Automatic Synthesis of Extended Burst-Mode Circuits Using Generalized C-elements*

Kenneth Y. Yun
Department of Electrical and Computer Engineering
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093-0407 USA
kyy@UCSD.EDU

Abstract

This paper presents a new automatic synthesis technique for extended burst-mode *circuits, a class of asynchronous circuits that allow multiple-input changes between state transitions and a choice of next states based on input signal levels. The target implementation is a pseudo-static asymmetric CMOS complex gate per each output, known as* generalized C-element *[3, 12]. The synthesis algorithm generates hazard-free covers for* set and reset *functions of each output using Nowick and Dill's exact hazard-free logic minimization algorithm [14]. Each output circuit is formed by mapping its set and reset logic to N and P stacks of an asymmetric CMOS gate connected to a sustainer; long series stacks are decomposed into static gates followed by short stacks. A simple heuristic is used to ensure that no short circuit paths exist from $V_{\rm dd}$ to ground. The resulting circuits for small-to-medium size extended burst-mode specifications are 40% smaller and 30% faster than two-level circuits generated by the 3D synthesis tool [19] and significantly smaller and faster than complex-gate circuits generated by the method of Kudva et al [9].*

1. Introduction

With clock speeds exceeding 300MHz in high performance microprocessor designs, designers are discovering the limitations of synchronous designs: complex clock skew management, high peak power dissipation, worst case design requirements, etc. Many have been advocating asynchronous design as a possible solution to these problems. As a result, there have been many recent advances in asynchronous design techniques, particularly in the area of automated synthesis [1, 15, 4, 8, 10, 12, 13, 17]. There have been some attempts at real system designs employing asynchronous techniques as well [16, 2, 5, 7, 11]. It is becoming increasingly clear that *system designers* recognize asynchronous design as a viable alternative to strictly synchronous design at least for interface modules and key performance or power critical areas of system designs.

Arguably the most difficult part of asynchronous design is controller design, because of complex hazard avoidance requirements and its implications on performance of overall circuits. This paper focuses on the performance side of controller design, namely an efficient synthesis technique for extended burst-mode circuits [19, 17, 20], which have proved to be practically useful and promised good performance. The new synthesis algorithm described in this paper is geared toward synthesizing high performance circuits for small-to-medium size extended burst-mode specifications. Previous synthesis techniques for (extended) burst-mode specifications targeted two-level AND-OR circuits [19, 13] or multi-level circuits reduced from multiplexor trees [20]. These implementations are derived from *on-set covers* of next-state logic. These synthesis techniques produced efficient, high-performance circuits for large specifications, utilizing a global logic minimization algorithm [14]. However, while attempting to synthesize circuits for specifications with very stringent performance requirements (as a part of the Asynchronous Instruction Decoder Project at Intel Corporation), it was determined that two-level circuits may be inefficient in some cases. The natural course of action at that point was to investigate alternate circuit structures. After a series of experiments by this author and other researchers, it was discovered that pseudo-static asymmetric CMOS complex gates, known as *generalized C-elements*, are faster *in the operating environment* than fully complementary logic gates because they tend to have less input capacitance and higher output drives than static CMOS gates with equivalent number of fanins. A model of generalized C-element is shown in figure 1. This paper thus presents a new synthesis algorithm for extended burst-mode circuits using generalized C-elements as target implementations. In addition, the synthesis results are compared to the implementations generated by Yun's 3D synthesis tool [17] and by the complex-gate method of Kudva et al [9].

*This research was supported in part by a gift from Intel Corporation.

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

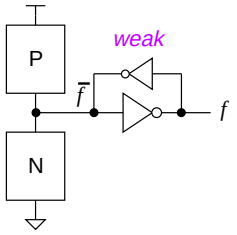

Figure 1: A generalized C-element with a sustainer: $P \neq \overline{N}$

The synthesis procedure consists of two steps: state assignment and logic implementation. The state assignment step ensures that every transition specified in the specification is function-hazard-free. As in the 3D synthesis method, each specification state is assigned to a layer; compatible layers are merged to minimize the number of layers; finally, layers are encoded so that every specified transition between the layers is free of critical races. The logic implementation step finds covers for the set and reset regions for each output and maps the minimized set and reset logic to N and P stacks of a CMOS gate followed by a sustainer.

