CAD Directions for High Performance Asynchronous Circuits

Ken Stevens[1], Shai Rotem[1], Steven M. Burns[1], Jordi Cortadella[2],
Ran Ginosar[1,3], Michael Kishinevsky[1], and Marly Roncken[1]
[1]Strategic CAD Labs, Intel Corporation, Hillsboro, OR, USA
[2]Universitat Politècnica de Catalunya, Barcelona, Spain
[3]VLSI Systems Research Center, Technion, Haifa, Israel

Abstract

This paper describes a novel methodology for high performance asynchronous design based on timed circuits and on CAD support for their synthesis using Relative Timing. This methodology was developed for a prototype iA32 instruction length decoding and steering unit called RAPPID ("Revolving Asynchronous Pentium®Processor Instruction Decoder") that was fabricated and tested successfully. Silicon results show significant advantages - in particular, performance of 2.5-4.5 instructions per nS - with manageable risks using this design technology. RAPPID achieves three times faster performance and half the latency dissipating only half the power and requiring a minor area penalty as a comparable 400MHz clocked circuit.

Relative Timing is based on user-defined and automatically extracted *relative* timing assumptions between signal transitions in a circuit and its environment. It supports the specification, synthesis, and verification of high-performance asynchronous circuits, such as pulse-mode circuits, that can be derived from an initial speed-independent specification. Relative timing presents a "middle-ground" between clocked and asynchronous circuits, and is a fertile area for CAD development. We discuss possible directions for future CAD development.

1 Introduction

Power, process variations, and increased clock frequency present formidable challenges today for high-performance VLSI design, with increasing risk in future process generations. Self-timing (or asynchronous design[1]) presents potential solutions to some of these challenges. In fact, self-timing is already used in industry in restricted forms.

Asynchronous communication utilizes handshaking to ensure functionality. Handshake protocols are orthogonal to implementation media, and have been used or shown to work in technologies ranging from relays and vacuum tubes, TTL, MOS, and RSFQ devices and multiple implementation styles. Therefore, it should be possible to implement self-timed circuits using any (or most of) future circuit or implementation technology that brings out advantages in performance.

Timing information can be used to combat the full handshake overhead in area and delay by removing redundant handshakes and associated logic. Since absolute timing information is mostly unknown until layout is complete, relative timing information in the form "event *a* occurs before event *b*" is a natural representation of timing that can be used in the design flow. This idea led us to develop Relative Timed asynchronous circuits and methods for their automatic synthesis.

Section 2 presents a brief review of RAPPID microarchitecture and results. More information can be found in [10]. Section 3 describes the Relative Timing concept and automatic logic synthesis of RT asynchronous circuits. More examples of relative timing circuits used in RAPPID are presented in [11]; more details on the theory and methods of synthesis can be found in [4, 6]. Section 5 describes the status of verification. Finally, future directions for CAD development are discussed in Section 6.

2 Motivation: RAPPID design

The RAPPID research project started in 1995 and completed in 1998. The goal of the project was to demonstrate the ability to design high-speed asynchronous circuits as a potential solution for microprocessor design if and when clocked design becomes too difficult. The RAPPID project aggressively applied asynchronous techniques based on relative timing to evaluate the risks, compared prospective advantages against a comparable commercial product – the instruction length decoding and steering logic of a 400MHz clocked design, and developed a useful methodology.

2.1 RAPPID results

The RAPPID chip was fabricated on a 0.25μ CMOS process and tested successfully. The design uses static and domino gates from a standard synchronous library, with a few cus-

[1]In this paper we do not distinguish between asynchronous and self-timing, collectively using these names for design methodologies not relying on global clock signals.

DAC 99, New Orleans, Louisiana

tom circuits, such as C-elements. The results summarized in Table 1 show significant advantages of RAPPID – in particular, performance of 2.5-4.5 instructions/nS – with manageable risks using this design technology. The results were measured using the same fabrication process, temperature and voltages for the clocked and asynchronous parts. Significant advantages are shown for throughput, latency, and power, with a slight disadvantage in terms of area. Note that some significant differences between the designs do not fully support a direct comparison, and the asynchronous part could not be used as a direct replacement for the clocked part. However, these results strongly support the argument that asynchronous circuits can be designed to be competitive with high performance clocked circuits, and may even have significant advantages.

