7.1

Practical Advances in Asynchronous Design and in Asynchronous/Synchronous Interfaces

Erik Brunvand
Dept. of Computer Science
University of Utah
SLC, Utah 84112

Steven Nowick
Dept. of Computer Science
Columbia University
New York, NY 10027

Kenneth Yun
Department of ECE
University of California
San Diego, CA 92093

Abstract

Asynchronous systems are being viewed as an increasingly viable alternative to purely synchronous systems. This paper gives an overview of the current state of the art in practical asynchronous circuit and system design in four areas: controllers, datapaths, processors, and the design of asynchronous/synchronous interfaces.

1 Asynchronous Control

Classical asynchronous controllers were typically implemented as Huffman machines [67]. These machines do not use clocked latches or flip-flops: the state is simply stored on feedback loops. Typically a fundamental mode assumption is required, to insure correct operation: once an input change occurs, no new inputs may arrive until the machine has stabilized. Much of the basic theory on asynchronous state machines was developed by Huffman, Unger, and McCluskey (see [67]).

Hazards, or the potential for glitches, are an important consideration in any asynchronous design [67]. In synchronous systems, the global clock usually filters out the effect of glitches. In asynchronous systems, there is no global clock, so any glitch may be interpreted as a valid signal change, and cause a malfunction. A number of techniques to eliminate combinational hazards, as well as *critical races* and *essential hazards*, have been proposed [67, 22, 5].

While this early work laid the foundations of asynchronous controller synthesis, the design methods had major limitations: (i) lack of ability to handle highly-concurrent environments; (ii) poor performance; (iii) problems in hazard elimination (in some methods); and (iv) lack of CAD optimization algorithms and tools.

Since the early and mid 1980's, several controller synthesis methods were developed, to address these limitations. These methods fall into three general categories: (i) state machines; (ii) Petri-net and graph-based methods; and (iii) translation methods.

DAC 99, New Orleans, Louisiana

1.1 Asynchronous State Machines

Much of the recent work on asynchronous state machine design is centered on *burst-mode machines* [50, 52].

Burst-mode specifications grew out of earlier informal specifications by Davis et al. [18, 17]. Davis proposed machines which would wait for a collection of input changes (an "input burst"), and then respond with a collection of output changes (an "output burst"). The key contribution is that, unlike classical asynchronous machines, inputs within a burst could be uncorrelated: arriving in any order and at any time. Therefore, these machines could operate more flexibly in a concurrent environment. Unfortunately, their synthesis methods did not insure hazard-free designs.

Nowick and Dill [52, 50] modified and formalized these specifications into the final form called *burst-mode* (BM) [52, 50]. They also proposed a new self-synchronized design style called a locally-clocked state machine, which was the first burst-mode synthesis method to guarantee a hazard-free implementation [52, 50]. The method has been applied to large-scale designs such as a cache controller [51]. They also developed the first exact hazard-free 2-level logic minimization algorithm [53].

Yun and Dill proposed an alternative implementation method, called *3D* [79, 83]. The specifications were also generalized into into *extended burst-mode* (XBM), to allow greater concurrency and practicality [80, 82]. XBM specifications can be used to to synthesize controllers for synchronous/asynchronous interfaces, where the global clock is treated as one of the controller's inputs.

A number of optimization algorithms and CAD tools have been developed, for sequential and combinational logic synthesis [23, 53, 66, 36, 39], technology mapping [61], timing analysis [10], and synthesis for testability [54]. Burst-mode CAD tools have been applied to several industrial designs, including an experimental routing chip [17] and low-power infrared communications chip [40] at HP Laboratories, an experimental SCSI controller at AMD [81], and a high-performance experimental instruction-length decoder at Intel [60].

1.2 Petri-Net and Graph-Based Methods

Petri nets and state graphs can also be used to specify asynchronous circuits. A Petri net is a directed bipartite graph which can describe both concurrency and choice. The net consists of two kinds of vertices: places and transitions. An assignment of tokens to the places is called a marking, which captures the state of the concurrent system.

Several synthesis methods use restricted Petri nets, called *marked graphs*, which model concurrency, but not choice. More general Petri nets called *Signal Transition Graph* (STG), as well as *state graphs* which specify interleaved conncurrency, are now commonly used [12, 46, 72, 2, 35].