The experimental results show that this new approach has many advantages over other synthesis methods [13, 17], which implement the next-state functions as two-level AND-OR or multi-level circuits. In most cases, the circuits synthesized using this new method are considerably smaller (have fewer literal counts) than the circuits synthesized by other methods, and thus faster. This is because the number of dynamic transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$) for most outputs are substantially fewer than the total number of $0 \rightarrow 1$, $1 \rightarrow 1$ and $1 \rightarrow 0$ transitions.

There is a long line of automated synthesis techniques for (extended) burst-mode circuits which produce efficient circuits. Also, generalized C-elements have been used in asynchronous circuit designs elsewhere [3, 6, 12], and, of course, set/reset logic, such as SR-latches and C-elements, has been used by many asynchronous synthesis tools. However, this synthesis technique brings those two together with a global logic minimization algorithm.

2. Overview

2.1. Specification

Figure 2 describes an extended burst-mode state machine (*biu-fifo2dma*) having 4 inputs (*ok*, *cntgt1*, *fain*, *dackn*) and 2 outputs (*frout*, *dreq*). Signals not enclosed in angle brackets, such as *ok*, *fain*, and *dackn* are *edge signals*. Edge signals ending with $+$ or $-$ are *terminating signals*; the ones ending with $*$ are *directed don't cares*. If a state transition is labeled with a directed don't care $a*$, then the following state transition must be labeled with $a*$ or $a+$ or $a-$. A terminating signal $a+$ denotes a $0 \rightarrow 1$ transition of a if a was initially 0, and no transition at all if a was initially 1.

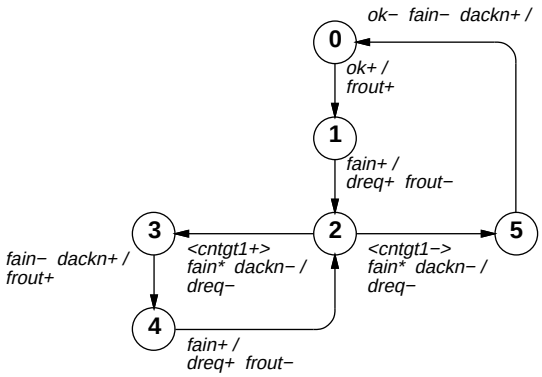

Figure 2: *Biu-fifo2dma* specification.

A sequence of state transitions labeled with $a*$ and terminated with $a+$ represents a *single* $0 \rightarrow 1$ transition of a at any point in the sequence. A terminating signal not immediately preceded by a directed don't care represents a *compulsory* transition. Signals enclosed in angle brackets, such as *cntgt1*, represent *conditional* or *level signals*. $\langle cntgt1+ \rangle$ and $\langle cntgt1- \rangle$ denote conditional clauses "if *cntgt1* is high" and "if *cntgt1* is low."

An input burst is a non-empty set of input edges (*terminating* or *directed don't care*) at least one of which must be a compulsory transition. An output burst consists of a possibly empty set of output edges. If a state transition is not labeled with a level signal, the signal may change freely during the transition. However, if an edge signal is not mentioned in a transition, it is not allowed to change.

In a given state, when all the specified conditional signals have correct values and when all the specified terminating signals in the input burst have changed, the machine generates the corresponding output burst and moves to a new state. Specified edges in the input burst may appear in arbitrary temporal order. However, the conditional signals must stabilize to correct levels before any compulsory edge in the input burst appears and must hold their values until after all of the terminating edges appear. Outputs may be generated in any order, but the next set of compulsory edges from the next input burst may not appear until the machine has stabilized.