Throughput	3×	Latency	2×
Power	2×	Area	−22%
Testability		95.9%	

Table 1: Improvement of RAPPID over 400MHz clocked circuit

2.2 RAPPID microarchitecture

RAPPID receives 16-byte wide instruction cache lines at its input, extracts the instructions, and places each instruction separately in the output buffers. As shown in Figure 1, sixteen parallel length decoders are employed, which *speculatively* compute the length as if a new instruction began at each byte position. A torus-like distributed tagging and crossbar switching circuit with 16 columns and 4 rows packs the bytes into instructions and steers them into four output buffers. These dimensions are designed to balance the average computation rates. The architecture is scalable in both the horizontal (length decoding cycle) and vertical (steering logic cycle) dimensions, hence the performance can be further increased through additional parallelism.

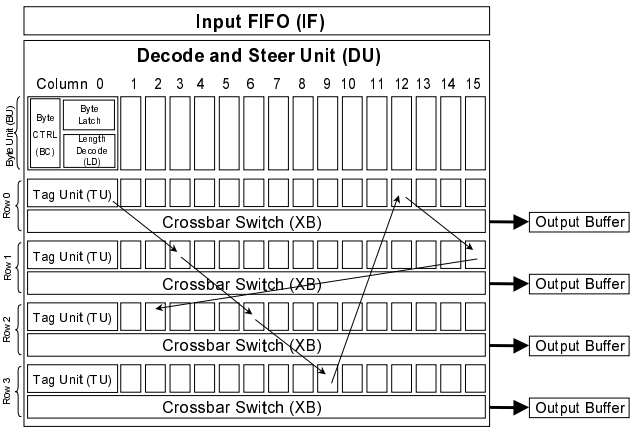

Figure 1: RAPPID Microarchitecture

RAPPID's operation consists of independent self-timed cycles. The major cycles are:

- *The length decoding and instruction ready cycle.* This cycle accepts a byte from the Input FIFO, decodes the instruction length (as all necessary bytes become available), and generates the Instruction Ready flag based on the calculated length and the Byte Ready bits from the Byte Latches of the remaining bytes in the instruction.
- *The steering logic cycle.* This cycle aligns instruction bytes from the Byte Latches and forwards them to the output buffer over the Crossbar Switch. RAPPID is a four-issue architecture.
- *Tag cycle.* This cycle directly forwards the tag to the start of the next instruction, and also synchronizes and orders the above two cycles.

Each cycle has its characteristic latency that can be independently optimized based on performance targets. The length decoding cycle is optimized for common instructions [10]. The tag cycle is optimized for common lengths. The steering logic cycle is matched to the throughput and latency of the output buffers. We can compose these cycles, using asynchronous protocols, in a scalable fashion to achieve the targeted system performance.

These three intertwined cycles demonstrate two advantages of the asynchronous solution:

- Performance is determined by the *average* rather than worst case. For example, the TAG cycle, being a central point of gathering and distributing instructions, is the performance-critical component in this architecture. It achieves the average rate of 3.6 GIPS (close to 4.5 GIPS in some of the tests), consuming on average 720M cache lines per second. Lines with fewer than five instructions (average length greater than three bytes) are consumed faster, whereas lines with shorter instructions are consumed slower.
- *Multiple frequency domains* are naturally intertwined. RAPPID combines frequency domains operating at approximately 3.6GHz, 900MHz, and 700MHz. These correspond to average frequencies of the cycles listed above.

For a more detailed presentation of RAPPID design we refer to [10, 11].

3 Relative Timing methodology

One of the goals of the RAPPID project was to study the suitability of asynchronous design styles and circuit families for aggressive high-performance circuit design. It became clear that none of the existing circuit methodologies sufficed:

- Speed-Independent (SI, a.k.a. Quasi-Delay Insensitive) design styles were not satisfactory for the critical path of the design due to area/performance overhead [2, 3, 6, 7].
- Extended Burst Mode (XBM) machines, synthesized by the 3D tool, showed improved performance due to the fundamental-mode timing assumption [14]. However, restrictions on expressing concurrency limit the types of protocols that can be implemented using this style, and further timing assumptions are not allowed.
- Timed circuits synthesis based on metric timing in the tool ATACS [8] produced circuits with improved performance, but the results were still below our expectations. Timed circuit methodology relies on absolute timing information - delay times or ranges for all devices and the environment are required for optimization. This information is largely unknown at the initial stages of the design and hence complete characterization of environment delays as well as estimation of the latencies of the circuits to be synthesized seemed awkward. Another drawback was that the state encoding problem was not solved for Timed circuits.

