# Implementing Asynchronous Circuits using a Conventional EDA Tool-Flow

Christos P. Sotiriou
Institute of Computer Science (ICS), Foundation for Research and Technology - Hellas (FORTH),
P.O Box 1385, Heraklion, Crete, GR 711 10 Greece.
sotiriou@ics.forth.gr

## Abstract

This paper presents an approach by which asynchronous circuits can be realised with a conventional EDA tool flow and conventional standard cell libraries. Based on a gate-level asynchronous circuit implementation technique, direct-mapping, and by identifying the delay constraints and exploiting certain EDA tool features, this paper demonstrates that a conventional EDA tool flow can be used to describe, place, route and timing-verify asynchronous circuits.

**Categories and Subject Descriptors**
B7.1. [**Integrated Circuits**]: Types and Design Styles

**General Terms**
Design, Experimentation, Standardization

**Keywords**
Asynchronous, EDA, Tool-Flow

## 1. INTRODUCTION

Asynchronous design has often been quoted as an alternative design approach which can help overcome the implementation problems of Deep-Sub-Micron (DSM), high-performance synchronous circuits, *i.e.* maintaining low clock skew, efficient clock buffering, high-power consumption, high Electromagnetic Interference (EMI) [9][3]. However, the adoption of asynchronous design has been hindered by the lack of commercial support and inability to design asynchronous systems using conventional tools. In this paper we show how a class of asynchronous circuits can be manually described, timing-constrained and timing-verified and then automatically placed and routed using conventional EDA tools. Our emphasis is on asynchronous control circuits but the same principles can be applied to asynchronous datapaths.

## 2. ASYNCHRONOUS CIRCUITS IN EDA

A variety of approaches exist for the design and implementation of asynchronous circuits [5]. Transition-based approaches specify circuits and systems in terms of signal transitions using Petri-nets [11], then using Signal Transition Graphs (STGs) [2] map the specification into an implementable Asynchronous Finite State Machine (AFSM) form. Micropipeline circuits [16] are asynchronous pipelines that may incorporate logic between their stages. Programming/compilation approaches [8][19] start from a circuit specification in a formal language, then through a sequence of syntax-transformations derive circuit descriptions. Finally, FSM-based asynchronous design approaches include asynchronous Huffman machines [18], Burst-Mode AFSMs [15][21] and Direct-mapped AFSMs [6][14]. However, only a few of these approaches are suitable for EDA. The fundamental requirement is that they use standard logic gates. An important additional requirement is that the derived circuits possess easy to determine timing constraints, which can be supported throughout the stages of an EDA flow.

Two industrial-strength asynchronous EDA tool flows actually exist, the Tangram tool set which has been used to design the 80C51 microcontroller [20] and the Theseus Logic tool flow [7]. However, the former is based on proprietory tools, whereas the latter uses proprietory standard cell libraries.

In this work we fully-automate the timing verification, placement and routing of asynchronous circuits using standard EDA tools and standard cell libraries based the direct-mapping approach introduced by L. A. Hollaar [6], which uses set-reset flip-flops for generating and storing the states of a "one-hot" encoded AFSM [18]. The one-hot encoding eliminates general races between state variables and simplifies the circuit implementation as the logic that generates the state signals assumes a regular form. One-hot AFSMs have the attractive property that their timing constraints for correct operation are independent of the actual circuit specification, in contrast to other asynchronous design approaches. In Hollaar's implementation each state corresponds to a set-reset flip-flop. The condition for entering a state is fed to the set input of the state's flip-flop, whereas the condition for leaving the state is fed to its reset input. The set and reset conditions can be implemented by combinational logic. The reset condition is in most cases the setting of a successor state [6][13].

Figure 1 shows an set-reset flip-flop AFSM implementation of a sequential portion of a state diagram. Portions of states s1, s2 and s3 are shown. State s2 is entered on the transition of signal x. The circuit operates as follows: if input x is asserted, then the output of the NAND gate to which x is input will drop as both x and s1 are high. The consequence of this is the setting of state s2 and the resetting of state s1.