2.2. Example

The *biu-fifo2dma* example in figure 2 is used to illustrate the synthesis procedure. The first step of the synthesis procedure is the state assignment step. Details of this step will not be discussed, because it is virtually identical[1] to the same step in the 3D synthesis [17]. In summary, each specification state is assigned to a layer (there are 6 layers initially); compatible layers are merged to minimize the number of lay-

[1] Only the definition of DHF-compatibility is modified.

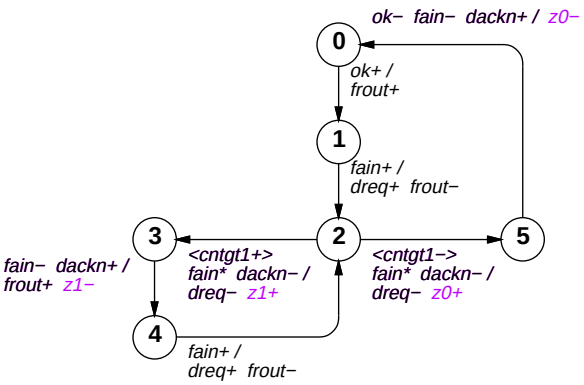

Figure 3: *Biu-fifo2dma* specification with state variable transitions annotated.

ers (there are 3 layers after minimization); finally, layers are encoded so that every specified transition between layers is free of critical races. Because the synthesis step assumes that state variable transitions are done concurrently with output transitions, state variable transitions can be *backannotated* to the initial specification to form a specification with a unique next-state code property [17] as shown in figure 3.

The specification annotated with state variable transitions as above maps directly to a *next-state table*, a 3-dimensional tabular representation of the next-state function $\delta : X \times Y \times Z \rightarrow Y \times Z$, where X is a non-empty set of primary input symbols, Y a non-empty set of primary output symbols, and Z a possibly empty set of internal state variable symbols.

The second step of the synthesis procedure is the logic implementation step. From the next-state table and the specified transitions in the annotated specification, a set of states (minterms in the next-state table) that enable each output to rise is determined. For example, for the state variable z_0, there are just two states ($ok\ cntgt1\ fain\ dackn\ frout\ dreq\ z_0\ z_1 = 10\text{x}00100$) that enable z_0 to rise (see state 2 in figure 3). The set logic for z_0 is the function $z_{0\text{set}}$ whose on-set is $ok\ cntgt1\ fain\ dackn\ frout\ dreq\ z_0\ z_1 = 10\text{x}00100$ and whose off-set is the same as z_0's. This logic is minimized using Nowick and Dill's exact two-level logic minimizer, which ensures that every $0 \rightarrow 1$ transition of $z_{0\text{set}}$ is dynamic-hazard-free. Similarly, the logic minimization is carried out on the function $z_{0\text{reset}}$, whose on-set is the set of states that enable z_0 to fall and whose off-set is the same as z_0's on-set, to find the reset logic for z_0.

The resulting logic is guaranteed to be hazard-free for all the specified transitions in the annotated state diagram. When both $z_{0\text{set}}$ and $z_{0\text{reset}}$ are at 0, the sustainer will hold the logic value.

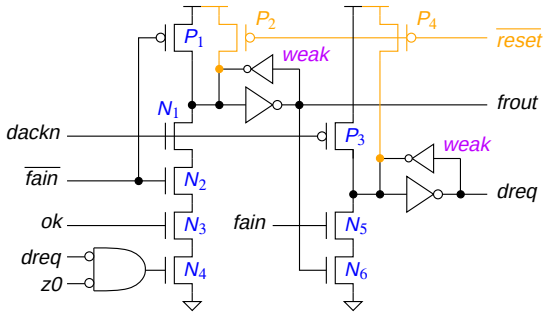

Figure 4: *Biu-fifo2dma* circuit (z_0 and z_1 gates not shown).