A number of synthesis algorithms have been developed, for state minimization and assignment [37, 16] and hazard-free logic decomposition [8] (see also [72, 2, 35]). Full-scale CAD packages are now available, including one incorporated into the Berkeley *SIS* package [38], as well as *Petrify* [16]. Another synthesis method, called ATACS, focuses on *timed circuits* [48].

1.3 Translation Methods

Translation methods specify an asynchronous system using a high-level concurrent programming language. Common languages include variants of Hoare's *CSP*, *occam* and *trace theory*. The program is then transformed, stepwise, into a low-level program which maps directly onto a circuit. These methods can be used to synthesize both datapath and control. A few methods use formal algebraic derivations [21, 33]. More commonly, though, compiler-oriented techniques are used.

At Caltech, Martin et al. [42] specify and asynchronous system using a CSP-like parallel language, augmented with sequential constructs based on Dijsktra's guarded commands. The specification describes a set of concurrent processes which communicate on channels. The specification is then automatically compiled into a collection of gates and components which communicate on wires [9]. An alternative approach was developed by Brunvand and Sproull based on occam specifications [7].

At Philips Research and Eindhoven University, van Berkel et al. [68, 69] have developed an industrial synthesis package, based on their *Tangram* language. The tool has been applied to both commercial and experimental designs, including a DCC error corrector and an 80C51 microcontroller (discussed in Section 3).

2 Datapath

This section describes some of the recent advances in self-timed datapath design, concentrating on performance issues only.

A datapath can be classified as pipelined or non-pipelined. There has been a tremendous amount in asynchronous pipelines, starting with the classical *micropipeline* work by Sutherland [63]. Pipeline control can be implemented using either a two-phase protocol [24, 76, 1] or a four-phase protocol [19, 28, 25, 27].

All of the asynchronous datapath designs strive to obtain higher average-case speed than the worst-case speed of comparable synchronous circuits. For *non-pipelined datapaths*, the performance advantage of non-pipelined asynchronous circuits is much clearer. The latency, the only relevant metric in non-pipelined datapaths, is simply the sum of all datapath element delays in the critical path. Thus the average-case latency for asynchronous datapaths, determined roughly by the sum of the average-case delay of individual elements, is in general much lower than synchronous counterparts. Some examples of non-pipelined datapaths are Williams's divider ring [75], van Berkel et al's DCC error corrector [4], Yun et al's differential equation solver [77], and Benes et al's Huffman decoder [3].

For *pipelined datapaths*, tradeoffs are more complex. Work at Sun Labs [15] shows that asynchronous pipelines, if designed properly, can approach the speed of synchronous shift registers. However, it is unclear if asynchronous pipelines, except in some special cases [60], can ever out-perform synchronous counterparts. A goal is therefore to aim for comparable performance as a synchronous pipeline, but with the added benefits of "elasticity" (variable rate operation).

Our conjecture is that the average-case throughput (taking into account only data dependency, not operating conditions) of a *deeply-pipelined* asynchronous circuit would be close to the worst-case throughput. Shorter pipelines, though, tend to exhibit much better average-case behavior.

We describe below some recently introduced techniques to improve the average-case performance of self-timed datapaths.

2.1 Adders

In order to exploit variable data-dependent delays, self-timed datapath elements incorporate some form of completion detection mechanisms. The most common form is based on dual-rail logic [43]. However, as the datapath becomes wider, the overhead for completion detection becomes significant. Yun et al [77] observed that one way to tackle this problem is to parallelize the computation and completion detection as much as possible. Their techniques resulted in 2.8ns average-case delay for a 32-bit carry bypass adder fabricated in 0.6μm CMOS process, with only 20% completion sensing overhead on average. Another way to deal with wide datapaths is to perform *bitwise* completion detection. Martin et al [44] showed an impressive throughput gain (at the expense of sacrificing latency) using this technique.

A somewhat different twist to completion detection is called the *speculative completion*. This technique assumes the circuit normally finishes computation significantly faster than the worst-case. If the circuit cannot complete the computation in time, it aborts reporting completion. This technique requires a special auxiliary circuit called "abort detection circuit", which operates in parallel with the datapath element itself. Nowick et al [55] applied this technique to a 32-bit Brent-Kung adder and resulted in the simulated average-case delay to be less than 2ns in 0.6μm CMOS process.