This brought us to the conclusion that a novel approach, based on *Relative Timing (RT)* should be used. This technique allowed us to use aggressive self-resetting and pulsed logic for avoiding the full handshake overhead [11] and produced circuit solutions superior in performance, area, power and testability. The RAPPID design used aggressive manual solutions. Later a joint effort in cooperation with Academia automated the Relative Timing design style [4]. This led to implementing automatic RT synthesis in the tool `petrify`.

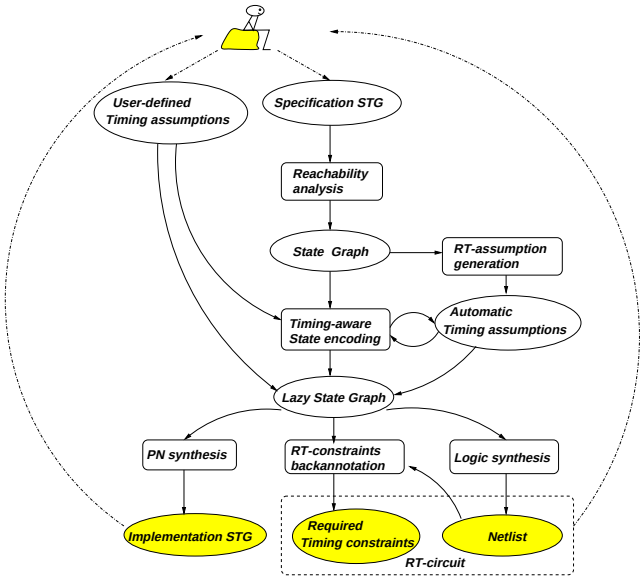

Figure 2: Design flow for synthesis of Relative timing circuits.

The design flow for synthesizing relative timing circuits in `Petrify` is shown in Figure 2. Relative timing allows the designer or CAD algorithms to specify the *effect* of delays in a circuit and its environment in terms of possible assumptions on relative ordering of events, e.g. *a* goes high before *b* goes low. The circuits are then designed using the assumptions for area and delay optimization. RT circuits can be optimized with respect to the untimed circuits for two reasons:

- RT assumptions reduce the set of reachable states and hence increase the number of don't care states for logic optimization of all signals.
- It is possible to extend the set of states in which a signal is enabled without changing the set of reachable states if other enabled signals are known to be or can be made faster than the early enabled (a.k.a. lazy) signal. This additional flexibility adds local don't cares that can differ from one signal to another.

A subset of timing assumptions used for optimization is back-annotated by the tool and become timing *constraints*. The circuits are then designed to meet the relative orderings, or verified that the restrictions are already part of the delays in the system.

Most of the timing assumptions used in RAPPID circuits can be *automatically extracted* from an initial untimed speed-independent specification. Only architectural or environmental assumptions on the inputs needed to be specified by the user for the RAPPID circuits we synthesized using `Petrify`.

4 Timing evolution in a ring

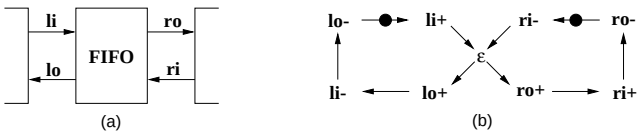

Figure 3: FIFO controller (a) and its specification with a Signal Transition graph (b). ε stands for a silent transition.

In this section we trace the development of a FIFO cell (specified in Figure 3), a simplified abstraction of a part of the RAPPID design. A speed-independent circuit implementing this specification is shown in Figure 4. Note that gates with one of the fan-ins coming vertically represent footed (unless otherwise specified) domino gates. We review a succession of progressively simpler circuits, enabled through application of relative timing.