2.3. Circuit Operation

A machine cycle consists of an input burst followed by a concurrent output and state burst. Initially or after completion of the previous output and state burst, the machine waits for an input burst to arrive. When the machine detects all of the terminating edges of the input burst, it generates a concurrent output/state burst. As in the 3D implementation of extended burst-mode circuits, no fed-back output or state variable change arrives at the gate input until all of the specified edges in the output and state burst have appeared at the gate output. These conditions are met by inserting delays in the feedback paths as necessary.

Consider the example, *biu-fifo2dma*, shown in figures 3 and 4. When the machine is reset, $\overline{reset}$ goes low causing *frout* and *dreq* to fall. During reset, transistors P_2, P_4, N_1, N_2, and N_4 are on but all other transistors are off, because $fain\ dackn\ ok\ frout\ dreq\ z_0 = 01000$. When $\overline{reset}$ goes high turning off P_2 and P_4, the weak inverters are used to sustain the logic levels. When the environment raises *ok*, N_3 turns on, causing *frout* to rise, which in turn causes N_6 to turn on. Once *frout* rises, the machine is in state 1, waiting for the environment to raise *fain*. When *fain* rises, N_5 turns on, causing *dreq* to rise. Concurrently, as $\overline{fain}$ falls, N_1 turns off and P_1 turns on, causing *frout* to fall.

3. Logic Implementation

The synthesis method in this paper (3D-gC) produces two-level AND-OR circuits for both set logic (f_{set}) and reset logic (f_{reset}). The N stack of the generalized C-element in figure 1 is simply the N stack of the fully complementary complex AND-OR-NOT gate that implements $\overline{f_{\text{set}}}$; the P stack of the generalized C-element is the P stack of the full complementary complex AND-OR-NOT gate that implements f_{reset}.

3.1. Hazards in Generalized C-elements

The hazard avoidance techniques used for two-level AND-OR apply directly here, because of the way pull-down and pull-up stacks are implemented as described above. The

only difference is that no special precautions are necessary to make $1 \rightarrow 1$ transitions hazard-free.

In the generalized C-element implementations described in this paper, a property of the AND-OR structure is used to avoid static hazards: namely, all $0 \rightarrow 0$ transitions free of function hazards are also free of logic hazards [17, 14]. When f undergoes a $1 \rightarrow 1$ transition, $f_{\rm reset}$ remains low, keeping the P stack turned off. The N stack, in the meantime, may remain turned on or off, but the old value will be maintained by the sustainer (In fully complementary MOS gates, static hazards are possible because the N and P stacks are duals of each other, i.e., when the N stack is turned off, the P stack is on and vice versa). The key point is that both N and P stacks are derived from two-level AND-OR logic so that certain properties of two-level AND-OR can be exploited for hazard freedom.

On the other hand, special steps must be taken to avoid dynamic hazards. As in two-level AND-OR logic, for a $0 \rightarrow 1$ transition to be hazard-free, all on-set minterms in each trajectory of the transition must be covered by a single cube, and every cube that intersects the trajectory must also include the end-point of the trajectory [17, 14]. Consider a transition $a * b+$ ($a = b = 0$ and $c = d = 1$ initially), in which f is supposed to rise monotonically when b rises, regardless of the behavior of a. Suppose that $\overline{a}$ and b change as shown in figure 5. $\overline{f}$ starts to discharge while b and $\overline{a}$ are both high, stops when $\overline{a}$ falls, and starts again when N_3 is fully turned on (after the AND output rises). Although it is very unlikely that complex gates exhibit glitches as illustrated in figure 5, it may be worthwhile to avoid any such possibilities. The synthesis algorithm removes any possibilities of dynamic hazards using a dynamic-hazard-free state minimization algorithm similar to the one used in [19] and Nowick and Dill's dynamic-hazard-free logic minimization algorithm [14].