2.2 Iterative structures

The development of *zero-overhead self-timed ring* technique by Williams [74] is clearly the most significant breakthrough in self-timed iterative structures. Williams showed that a self-timed ring can be designed in dual-rail domino logic with essentially zero overhead. He applied this technique to a self-timed 160ns 54-bit mantissa divider [75] as a part of a floating-point divider. This design was incorporated in a commercial microprocessor design [73].

It can be shown that this technique is generally applicable to any iterative structure in which the latency needs to be optimized. Consequently, this technique has been applied to other academic and industrial designs, such as a division and square root unit design by Matsubara and Ide [45], a self-timed packet switch design by Yun et al [78], and a Huffman decoder design by Benes et al[3]. There have been other iterative structure designs that achieve high performance with data-dependent computation times, such as a bundled data multiplier design by Kearney and Bergmann [34].

2.3 Large scale examples

In certain applications in which there is a *large* variation in processing delays between common and rare cases, asynchronous designs tend to fare much better than synchronous designs. A research group at Intel demonstrated this with their asynchronous instruction length decoder design called RAPPID ("Revolving Asynchronous Pentium Processor© Instruction Decoder") [60]. The RAPPID's length decoding out-performs, by a factor of 3, the same function inside a 400MHz Pentium II fabricated in the identical 0.25μm CMOS process. This speedup is primarily attributed to optimizations for common, short-length instructions and self-timed techniques enabling these optimizations.

In another application, an asynchronous Huffman decoder design by Benes et al [3], by exploiting the large data-dependent variation in decoding time, achieves a similar average-case performance as the worst-case performance of comparable synchronous designs, but with *5-10 times smaller area.*

So far, we have only discussed techniques to exploit variable data-dependent delays. However, if the operating condition is taken into account, we can obtain much more significant performance benefits from asynchronous circuits. This speedup is essentially due to inherent margins that must be built in synchronous systems to accommodate worst-case timing behavior but are not required for asynchronous systems. Dean [20] proposed a self-timed processor architecture called *STRiP* based on this idea. Yun et al [77] demonstrated a high-performance asynchronous differential equation solver chip, whose average-case speed (tested at 22°C and 3.3V) is 48% faster than comparable synchronous designs (designed to operate at 100°C and 3V for the slow process corner).

3 Asynchronous Processors

This is an exciting time for asynchronous processors. Recently, at Phillips Semiconductors, pagers with asynchronous chips have been released commercially to market (see below). In addition to the current academic interest in asynchronous systems, several companies such as Intel, Sharp, Sun, and HP have shown interest. The asynchronous circuits these companies have developed are showing some promise of making their way into products.

Processors are, in many ways, the most demanding application for asynchronous techniques. In addition to being extremely complex systems, processors are often the target of the most aggressive optimization that the circuit designers can bring to bear. The optimization criterion may be raw speed, low power, noise and EMC (Electro-Magnetic Compatibility) properties, or some combination of these, but it is in a processor where such requirements are the most critical. It is also the case that the organization of most modern high-performance microprocessors uses a synchronous pipelined approach, and alternative architectures may be required to achieve comparable results with asynchronous processors. But, it is the potential benefits of the asynchronous approach that are compelling in this world of highly-optimized systems. In terms of raw speed, lowered power, and improved EMC properties, asynchronous techniques may have much to offer.

Until recently, there have been relatively few asynchronous processors reported in the literature. Early work in asynchronous computer architecture includes the *Macromodule project* during the early 70's at Washington University [13, 14] and the self-timed dataflow machines called *DDM-1* and *DDM-2* (Data Driven Machine) built at the University of Utah in the late 70's [18].

More recent academic projects include the Caltech Asynchronous Microprocessor [41] which was the first asynchronous microprocessor of the VLSI era, the NSR [6], fully decoupled and built from FPGAs, and the Rotary Pipeline Processor [47] which takes a circular ring approach to the pipeline. In addition, at Sun Labs, a new *counterflow* architecture has been proposed, with a fully asynchronous implementation [62]. Some recent asynchronous processors are highlighted below.

Philips Asynchronous 80C51. At Philips Labs, an asynchronous version of the venerable 80C51 controller has been developed that exhibits nearly four times lower power than a power-optimized synchronous version. It also has significantly reduced EM emissions. These properties have convinced Philips to develop a family of these asynchronous controllers for pagers, and commercial pagers using these chips are now on the market [70, 71].