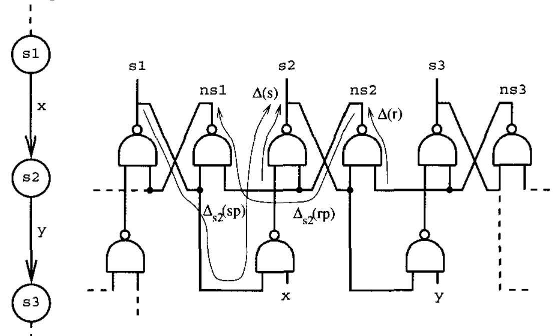

**Figure 1: One-hot Gate-Level AFSM segment**

### 2.1 One-Hot Gate-level AFSM Delay Analysis

In this section the delay assumptions for correct operation of one-hot gate-level AFSMs are identified. For a gate-level AFSM, the following delay paths may be defined:

- (a) Set input delay, *i.e.* delay between the set input of a state (the set input of the state flip-flop) and the state output, $\Delta(s)$.
- (b) Reset input delay, *i.e.*, delay between the reset input of a state (the reset input of the state flip-flop) and the inverted state output, $\Delta(r)$.
- (c) State setting delays (set paths), *i.e.* delays between the predecessor state signal outputs and the state output. This comprises the delays through combinational logic and the state flip-flop. For state $n$ with $m$ predecessors these delays can be represented as $\Delta_{n(1)}(sp)$, $\Delta_{n(2)}(sp)$, ... $\Delta_{n(m)}(sp)$ for m predecessors.
- (d) State resetting delays (reset paths), *i.e.* delays between the complementary output of a state (inverted version) and the state signal outputs of its predecessors. This comprises the delays through potentially combinational logic and the flip-flops of the predecessor states. For state $n$ with $m$ predecessors these delays can be represented as $\Delta_{n(1)}(rp)$, $\Delta_{n(2)}(rp)$, ... $\Delta_{n(m)}(rp)$ for m predecessors.

These critical delays are identified for state s2 in the simple gate-level AFSM segment of Figure 1. In order to ensure correct circuit operation one-hot critical races between two or more states must be avoided for all states of the AFSM.

### 2.1.1 One-hot race between a pair of states.

The correct order of state transitions is $10 \rightarrow 11 \rightarrow 01$. The condition for correct operation is that the next state must be properly entered before the previous state is left. In CMOS one-hot AFSMs [14] this translates to selecting appropriate transistor sizes for the n-type transistors, used to enter a state, and p-type transistors, used to leave a state.

In gate-level AFSMs, the condition for correct operation depends on the threshold points of the set-reset flip-flops, *i.e.* the point where flip-flops change state. If all the NAND gates forming the set-reset flip-flops are identical, then it is nearly impossible to make the circuit fail for a 2 state transition, as the circuit is delay insensitive, *i.e.* will work correctly for arbitrary delays of wires and gates. However, if NAND gates of different thresholds are used (different $\frac{N}{P}$ transistor sizing for example) it is possible that the previous state's flip-flop changes state, before the next state's has flipped. This type of failure is illustrated in Figure 2, where the output waveforms of an HSPICE [10] simulation of two AFSM states s1 and s2 are shown. The top and bottom panels of Figure 2 show the two outputs of state flip-flops s1 and s2, *i.e.* s1 and ns1 and s2 and ns2 respectively. Failure has been achieved by moving the threshold point of the s1 flip-flop towards VDD and loading ns2.