To summarize, for the output of a generalized C-element (see figure 1) to be hazard-free for a set of specified transitions, the following requirements must be met:

1. All specified transitions are function-hazard-free;

2. There are no *reachable* states in which both P and N stacks are on;

3. N stack is hazard-free for all specified $0 \rightarrow 1$ transitions; P stack is hazard-free for all specified $1 \rightarrow 0$ transitions.

The state assignment step ensures that the requirement 1 is met. The requirement 2 is met by ensuring that the on-set of $f_{\rm set}$ ($f_{\rm reset}$) is devoid of off-set (on-set) minterms of f. Nowick and Dill's logic minimization algorithm in conjunction with the state assignment step ensures that the requirement 3 is met.

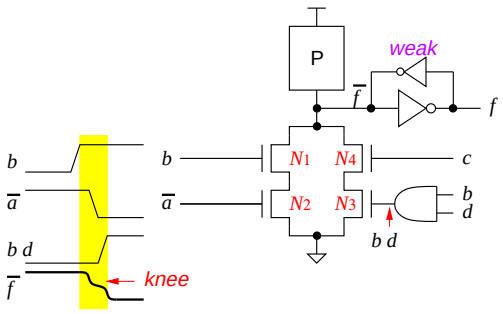

Figure 5: Dynamic hazard in generalized C-elements.

3.2. Signal Placement and Decomposition of Long Series Stacks

In (extended) burst-mode circuits, the order of signal arrivals is largely pre-determined, so the signal placement can be optimized for performance. In the 3D-gC synthesis method, primary input signals that enable an output to change, i.e., *trigger* signals for the output, are placed at the *top* of the stack (farthest from $V_{\rm dd}$/Ground). Fed-back outputs and state variables are placed at the *bottom* of the stack (nearest to $V_{\rm dd}$/Ground), because feedback signals do not enable outputs to change.

In general combinational circuits, long series stacks are not acceptable, because the worst-case delay grows quadratically as the stack size and the order of signal arrivals is not known a priori. However, in most (extended) burst-mode circuits, only 2-3 signals at the top of the stack are late arriving signals. Thus the actual delay grows closer to linearly than quadratically as the stack size. Post-layout simulations for actual designs [18] show that the circuits with the stack size of 5-6 and 2-3 trigger signals have roughly the same delays as the worst-case delay for the stack size of 4.

Although somewhat longer series stacks can be used in (extended) burst-mode circuits than in conventional combinational circuits, larger specifications and deep submicron designs require a capability to decompose long stacks. The most straight-forward way to decompose a long stack is to partition the signals and map every partition with more than one signal to a static AND/NAND followed by a transistor as shown in figure 6. This decomposition is hazard-free because each series stack corresponds to an AND gate in the AND-OR network that implements $f_{\rm set}$ or $f_{\rm reset}$ and decomposing AND gates recursively is hazard-free. However, arbitrary partitioning is not allowed because it can lead to DC-path problems during dynamic transitions. Consider the *frout* circuit in figure 6b. In state 1, the N stack is on and the P stack is off while the machine awaits *fain* to rise. When $\overline{fain}$ falls, the P and N stacks should turn on and off simultaneously (as do basic CMOS gates when they switch). But in the circuit in figure 6b, both the N and P stacks are on for the duration of the AND gate delay — there exists a short

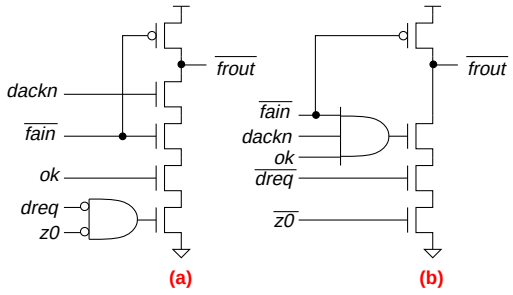

Figure 6: Decomposition.

circuit path from $V_{\rm dd}$ to Ground.