Sharp DDMP Signal Processor. Sharp Corporation has developed an experimental self-timed data driven multi-media processor aimed at digital television receivers and other applications. The fabricated processor exhibits impressive performance and power consumption, operating at a speed of 8600 Million Operations per Second and with power consumption less than 1 watt. The processor consists of 8 programmable, data-driven processing elements connected by an elastic router[65].

The Amulet. A group at the University of Manchester has built a number of versions of a self-timed micropipelined VLSI implementation of the ARM processor [26] which is an extremely power-efficient commercial microprocessor. The first-generation Amulet design is within a factor of two of the commercial ARM of the same time [56]. The second-generation Amulet 2e was targeted at embedded applications and demonstrated a modest improvement in power per MIPS over the commercial synchronous version [27, 29], as well as nearly immediate restart from full standby mode. The third-generation Amulet 3 promises further improvements in both performance and low power [30].

The Fred Architecture. Fred is a self-timed, decoupled, concurrent, pipelined computer architecture [59, 58]. It dynamically reorders instructions to issue out of order using an instruction window to organize the reordering, and allows out-of-order instruction completion. It handles exceptions and interrupts, and includes a novel functionally precise exception model that works well in the asynchronous, decoupled, out of order environment [57].

Caltech Asynchronous MIPS R3000. Subsequent to the success of their first small asynchronous processor, the asynchronous group at Caltech has built an asynchronous version of the MIPS R3000 processor. Their processor uses deep, fine-grained pipelining which is exploited naturally by the underlying asynchronous circuits. The asynchronous R3000 exhibits significantly improved MIPS/watt performance over the synchronous version when scaled to account for different processes and voltages [44].

TITAC. A group at Tokyo Institute of Technology and Tokyo University has fabricated several versions of a new architecture they call TITAC [49]. The most recent version is a full-featured 32-bit architecture that uses delay-scaling techniques to improve performance by taking real circuit delays into account, rather than conservatively assuming unbounded gate delays [64].

4 Asynchronous/Synchronous Interfaces

It is clear that there are interesting applications that can take advantage of asynchronous techniques. However, a vast majority of systems are and will continue to be synchronous. The question then is how to utilize some of the proven benefits of asynchronous circuits in a largely synchronous environment.

Some have suggested that communication between modules should be asynchronous (although the modules themselves are synchronous) because the cost of global synchrony is prohibitively high in large-scale VLSI systems. Chapiro first suggested the idea of GALS system in [11]. Yun and Donohue demonstrated a prototype GALS system with a mixture of asynchronous and synchronous modules in [84]. In this chip, synchronous modules were equipped with *pausible clocking control* to prevent synchronization failures.

Yet others have argued that maintaining precise frequency reference in a globally synchronous environment is not too difficult. The real problem is the uncertainty in clock phases. Ginosar and Kol [31] suggested an adaptive synchronization scheme to remedy this problem. Furthermore, some synchronous systems [32, 85] are moving closer to asynchronous by allowing significant time borrowing to overcome clock skew and jitter problems.

References

[1] S.S. Appleton, S.V. Morton, and M.J. Liebelt. Two-phase asynchronous pipeline control. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, April 1997.

[2] P.A. Beerel and T. Meng. Automatic gate-level synthesis of speed-independent circuits. In *ICCAD*, pages 581–586. IEEE Computer Society Press, November 1992.

[3] M. Benes, S.M. Nowick, and A. Wolfe. A fast asynchronous Huffman decoder for compressed-code embedded processors. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pages 43–56, 1998.

[4] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Schalij. A fully-asynchronous low-power error corrector for the DCC player. *IEEE JSSC*, 29(12):1429–1439, December 1994.

[5] J.G. Bredeson and P.T. Hulina. Elimination of static and dynamic hazards for multiple input changes in combinational switching circuits. *Information and Control*, 20:114–224, 1972.

[6] E. Brunvand. The NSR processor. In *Proceedings of the 26th International Conference on System Sciences*, Jan 1993.

[7] E. Brunvand and R.F. Sproull. Translating concurrent programs into delay-insensitive circuits. In *ICCAD*, pages 262–265. IEEE Computer Society Press, November 1989.