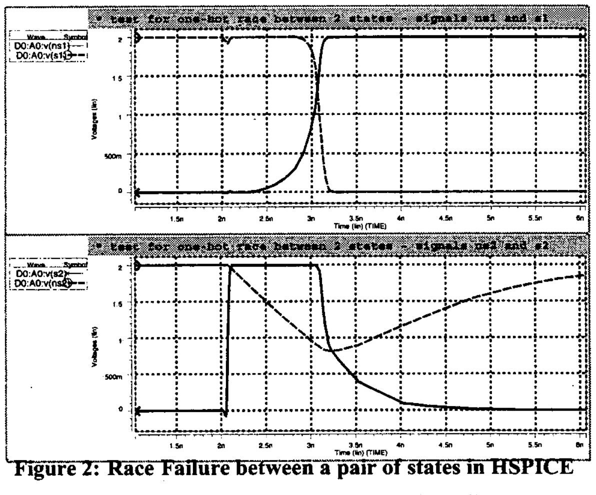

Figure 2: Race Failure between a pair of states in HSPICE

It is possible to express delay constraints which will ensure one-hot race free transitioning even if the NAND gates are non uniform. A state $n$ is entered by a set path, $\Delta_n(sp)$, whereas it is left by a corresponding reset path, $\Delta_n(rp)$. Thus, for one-hot race free operation, the delay through the reset path and through the combinational logic leading back to the set input of the state must be longer than the delay of setting the state once the set input is being driven (for all possible set and reset paths), *i.e.*:

$$\Delta_n(rp) + (\Delta_n(sp) - \Delta(s)) > \Delta(s) \Rightarrow$$
$$\Delta_n(rp) + \Delta_n(sp) > 2\Delta(s) \quad \text{(EQ1)}$$

To ensure that this delay assumption holds it is possible to bias the $\Delta(s)$ and $\Delta(r)$ delays so that $\Delta(s) < \Delta(r)$ holds.

### 2.1.2 One-hot race between three or more states.

In the case of three states different transition orders are possible: $100 \rightarrow 110 \rightarrow 010 \rightarrow 011 \rightarrow 001$ or $100 \rightarrow 110 \rightarrow 111 \rightarrow 011 \rightarrow 001$. In any case, state 101 should not be reached. The prohibited 101 state can only be reached if the middle of the three states has been set and reset before the left state has been reset. This can only occur if the right state is set (thus resetting the middle state) and the middle state reset faster than the middle state can reset the state on the left, *i.e.* the situation to avoid is: $\Delta$(sp of right state) + $\Delta$(rp of right state) < $\Delta$(rp of middle state). For two states, $n$-1 and $n$, the opposite condition must hold for correct operation (for all possible set and reset paths), *i.e.*:

$$\Delta_n(sp) + \Delta_n(rp) > \Delta_{n-1}(rp) \quad \text{(EQ2)}$$

Due to the nature of one-hot coding, ensuring that one-hot critical races do not occur for more than three states can be achieved by ensuring that two and three state races do not occur for every possible sequence of two and three states of the AFSM.

To summarise, correct circuit operation for a one-hot gate-level AFSM can be guaranteed if equations (EQ1) and (EQ2) above hold for all states of the AFSM. In the next sections we identify the features of the EDA tools that can be used to implement asynchronous circuits.

## 2.2 Asynchronous Circuit Specifications

In order to describe a gate-level asynchronous circuit into a synthesis tool, a circuit netlist form must be used, since synthesis is not (as yet) possible. One such form is the Synopsys GTECH [17] (Generic TECHnology) format, which allows for a circuit to be described in terms of generic technology-independent unsized gates. It is then possible to both map the GTECH specification into any technology and to specify path constraints which will appropriately size the technology mapped gates.