4.1 Automatic RT assumptions

To illustrate the power of automatic extraction of relative timing assumptions let us consider the implementation of

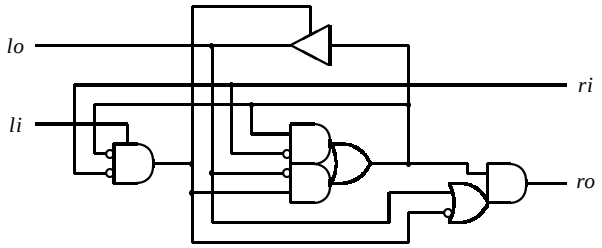

Figure 4: Speed-independent FIFO cell

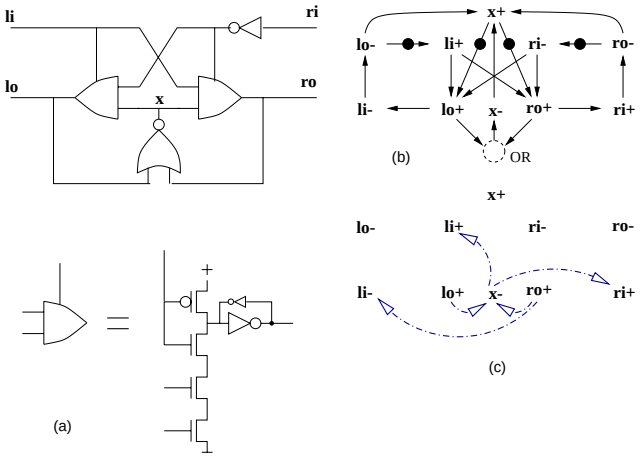

Figure 5: RT implementation of FIFO controller with fully automatic timing assumptions: (a) circuit and a schematic for a footed domino gate with a keeper, (b) STG with OR causality for $x-$, and (c) automatically generated timing constraints sufficient for correctness.

the FIFO controller (Figure 5) obtained without any user-defined timing assumptions. `Petrify` generates all necessary assumptions automatically using rules based on a simple delay model, e.g., "one gate can be made faster than two". Five timing constraints (Figure 5.(c)), sufficient for correct operation of the circuit, are generated automatically. Note that as a result of timing-aware state encoding, the state signal x is never in the critical path. Falling transitions on x are early enabled. The response time of the circuit with regard to the environment is only one domino gate delay.

The constraints "l_o+ before $x-$" and "r_o+ before $x-$" are dependent. Since the implementation of x is $x = \overline{l_o + r_o}$, it is always guaranteed that one of them will hold, whereas the other must be ensured. Since l_o+ and r_o+ are enabled simultaneously, these constraints will always hold if the delay of the domino gate plus the delay of the NOR gate is longer than the delay of the other domino gate. From the rest of constraints, the most stringent is "$x-$ before r_i+". It is required the delay of $x-$ to be shorter than the delay of r_i+ from the environment.

4.2 Assumptions about environment

RT assumptions about two input events cannot be generated automatically from the circuit specification. They must be provided by the user or obtained from the analysis of the environment.

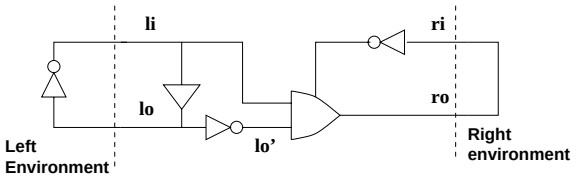

Figure 6: RT implementation of FIFO controller with one user defined assumption.

Assume that we connect the circuit of Figure 5 into a ring with a single token. The token will always arrive at an idle cell due to circuit delays if the ring is sufficiently large. Hence the handshake at the right will always complete before a new handshake at the left – hence timing assumption "$ri-$ before $li+$" is valid. It allows one to derive a more aggressive RT circuit of Figure 6. Note that an unfooted domino gate can be used in this circuit. This RT circuit requires two more constraints for correct operation adding up to three required RT constraints - one defined by the user and two automatically derived:

$$ri- \text{ before } li+ \quad \text{– user-defined}$$

$$ro+ \text{ before } lo'- \quad \text{– automatic}$$

$$lo'- \text{ before } ro- \quad \text{– automatic}$$

4.3 Pulse-mode FIFO

RT circuit of Figure 6 can be further optimized to a fully pulse-mode FIFO cell shown in Figure 7. We first include models of the left and right environment inside the circuit and then remove becoming redundant circuitry and handshake signals, lo and ri. Relative timing constraints on the pulse-mode are represented in Figure 7.(b). Arc 1 is implemented as a causal dependency, while arcs 2-4 put relative timing constraints on cooperation of the circuit and the environment [11].