[8] S.M. Burns. General condition for the decomposition of state holding elements. In *Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pages 48–57. IEEE Computer Society Press, November 1996.

[9] S.M. Burns and A.J. Martin. Syntax-directed translation of concurrent programs into self-timed circuits. In *Advanced Research in VLSI*, pages 35–50. MIT Press, Cambridge, MA, 1988.

[10] S. Chakraborty, D.L. Dill, and K.Y. Yun. Min-max timing analysis and its application to asynchronous circuits. *Proceedings of the IEEE*, 87(2), Feb 1999.

[11] D.M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems.* PhD thesis, Stanford University, October 1984.

[12] T.-A. Chu. Synthesis of self-timed vlsi circuits from graph-theoretic specifications. Technical Report MIT-LCS-TR-393, MIT, 1987. Ph.D. Thesis.

[13] W.A. Clark. Macromodular computer systems. In *Spring Joint Computer Conference.* AFIPS, April 1967.

[14] W.A. Clark and C.E. Molnar. Macromodular system design. Technical Report 23, Computer Systems Laboratory, Washington University, April 1973.

[15] W.S. Coates, J.K. Lexau, I.W. Jones, S.M. Fairbanks, and I. E. Sutherland. A FIFO data switch design experiment. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pages 4–17, 1998.

[16] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Methodology and tools for state encoding in asynchronous circuit synthesis. In *DAC*, June 1996.

[17] A. Davis, B. Coates, and K. Stevens. Automatic synthesis of fast compact self-timed control circuits. In *IFIP Working Conference on Asynchronous Design Methodologies*, 1993.

[18] A.L. Davis. The architecture and system method for DDM1: A recursively structured data-driven machine. In *5th Annual Symp. on Computer Architecture*, April 1978.

[19] P. Day and J.V. Woods. Investigation into micropipeline latch design styles. *IEEE TVLSI*, 3(2):264–272, June 1995.

[20] M.E. Dean. *STRiP: A Self-Timed RISC Processor Architecture.* PhD thesis, Stanford University, 1992.

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

[22] E.B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9(2):90–99, 1965.

[23] R.M. Fuhrer, B. Lin, and S.M. Nowick. Symbolic hazard-free minimization and encoding of asynchronous finite state machines. In *ICCAD*, pages 604–611, November 1995.

[24] S. Furber. Computing without clocks: Micropipelining the ARM processor. In Graham Birtwistle and Al Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 211–262. Springer-Verlag, 1995.

[25] S.B. Furber and P. Day. Four-phase micropipeline latch control circuits. *IEEE TVLSI*, 4(2):247–253, June 1996.

[26] S.B. Furber, P. Day, J.D. Garside, N.C. Paver, and J.V. Woods. A micropipelined ARM. In *Proceedings of VLSI93*, Grenoble, France, 1993.

[27] S.B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, and N.C. Paver. AMULET2e: An asynchronous embedded controller. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, April 1997.

[28] S.B. Furber and J. Liu. Dynamic logic in four-phase micropipelines. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, March 1996.

[29] J. D. Garside, S. Temple, and R. Mehra. The AMULET2e cache system. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, March 1996.

[30] J.D. Garside, S.B. Furber, and S.-H. Chung. AMULET3 revealed. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, April 1999.

[31] R. Ginosar and R. Kol. Adaptive synchronization. In *ICCD*, pages 188–189, October 1998.

[32] D. Harris and M.A. Horowitz. Skew-tolerant domino circuits. *IEEE JSSC*, 32(11):1702–1711, November 1997.

[33] M.B. Josephs and J.T. Udding. An overview of D-I algebra. In *HICSS*, volume I, pages 329–338. IEEE Computer Society Press, January 1993.

[34] D. Kearney and N.W. Bergmann. Bundled data asynchronous multipliers with data dependant computation times. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, April 1997.

[35] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *DAC*, pages 56–62. ACM, June 1994.

[36] D.S. Kung. Hazard-non-increasing gate-level optimization algorithms. In *ICCAD*, pages 631–634, November 1992.

[37] L. Lavagno, C.W. Moon, R.K. Brayton, and A. Sangiovanni-Vincentelli. Solving the state assignment problem for signal transition graphs. In *DAC*, pages 568–572, June 1992.