## 2.3 Delay Constraints and Hierarchy

In asynchronous circuit design, only true path delays are relevant to the circuit operation. For example, the delay constraints described in the previous section may be verified by measuring the true path delays for the corresponding paths. However, it is often the case that static timing analysis reports a path delay inappropriately, *i.e.* by combining arrival times of signals which will not change. In addition, timing analysis is particularly confused by the feedback of state registers. Careful hierarchical definition of the circuit can overcome this problem. Paths which are to be measured and constrained by the synthesis tool should not contain feedback. For example, the set and reset NAND gates may need to be constrained so that $\Delta(s) < \Delta(r)$ holds. If a cell is described so as to make feedback external to it, it is then possible for delay constraints to be fulfilled appropriately by the synthesis tool.

## 2.4 Constraint Specification

The delay constraints of direct-mapped AFSMs presented in Section 2.1 are relative, *i.e.* the delays of critical circuit paths are interrelated. There is currently little support for relative constraint specification in contemporary EDA tools, even though standards such as SDF (Standard Delay Format) v3.0 support them [12]. However, it is possible to specify absolute timing constraints on circuit paths through the commonly used GCF (General Constraint Format) timing constraints [1]. Experiments performed with Cadence's Pearl timing analyser demonstrated that timing analysis for GCF `path_delay` constraints considers the true paths between the specified points. GCF constraints are also compatible with Cadence's P&R (Placement and Routing) tools. However, as GCF constraints are designed to be imposed only on external, top-level pins, it was found necessary to introduce dummy pins to a physical design file (DEF format) to be able to constrain the appropriate paths.

## 2.5 Physical Design

Hierarchical P&R allows for timing constraints to be localised to the individual circuits and makes the process of constraining circuits manageable. The breakdown of a system into constituent parts is a very natural process for asynchronous systems, as these are generally composed of asynchronous blocks which communicate with each other via delay-insensitive asynchronous handshaking protocols. Therefore, it is straightforward to draw block boundaries at the handshake interfaces. In this way, constraints are localised, and at the top (block) level, where blocks communicate, it must be ensured that the handshake data are "bundled", *i.e.* synchronised, to their corresponding handshake control signals.

## 3. AN EXAMPLE DESIGN

We now demonstrate the process of realising a fully-asynchronous circuit using a conventional EDA flow by manually constraining an example circuit: an asynchronous, 5-stage, 32-bit pipeline, constructed out of fully-decoupled four-phase latch controllers [4][13]. The circuit was mapped to the VST-UMC 0.18$\mu$m technology.

## 3.1 Constraint Specification

Table 1 shows the delay analysis for the critical paths of the unconstrained version of the circuit. The timings of Table 1 are obtained by mapping the GTECH circuit specification to the technology library and then measuring the critical path delays. The numbers in bold represent the path delays reported by the Pearl `FindPathsBetween` command (with false paths blocked) on the critical paths, whereas the numbers in italics represent the path delays reported by the DC `report_timing` command. The DC timings are obtained by measuring the path delays at appropriate levels of hierarchy so that false paths are not considered. As Pearl allows for false paths to be blocked at the top level, the Pearl timings are more accurate as they include input and output loading.

| *State* | Δ(s) | Δ(r) | Δ(sp) | Δ(rp) |
|---|---|---|---|---|
| ackin | **0.11**/*0.06* | **0.04**/*0.03* | **0.25**/*0.17* | **0.18**/*0.15* |
| waita | **0.07**/*0.06* | **0.04**/*0.03* | **0.19**/*0.16* | **0.14**/*0.15* |
| reqout | **0.11**/*0.06* | **0.04**/*0.03* | **0.24**/*0.16* | **0.18**/*0.15* |
| waitr | **0.04**/*0.03* | **0.06**/*0.05* | **0.18**/*0.15* | **0.15**/*0.15* |
| idle | **0.10**/*0.06* | **0.04**/*0.03* | **0.26**/*0.20* | **0.15**/*0.15* |