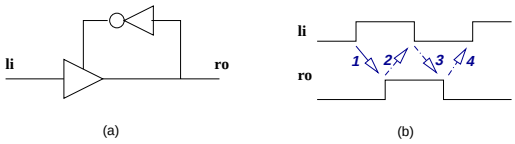

Figure 7: Relative timing pulse-mode FIFO (a) and pulse handshake protocol constraints (b)

Circuit	Worst Delay	Average Delay	Switching Energy	Area # Trans.	Stuck-at Testability
SI	2160pS	1560pS	37.6pJ	39	91%
RT-BM	1020pS	550pS	32.2pJ	40	74%
RT (Fig. 6)	595pS	390pS	18.2pJ	20	100%
Pulse	350pS	350pS	16.2pJ	17	100%

Table 2: Comparison of FIFO implementations. Energy accounts for a complete four-phase cycle. Synchronous testing in COSMOS required extra test gate for pulse circuit.

4.4 Ring summary

Some consequences of evolving a simple FIFO-like controller from a speed-independent to a pulse-mode circuit are summarized in Table 2. For a reference we included the results of the Burst-Mode implementation assuming that the Fundamental Mode assumptions are satisfied by the environment (the second row).The more aggressive timing assumptions tend to increase the performance of the circuits, reduce the area and power, and generally increase the testability. Note that the most significant improvements in performance, area and power have all been achieved by the RT transformations. The additional savings awarded by going to pulse mode are much less pronounced. Indeed, the 'aggressive' RT controller from Figure 6 may already be considered a pulse mode circuit.

5 Verification

Designing RT circuits has two aspects:

1. First the circuit is optimized through timing *assumptions* based on the architecture, environment, and circuit considerations. This is part of the synthesis flow discussed in the previous section.

2. The final circuit will have relative timing *constraints* that must hold for the circuit to operate correctly. These constraints must be shown to be valid in the implementation.

We see two solutions to timing verification of RT circuits.

1. Dense-time models can be employed where actual device delays are modeled and the circuit is verified for correct operation[1, 13].

2. The circuits are verified using unbounded delay models to extract the RT requirements. Paths are then generated and verified using simulation or separation analysis [5, 9].

The application of timed verification using unbounded verification will be briefly shown through example. Assume we have a static C-element $c = ab + ac + bc$ implemented with three AND gates and one OR gate. Errors occur when the circuit is verified against the specification using unbounded delays. Assume that the errors are due to timing faults. They can be avoided by disallowing the erroneous firing through relative timing in the verifier. The error occurs when the output from gate *ab* falls before either *ac* or *bc* rise. By placing a timing constraint that *ac* and *bc* will rise before *ab* falls, the circuit verifies correctly.

The RT requirements on the circuit generated by the verification can be turned into path constraints by finding the earliest common enabling signal. Observe the RT requirement that *bc* rises before *ab* falls. The common source for these signal transitions is $c+$. For this requirement to hold the path $c+ \rightarrow bc+$ must occur faster than $c+ \rightarrow a- \rightarrow ab-$. SPICE simulations or separation analysis can be used to guarantee that this timing requirement on the physical circuit will hold. This circuit will be valid if the delay in the environment producing the input $a-$ is slower than $bc-$.

An RT enhanced version of the verifier [12] was used to check the timing of many of the hand-designed timed circuits in RAPPID using this method.

6 Future directions

Although significant progress has been reached in automation for RT asynchronous circuits during the RAPPID project, in particular in the area of logic design, we believe that more work should be done. Some of the important directions are as follows:

- *High-level design* of asynchronous microarchitectures including the following directions:
  - High-level and RTL specification and simulation.
  - High-level transformations of the initial sequential specification of the problem to the parallel microarchitecture with speculation.
  - Relative-timing based synthesis from a high-level model. The initial specification is speed-independent and has full handshake signaling everywhere, while the implementation takes advantage of explicit and implied (automatically derived) timing assumptions. The tool should include an assistant that will suggest timing assumptions that can simplify the circuit. This item is largely covered by the current status of `petrify`, but linking to the higher level models (than Signal Transition Graphs) would be required for direct compilation from the high-level specifications.

  - Automatic handling of interrupts and preemptions based on timing assumptions. This is especially important for parallel architectures with speculation.