[38] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for synthesis and testing of asynchronous circuits.* Kluwer Academic, 1993.

[39] B. Lin and S. Devadas. Synthesis of hazard-free multi-level logic under multiple-input changes from binary decision diagrams. In *ICCAD*, pages 542–549, Nov. 1994.

[40] A. Marshall, B. Coates, and P. Siegel. The design of an asynchronous communications chip. *IEEE Design and Test*, 11(2):8–21, Summer 1994.

[41] A. Martin, S. Burns, T.K. Lee, D. Borkovic, and P. Hazewindus. The design of an asynchronous microprocessor. In *Proc. Cal Tech Conference on VLSI*, 1989.

[42] A.J. Martin. Programming in VLSI: From communicating processes to delay-insensitive circuits. In C.A.R. Hoare, editor, *Developments in Concurrency and Communication*, pages 1–64. Addison-Wesley, Reading, MA, 1990.

[43] A.J. Martin. Asynchronous datapaths and the design of an asynchronous adder. *Formal Methods in System Design*, 1(1):119–137, July 1992.

[44] A.J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings. The design of an asynchronous MIPS R3000 microprocessor. In *Advanced Research in VLSI*, September 1997.

[45] G. Matsubara and N. Ide. A low power zero-overhead self-timed division and square root unit combining a single-rail static circuit with a dual-rail dynamic circuit. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, April 1997.

[46] T.H.-Y. Meng, R.W. Brodersen, and D.G. Messerschmitt. Automatic synthesis of asynchronous circuits from high-level specifications. *IEEE TCAD*, 8(11):1185–1205, November 1989.

[47] S. Moore, P. Robinson, and S. Wilcox. Rotary pipeline processors. *IEE Proceedings, Computers and Digital Techniques*, 143(5), September 1996.

[48] C. Myers and T. Meng. Synthesis of Timed Asynchronous Circuits. *IEEE TVLSI*, 1(2):106–119, June 1993.

[49] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako, and A. Takamura. TITAC: Design of a quasi-delay-insensitive microprocessor. *IEEE Design & Test of Computers*, 11(2):50–63, 1994.

[50] S.M. Nowick. Automatic synthesis of burst-mode asynchronous controllers. Technical report, Stanford University, March 1993. Ph.D. Thesis (available as Stanford Univ. Cptr. Sys. Lab. tech report, CSL-TR-95-686, Dec. 95).

[51] S.M. Nowick, M.E. Dean, D.L. Dill, and M. Horowitz. The design of a high-performance cache controller: a case study in asynchronous synthesis. *INTEGRATION, the VLSI journal*, 15(3):241–262, October 1993.

[52] S.M. Nowick and D.L. Dill. Synthesis of asynchronous state machines using a local clock. In *ICCD*, pages 192–197. IEEE Computer Society Press, October 1991.

[53] S.M. Nowick and D.L. Dill. Exact two-level minimization of hazard-free logic with multiple-input changes. *IEEE TCAD*, 14(8):986–997, August 1995.

[54] S.M. Nowick, N.K. Jha, and F.-C. Cheng. Synthesis of asynchronous circuits for stuck-at and robust path delay fault testability. *IEEE TCAD*, 16(12):1514–1521, December 1997.

[55] S.M. Nowick, K.Y. Yun, and P.A. Beerel. Speculative completion for the design of high-performance asynchronous dynamic adders. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, April 1997.

[56] N.C. Paver. *The Design and Implementation of an Asynchronous Microprocessor.* PhD thesis, University of Manchester, 1994.

[57] W.F. Richardson and E. Brunvand. Precise exception handling for a self-timed processor. In *ICCD*, pages 32–37, Los Alamitos, CA, October 1995. IEEE Computer Society Press.

[58] W.F. Richardson and E. Brunvand. Architectural considerations for a self-timed decoupled processor. *IEE Proceedings, Computers and Digital Techniques*, 143(5), September 1996.

[59] W.F. Richardson and E. Brunvand. Fred: An architecture for a self-timed decoupled computer. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1996.

[60] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev. RAPPID: an asynchronous instruction length decoder. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1999.

[61] P. Siegel, G. De Micheli, and D. Dill. Technology mapping for generalized fundamental-mode asynchronous designs. In *DAC*, pages 61–67. ACM, June 1993.

