Press, 1985.

[133] D. E. Muller. The general synthesis problem for asynchronous digital networks. In *8th Symposium on Switching and Automata Theory*, New York, 1967.

[134] David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, April 1957.

[135] C. D. Nielsen, J. Staunstrup, and S. R. Jones. Potential performance advantages of delay insensitivity. In *Proceedings of Workshop on Silicon Architectures for Neural Nets*, St. Paul-de-Vence, France, November 1990.

[136] Steven M. Nowick and David L. Dill. Practicality of state-machine verification of speed-independent circuits. In *Proceedings of ICCAD-89*, pages 266–269. IEEE, November 1989.

[137] Fredrik Orava. Verifying safety and deadlock properties of networks of asynchronously communicating processes. In Ed Brinksma, Guiseppe Scollo, and Chris A. Vissers, editors, *Protocol Specification, Testing and Verification: Proceedings of the 9th International IFIP WG 6.1 Workshop*, 1989.

[138] Ad Peeters. Decomposition of delay-insensitive circuits. Computing Science Notes 90/04, Dept. of Math. and C.S., Eindhoven Univ. of Technology, April 1990.

[139] Christian Piguet. Logic synthesis of race-free asynchronous CMOS circuits. *IEEE Journal of Solid-State Circuits*, 26(3):371–380, March 1991.

[140] David K. Probst and Hon F. Li. Abstract specification, composition and proof of correctness of delay-insensitive circuits and systems. Technical Report CS-VLSI-88-2, Dept. of C.S., Concordia Univ., Montreal, Canada, April 1988.

[141] S. Purushothaman and P. A. Subrahmanyam. An algebraic basis for specifying and reasoning about protocols for designing self timed circuits. In F. Anceau and E. J.

Aas, editors, *VLSI 83: VLSI Design of Digital Systems*, pages 133–144. Elsevier Science Publishers, August 1983.

[142] Martin Rem. Trace theory and systolic computations. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE: Parallel Architectures and Languages Europe, Vol. I*, volume 258 of *Lecture Notes in Computer Science*, pages 14–33. Springer-Verlag, 1987.

[143] Martin Rem. The nature of delay-insensitive computing. In Graham Birtwistle, editor, *IV Higher Order Workshop, Banff 1990*, pages 105–122. Springer-Verlag, 1991.

[144] Fred U. Rosenberger and Charles E. Molnar. Comments on 'metastability of CMOS latch/flip-flop'. *IEEE journal of Solid-State Circuits*, 27(1):128–130, January 1992. Reply by Robert W. Dutton pages 131–132 of same issue.

[145] Fred U. Rosenberger, Charles E. Molnar, Thomas J. Chaney, and Ting-Pien Fang. Q-modules: Internally clocked delay-insensitive modules. *IEEE Transactions on Computers*, C-37(9):1005–1018, September 1988.

[146] Huub M. J. L. Schols. A formalisation of the foam rubber wrapper principle. Master's thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1985.

[147] C.-J. Seger. On the existence of speed-independent circuits. Research Report CS-87-63, Computer Science Dept., Univ. of Waterloo, Canada, November 1987.

[148] C.-J. Seger. Models and algorithms for race analysis in asynchrnonous circuits. Research Report (PhD thesis) CS-88-22, Computer Science Dept., Univ. of Waterloo, Canada, May 1988.

[149] C-J. Seger and J.A. Brzozowski. An optimistic ternary simulation of gate races. *Theoretical Computer Science*, 61(1):49–66, October 1988.

[150] Charles L. Seitz. Self-timed VLSI systems. In Charles L. Seitz, editor, *Proceedings of the 1st Caltech Conference on Very Large Scale Integration*, pages 345–355, Pasadena, CA, January 1979. Caltech C.S. Dept.

[151] Charles L. Seitz. Ideas about arbiters. *Lambda*, 1(1, First Quarter):10–14, 1980.

[152] Charles L. Seitz. System timing. In Mead and Conway [119], chapter 7.

[153] Charles L. Seitz, editor. *Advanced Research in VLSI: Proceedings of the Decennial Caltech Conference on VLSI.* MIT Press, 1989.

[154] Scott F. Smith and Amy E. Zwarico. Provably correct synthesis of asynchronous circuits. In Jørgen Staunstrup and Robin Sharp, editors, *2nd Workshop on Designing Correct Circuits, Lyngby*, pages 237–260. Elsevier, North Holland, 1992.

[155] Jørgen Staunstrup, S. Garland, and J. Guttag. Localized verification of circuit descriptions. In *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[156] H. J. Stucki and Jr. J. R. Cox. Synchronization strategies. In Charles L. Seitz, editor, *Proceedings of the First Caltech Conference on Very Large Scale Integration*, pages 375–393, 1979.

[157] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, January 1989.

[158] Ivan E. Sutherland, Charles E. Molnar, Robert F. Sproull, and J. Craig Mudge. The trimosbus. In Charles L. Seitz, editor, *Proceedings of the First Caltech Conference on Very Large Scale Integration*, pages 395–427, 1979.