- *RT logic and transistor-level synthesis. Circuit optimization.*
  - Automatic propagation of relative timing constraints to sizing tools and physical design flow. This requires transforming RT constraints in the form of events into delay constraints for gates, wires and paths in the circuit.
  - A lower-level circuit optimizer that will suggest timing assumptions for circuit optimization. This tool will run on a circuit implemented at transistor level, and will be able to do transistor sizing and timing analysis. The sizing tool should know how much race margin to take. Some "optimization hints" and margin reporting features are already implemeneted in ATACS [1, 8].
  - Automatic transformations to pulse-mode circuits based on transforming interface protocols (removing interface handshake signals and substituting them with timing assumptions).
  - Timing-aware logic decomposition and technology mapping for RT circuits.

- *Verification* for relative timing asynchronous circuits, including the identification and back-annotation of the timing constraints that should be met. Some of the important steps for RT verifications were made in [8, 5, 9].

- *Testing and DFT*.
  - Have the synthesis/testing tool flag the transistors which were added to prevent hazards, which may have undetectable faults.
  - Tools for functional DFT and debug – e.g., a tool that will flag the loops that should be broken in order to freeze the circuit before the state changes. This was done manually in RAPPID. Automatic support for selecting latches that should be scanned for achieving the required level of testability is desirable.

Acknowledgments

We would like to thank the many people from Intel Corporation who provided critical help with the RAPPID design, layout, and silicon. The following people from Academia contributed to the project. Peter Beerel, Chris Myers, Kenneth Yun, Rakefet Kol, Wei-chun Chou, Peter Yeh, John Perry, Ayoob Dooply, and Rajesh Pendurkar participated in the design of RAPPID. P. Pal Chaudhury developed the BIST logic. Henrik Hulgaard participated in timing verification. Alex Kondratyev, Luciano Lavagno, Alexander Taubin, and Alex Yakovlev contributed to the developing of the relative timing synthesis.

References

[1] Wendy Belluomini, Chris J. Myers, and H. Peter Hofstee. Verification of Delayed-Reset Domino Circuits using ATACS. In *1999 International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU99)*, pages 39–44, Monterey, CA, March 1999. ACM/IEEE.

[2] Kees van Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*, volume 5 of *International Series on Parallel Computation*. Cambridge University Press, 1993.

[3] S. M. Burns. General condition for the decomposition of state holding elements. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, March 1996.

[4] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Taubin, and A. Yakovlev. Lazy transition systems: application to timing optimization of asynchronous circuits. In *Proceedings of the International Conference on Computer-Aided Design*, pages 324–331, November 1998.

[5] Henrik Hulgaard and Steven M. Burns. Bounded delay timing analysis of a class of CSP programs. *Formal Methods in System Design*, 11(3):265–294, October 1997.

[6] M. Kishinevsky, J. Cortadella, and A. Kondratyev. Asynchronous interface specification, analysis and synthesis. In *Proceedings of the Design Automation Conference*, pages 2–7, June 1998.

[7] Alain J. Martin. Synthesis of asynchronous VLSI circuits. In J. Straunstrup, editor, *Formal Methods for VLSI Design*, chapter 6, pages 237–283. North-Holland, 1990.

[8] Chris J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Dept. of Elec. Eng., Stanford University, October 1995.

[9] Radu Negulescu and Ad Peeters. Verification of speed-dependences in single-rail handshake circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 159–170, 1998.

[10] S. Rotem, K. S. Stevens, R. Ginosar, P. A. Beerel, C. J. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev. RAPPID: An asynchronous instruction length decoder. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 1999.

[11] K. S. Stevens, S. Rotem, and R. Ginosar. Relative timing. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 1999.

[12] Kenneth S. Stevens. *Practical Verification and Synthesis of Low Latency Asynchronous Systems*. PhD thesis, University of Calgary, Calgary, Alberta, September 1994.

[13] Frank C. D. Young, Kenneth S. Stevens, and Robert P. Graham. Timed Logic Conformance and its Application. In *1999 International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU99)*, pages 95–100, Monterey, CA, March 1999. ACM/IEEE.

[14] Kenneth Yi Yun. *Synthesis of Asynchronous Controllers for Heterogeneous Systems*. PhD thesis, Stanford University, August 1994.