**Table 1: Unconstrained Latch Controller Delay Analysis (in ns)**

Table 2 shows the delay analysis for the constrained circuit. This version of the circuit was technology mapped and then constrained so that the the delay of the reset gate of the state flip-flops was at least twice that of the set gate, *i.e.* $\Delta(r) \geq 2\Delta(s)$, and the bundled-data assumption on the output request holds, *i.e* all data outputs are synchronised to signal `reqout`. Imposing the $\Delta(r) \geq 2\Delta(s)$ synthesis constraint to all states ensures that after circuit synthesis the delay constraints necessary for correct circuit operation of the one-hot AFSM, *i.e.* $\Delta_n(sp) + \Delta_n(rp) > 2\Delta(s)$ and $\Delta_n(sp) + \Delta_n(rp) > \Delta_{n-1}(rp)$, will be fulfilled for all states.

| *State* | Δ(s) | Δ(r) | Δ(sp) | Δ(rp) |
|---|---|---|---|---|
| ackin | **0.06**/*0.03* | **0.09**/*0.09* | **0.20**/*0.15* | **0.23**/*0.18* |
| waita | **0.04**/*0.03* | **0.11**/*0.09* | **0.17**/*0.14* | **0.20**/*0.18* |
| reqout | **0.06**/*0.03* | **0.09**/*0.09* | **0.20**/*0.14* | **0.24**/*0.19* |
| waitr | **0.04**/*0.03* | **0.10**/*0.09* | **0.17**/*0.14* | **0.20**/*0.23* |
| idle | **0.06**/*0.03* | **0.13**/*0.09* | **0.22**/*0.20* | **0.23**/*0.18* |

**Table 2: Constrained Latch Controller Delay Analysis (in ns)**

By using the data of Table 2 the conditions for correct circuit operation can be calculated. In this way, constraints can be specified which can propagate throughout the physical part of the EDA flow. The values of the critical delays are shown in Table 3. As can be seen by this table, $\Delta_n(sp) + \Delta_n(rp) > \Delta_{n-1}(rp)$ is a tighter constraint to $\Delta_n(sp) + \Delta_n(rp) > 2\Delta(s)$.

| *State* | $\Delta_{(n)}(sp) + \Delta_{(n)}(rp)$ | 2Δ(s) | $\Delta_{(n-1)}(rp)$ |
|---|---|---|---|
| ackin | **0.43** | **0.12** | **0.23** |
| waita | **0.37** | **0.08** | **0.23** |
| reqout | **0.44** | **0.12** | **0.23** |
| waitr | **0.37** | **0.08** | **0.24** |
| idle | **0.45** | **0.12** | **0.20** |

**Table 3: Constrained Latch Controller Critical Delays (in ns)**

From the figures of Table 3 an absolute upper delay bound can be obtained for the reset path of each state, *i.e.* Δ(rp). This value is equal to the sum of the set and reset paths of the next state. There is no upper bound on the set path, *i.e.* Δ(sp), but there is a lower bound for both the set and reset paths due to the same delay constraint equation. Table 4 shows appropriate minimum and maximum absolute delay constraints on the set and reset paths. These delay constraints can be specified in GCF file format, for them to be used in the physical parts of the EDA flow, by using the GCF `path_delay` construct.

| *State* | Δ(sp) range | Δ(rp) range |
|---|---|---|
| ackin | > **0.20** | > **0.23**, < **0.37** |
| waita | > **0.17** | > **0.20**, < **0.45** |
| reqout | > **0.20** | > **0.24**, < **0.37** |
| waitr | > **0.17** | > **0.20**, < **0.45** |
| idle | > **0.22** | > **0.23**, < **0.44** |

**Table 4: AFSM Delay Constraints (in ns)**

When the circuit enters the physical design phase of the EDA flow and is P&R, the path delays will change. As the delay constraints are expressed as absolute numbers and not relatively, it is likely that although the absolute timings may have changed, the relative timings are still within the appropriate ranges and absolute timing violations reported by the tools are not affecting the circuit's operation. If contemporary EDA tool flows did support relative timing constraints that would be ideal for describing the timing constraints of asynchronous circuits. Then, it would be possible to use relative timing constraint specifications to ensure that asynchronous circuits operate correctly and absolute timing constraints for paths that impact on the circuit's performance.