[159] Hidehiro Takata, Shinji Komori, Toshiyuki Tamura, Fumiyasu Asai, Hisakazu Satoh, Takio Ohno, Takeshi Tokuda, Hiroaki Nishikawa, and Hiroaki Terada. A 100-mega-access per second matching memory for a data-driven microprocessor. *IEEE Journal of Solid-State Circuits*, 25(1):95–99, February 1990.

[160] Jan Tijmen Udding. *Classification and Composition of Delay-Insensitive Circuits.* PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1984.

[161] Jan Tijmen Udding. A formal model for defining and classifying delay-insensitive circuits. *Distributed Computing*, 1(4):197–204, 1986.

[162] Jan Tijmen Udding and Tom Verhoeff. The mathematics of directed specifications. Technical Report WUCS-88-20, Dept. of C.S., Washington Univ., St. Louis, MO, June 1988.

[163] S. H. Unger. *Asynchronous Sequential Switching Circuits.* Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[164] Stephen H. Unger. Asynchronous sequential switching circuits with unrestricted input changes. *IEEE Transactions on Computers*, 20(12):1437–1444, December 1971.

[165] C. H. van Berkel. Beware the isochronic fork. Nat. Lab. Unclassified Report UR 003/91, Philips Research Lab., Eindhoven, The Netherlands, 1991.

[166] C. H. (Kees) van Berkel, Cees Niessen, Martin Rem, and Ronald W. J. J. Saeijs. VLSI programming and silicon compilation. In *Proceedings ICCD '88*, pages 150–166, Rye Brook, New York, 1988. IEEE Computer Society Press.

[167] Kees van Berkel, Joep Kessels, Marly Roncken, Ronald Saeijs, and Frits Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proceedings of the European Design Automation Conference*, pages 384–389, 1991.

[168] Jan L. A. van de Snepscheut. *Trace Theory and VLSI Design*, volume 200 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.

[169] Victor I. Varshavsky. Hardware support of parallel asynchronous processes. Research Reports Series A, No. 2, Digital Systems Laboratory, Helsinki Univ. of Technology, Otaniemi, Otakaari 5 A, SF-02150 ESPOO 15, Finland, September 1987.

[170] Victor I. Varshavsky, editor. *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*. Kluwer Academic, Dordrecht, The Netherlands, 1990.

[171] Harry J.M. Veendrick. The Behavior of Flip-Flops Used as Synchronizers and Prediction of Their Failure Rate. *IEEE Journal of Solid-State Circuits*, 15(2):169–176, 1980.

[172] Tom Verhoeff. Notes on delay-insensitivity. Master's thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1985.

[173] Tom Verhoeff. Delay-insensitive codes—an overview. *Distributed Computing*, 3(1):1–8, 1988.

[174] Tom Verhoeff. Characterizations of delay-insensitive communication protocols. Computing Science Notes 89/06, Dept. of Math. and C.S., Eindhoven Univ. of Technology, May 1989.

[175] Tom Verhoeff and Huub M. J. L. Schols. Delay-insensitive directed trace structures satisfy the foam rubber wrapper postulate. Computing Science Notes 85/04, Dept. of Math. and C.S., Eindhoven Univ. of Technology, August 1985.

[176] Martin Waardenburg. Composition and classification of components. Master's thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1989.

[177] Sterling R. Whitaker and Gary K. Maki. Pass-transistor asynchronous sequential circuits. *IEEE Journal of Solid-State Circuits*, 24(1):71–78, February 1989.

[178] T. E. Williams, M. Horowitz, R. L. Alverson, and T. S. Yang. A self-timed chip for division. In Paul Losleben, editor, *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*, pages 75–95. MIT Press, 1987.

[179] Ted E. Williams. *Self-Timed Rings and their Application to Division*. PhD thesis, Stanford, 1991.

[180] Sheng-Fu Wu and P.David Fisher. Automating the design of asynchronous sequential logic circuits. *IEEE Journal of Solid-State Circuits*, 26(3):364–370, March 1991.

[181] Alexandre Yakovlev. Designing self-timed systems. *VLSI Systems Design*, 6:70–90, September 1985.

[182] Alexandre V. Yakovlev. A relation-based approach to analysing semantics of asynchronous hardware specifications. Technical Report No. 286, University of Newcastle upon Tyne, November 1989.

[183] Tetsuo Yamasaki, Kenji Shima, Shinji Komori, Hidehiro Takata, Toshiyuki Tamura, Fumiyasu Asai, Takio Ohno, Osamu Tomisawa, and Hiroaki Terada. VLSI implementation of a variable-length pipeline scheme for data-driven processors. *IEEE Journal of Solid-State Circuits*, 24(4):933–937, August 1989.

[184] Michael Yoeli. Specification and verification of asynchronous circuits using marked graphs. In K. Voss, H. J. Genrich, and G. Rozenberg, editors, *Concurrency and Nets, Advances in Petri Nets*, pages 605–622. Springer-Verlag, 1987.