[62] R.F. Sproull, I.E. Sutherland, and C.E. Molnar. The counterflow pipeline processor architecture. *IEEE Design & Test of Computers*, 11(3):48–59, Fall 1994.

[63] I.E. Sutherland. Micropipelines. *CACM*, 32(6):720–738, June 1989.

[64] A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, U. Ueno, and T. Nanya. TITAC-2: an asynchronous 32-bit microprocessor based on scalable-delay-insensitive model. In *ICCD*, pages 288–294, October 1997.

[65] H. Terada, S. Miyata, and M. Iwata. Ddmps: Self-timed super-pipelined data-driven multimedia processors. *Proceedings of the IEEE*, 87(2), Feb 1999.

[66] M. Theobald, S.M. Nowick, and T. Wu. Espresso-HF: a heuristic hazard-free minimizer for two-level logic. In *DAC*, pages 71–76, June 1996.

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

[68] C.H. van Berkel and R.W.J.J. Saeijs. Compilation of communicating processes into delay-insensitive circuits. In *ICCD*, pages 157–162. IEEE Computer Society Press, 1988.

[69] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Schalij. Asynchronous Circuits for Low Power: a DCC Error Corrector. *IEEE Design & Test*, 11(2):22–32, June 1994.

[70] H. van Gageldonk. An asynchronous low-power 80C51 microcontroller. Technical report, Eindhoven University of Technology, Sept 1998. Ph.D. Thesis.

[71] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80C51 microcontroller. In *IEEE Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, April 1998.

[72] V.I. Varshavsky, M.A. Kishinevsky, V.B. Marakhovsky, V.A. Peschansky, L.Y. Rosenblum, A.R. Taubin, and B.S. Tzirlin. *Self-timed Control of Concurrent Processes.* Kluwer Academic Publishers, 1990. Russian edition: 1986.

[73] T. Williams, N. Patkar, and G. Shen. SPARC64: A 64-b 64-active-instruction out-of-order-execution MCM processor. *IEEE JSSC*, 30(11):1215–1226, November 1995.

[74] T.E. Williams. *Self-Timed Rings and their Application to Division.* PhD thesis, Stanford University, June 1991.

[75] T.E. Williams and M.A. Horowitz. A zero-overhead self-timed 160ns 54b CMOS divider. *IEEE JSSC*, 26(11):1651–1661, November 1991.

[76] K.Y. Yun, P.A. Beerel, and J. Arceo. High-performance two-phase micropipeline building blocks: double edge-triggered latches and burst-mode select and toggle circuits. *IEE Proceedings, Circuits, Devices and Systems*, 143(5):282–288, October 1996.

[77] K.Y. Yun, P.A. Beerel, V. Vakilotojar, A.E. Dooply, and J. Arceo. The design and verification of a high-performance low-control-overhead asynchronous differential equation solver. *IEEE TVLSI*, 6(4):643–655, December 1998.

[78] K.Y. Yun, S. Chakraborty, K.W. James, R. Fairlie-Cuninghame, and R.L. Cruz. A self-timed real-time sorting network. In *ICCD*, pages 427–434, October 1998.

[79] K.Y. Yun and D.L. Dill. Automatic synthesis of 3D asynchronous finite-state machines. In *ICCAD*, Nov. 1992.

[80] K.Y. Yun and D.L. Dill. Unifying synchronous/asynchronous state machine synthesis. In *ICCAD*, pages 255–260. IEEE Computer Society Press, November 1993.

[81] K.Y. Yun and D.L. Dill. A high-performance asynchronous SCSI controller. In *ICCD*, pages 44–49, Oct. 1995.

[82] K.Y. Yun and D.L. Dill. Automatic synthesis of extended burst-mode circuits: part I (specification and hazard-free implementations). *IEEE TCAD*, 18(2):101–117, February 1999.

[83] K.Y. Yun and D.L. Dill. Automatic synthesis of extended burst-mode circuits: part II (automatic synthesis). *IEEE TCAD*, 18(2):118–132, February 1999.

[84] K.Y. Yun and R.P. Donohue. Pausible clocking: A first step toward heterogeneous systems. In *ICCD*, pages 118–123, October 1996.

[85] K.Y. Yun and A.E. Dooply. Optimal evaluation clocking of self-resetting domino pipelines. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 121–124, January 1999.