## 3.2 Physical Design

Cadence's DesignPlanner was used for hierarchical design, Silicon Ensemble for block-level P&R and IC Craftsman for the top-level routing. The top-level floorplan and final layout are shown in Figure 3. The five stages correspond to the same synthesis-constrained latch controller circuit (as described in the previous section), only with different aspect ratios. After floorplanning, each stage was P&R separately in both the time-driven (with GCF constraints) and in the non time-driven (connection-driven) mode of the P&R tools.

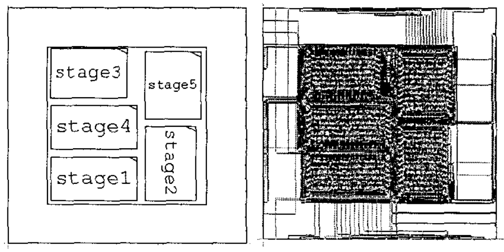

**Figure 3: Asynchronous Pipeline Floorplan and Layout**

Even in time-driven mode, small violations, less than 50ps, of the path constraints manifested. Figure 4 graphically represents the violations reported by Pearl after P&R for the non time-driven and time-driven versions of the 32-bit fully-decoupled latch controller circuits respectively. The violations shown are minimum path violations, *i.e.* the paths were faster than specified.

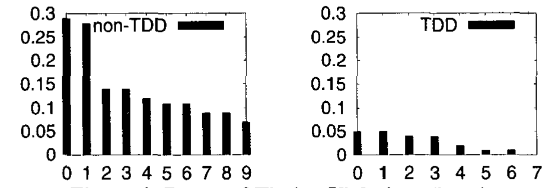

**Figure 4: Range of Timing Violations (in ns)**

Table 5 shows the timings of the post-layout critical paths of the time-driven design. As can be seen by these timings the circuit will operate correctly even though the timing analyser reports absolute timing violations. Contrasting this table with Table 3, which contains the critical post-synthesis delays, we can see that the defined absolute delays have controlled the placement and routing process effectively and have produced a circuit with delays very close to the specification.

| *State* | $\Delta_{(n)}$(sp) + $\Delta_{(n)}$(rp) | 2$\Delta$(s) | $\Delta_{(n-1)}$(rp) |
|---|---|---|---|
| ackin | **0.42** | **0.12** | **0.18** |
| waita | **0.31** | **0.08** | **0.23** |
| reqout | **0.56** | **0.12** | **0.18** |
| waitr | **0.40** | **0.08** | **0.30** |
| idle | **0.39** | **0.12** | **0.17** |

**Table 5: Time-driven Post-Layout Path Delays (in ns)**

| *State* | $\Delta_{(n)}$(sp) + $\Delta_{(n)}$(rp) | 2$\Delta$(s) | $\Delta_{(n-1)}$(rp) |
|---|---|---|---|
| ackin | **0.22** | **0.04** | **0.09** |
| waita | **0.19** | **0.04** | **0.11** |
| reqout | **0.25** | **0.10** | **0.09** |
| waitr | **0.23** | **0.04** | **0.14** |
| idle | **0.22** | **0.04** | **0.09** |

**Table 6: Non Time-driven Post-Layout Path Delays (in ns)**

Table 6 shows the timings of the post-layout critical paths of the non time-driven design. These timings demonstrate two important points. Firstly, they vary greatly from the post-synthesis timings (which estimate placement), indicating that the synthesis timings were conservative and affected the time-driven placement of the circuit negatively. The connection-driven algorithms of non time-driven mode have produced a faster circuit by minimising path lengths and packing the circuit. Secondly, despite the fact that the delays and ratios of the delays have changed, the relative path constraints still hold. The sizing of the set and reset gates of the state flip-flops, *i.e.* synthesis constraint $\Delta(r) \geq 2\Delta(s)$, and the bundled-data synthesis constraints make it hard for the placement and routing tool to violate the relative constraints for such a small circuit.

## 4. CONCLUSIONS

We have shown that it is indeed possible to exploit features of conventional synchronous EDA flows to implement asynchronous circuits. It is relatively straightforward to express the relative timing constraints that a gate-level direct-mapped control circuit must fulfill for correct operation. Thus, it is possible to perform timing verification of the asynchronous circuit's path constraints at various stages of the EDA flow. Further on, by exploiting the support for timing-driven design that EDA tools provide, it is possible to perform time-driven P&R, hence allowing for the asynchronous circuit's critical path delays to stay close to the specification during the physical design stages. However, as relative timing constraints are not as yet supported by commercial EDA flows, the timing constraints of an asynchronous circuit must be described as absolute constraints particular to a specific implementation technology. Such constraints are effectively stricter than the equivalent relative constraints which they represent and limit circuit performance.

## 5. REFERENCES

[1] Cadence Design Systems. *General Constraint Format Reference Manual*, June 1998.

[2] T. A. Chu. Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications. Technical Report MIT/LCS/TR-393, Massachusetts Institute of Technology, June 1987.

[3] M. J. Flynn, P. Hung, and K. W. Rudd. Deep-Submicron Microprocessor Design Issues. *IEEE Micro*, 19, No. 4:11–22, 1999.

[4] S. B. Furber and P. Day. Four-Phase Micropipeline Latch Control Circuits. *IEEE Transactions on VLSI Systems*, 4(2):247–253, June 1996.

[5] S. Hauck. Asynchronous Design Methodologies: An Overview. Technical Report TR 93-05-07, Department of Computer Science and Engineering, University of Washington, Seattle, 1993.

[6] L. A. Hollaar. Direct Implementation of Asynchronous Control Units. *IEEE Transactions on Computers, Vol. C-31, No. 12*, December 1982.

[7] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 114–125. IEEE Computer Society Press, Apr. 2000.

[8] A. J. Martin. Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits. In C. Hoare, editor, *Developments in Concurrency and Communication*, California Institute of Technology, 1990. Addison Wesley.

[9] D. Matzke. Will Physical Scalability Sabotage Performance Gains ? *IEEE Computer*, Sept. 1997.

[10] Meta-Software. *HSPICE User's Manual H9001*, 1990.

[11] T. Murata. Petri-Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77-4, pages 541–580, 1989.

[12] Open Verilog International. *Standard Delay Format Specification Version 3.0*, 1995.

[13] C. P. Sotiriou. *Design of an Asynchronous Processor.* PhD thesis, Institute for Computing Systems Architecture, Division of Informatics, University of Edinburgh, 2001.

[14] C. P. Sotiriou. Direct-Mapped Asynchronous Finite-State Machines in CMOS Technology. In *Proceedings of the 14th International ASIC/SOC Conference*, September 2001.

[15] K. S. Stevens. *Practical Verification and Synthesis of Low Latency Asynchronous Systems*. PhD thesis, Department of Computer Science, The University of Calgary, September 1994.

[16] I. E. Sutherland. Micropipelines. *Communications of the ACM, Volume 32, Number 6*, 1989.

[17] Synopsys. *Designware - GTECH Library*, 2000.

[18] S. H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley Interscience, Department of Electrical Engineering, Columbia University, 1969.

[19] K. van Berkel. *Handshake Circuits: An asynchronous architecture for VLSI programming*. Cambridge University Press, 1993.

[20] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An Asynchronous low-power 80C51 Microcontroller. In *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Apr. 1998.

[21] K. Y. Yun, D. L. Dill, and S. M. Nowick. Synthesis of 3D Asynchronous State Machines. In *Proc. International Conf. Computer Design (ICCD)*, Cambridge, Massachusets, 1992.