A simple constraint used to avoid this problem is as follows: when partitioning long series stacks for f_{reset} (f_{set}), the trigger signals for f_{set} (f_{reset}) are never placed in a partition with more than one signal. In other words, when a trigger signal toggles, enabling a series stack, say, an N stack to turn on, the PMOS transistors driven by the same signal should turn off immediately, in order to prevent a DC-path. Of course, it is only a sufficient condition to avoid DC-path problems, but this method works well in practice.

4. Experiments

A minor modification was made to the state assignment part of the 3D synthesis tool to account for the new definition of state compatibility. A new program was devised to interface the modified 3D front end to Fuhrer and Nowick's implementation of the exact logic minimizer. Of the 20 examples synthesized, 14 required no decomposition, i.e., every output and state variable can be mapped to a single generalized C-element with no series stack with more than 4 transistors. Out of the 6 examples that required decomposition (the ones with * next to its name in table 1), 3 examples had just one output that required 5 or 6 transistors in series with no more than 3 trigger signals, which means that alternative direct gC implementations (without decomposition) would not degrade the performance.

The total literal counts were considerably fewer than two-level solutions. The literal counts in the table actually represent the total number required for both N and P stacks; thus the actual transistor counts (excluding sustainers and reset logic) are equal to the literal counts, whereas for two-level solutions the transistor counts are more than double the literal counts shown in the table.

The worst-case performance of the gC circuit can be approximated by an equivalent AND gate with the number of fanins equal to the number of transistors in the longest series stack. According to a series of post-layout SPICE simulations [18], gC circuits (for diffeq examples) are at least 30% faster than equivalent two-level circuits.

The results were then compared to the complex-gate implementations by Kudva et al [9] (see table 2). Output f in their synthesis method is implemented with an asymmetric complex gate, P and N stacks of which correspond to pass transistor networks of f and $\overline{f}$ in sum-of-products form respectively. In some cases, their implementations have smaller area and better performance than two-level SOP implementations generated by the 3D tool, because $1 \rightarrow 1$ transitions need not be covered with single cubes. However, the implementations produced by the 3D-gC method are significantly better (smaller area and delay) than theirs. Because the set of required cubes for $f_{\rm set}$ ($f_{\rm reset}$) is a proper subset of f ($\overline{f}$) but the dynamic-hazard-free covering constraints are the same, the N stack (P stack) of the 3D-gC implementation has strictly fewer transistors, thus fewer fanins and higher current drives, than the P stack (N stack) of the implementations produced by Kudva's method. In addition, the sustainers attached to gC gates do not cost significantly. Most asynchronous controller outputs need to drive large capacitive loads; thus having an extra inverting stage is almost never a burden. In fact, driving large capacitive loads directly with complex gates may be more inefficient because enlarging the transistors for higher drives means higher fanin loads.

	Specification		**State vars**		**Literals**	
	States	I/O	*3D*	*gC*	*3D*	*gC*
iccad93ex	3	2 / 2	2	0	20	8
edac93ex	4	3 / 2	2	1	32	13
condtest	4	3 / 2	2	1	30	18
dff	4	2 / 2	2	0	28	16
q42	4	2 / 2	1	1	27	15
select2ph	4	2 / 2	2	0	42	24
selmerge2ph	8	3 / 2	2	1	89	52
ring-counter*	8	1 / 2	1	1	45	160
binary-counter	32	1 / 4	3	3	94	56
pe-send-ifc*	11	5 / 3	2	2	90	57
pe-rcv-ifc*	12	4 / 4	3	2	84	54
dramc*	12	7 / 6	1	0	71	38
stetson-p3	8	4 / 2	1	0	16	8
fifocellctrl	3	2 / 2	1	1	11	10
scsi-targ-send*	7	4 / 2	3	3	53	35
scsi-init-send	7	4 / 2	2	2	31	22
diffeq-ALU1	7	3 / 5	2	2	43	38
diffeq-ALU2*	14	5 / 7	3	2	141	25
diffeq-MUL1	4	3 / 3	1	1	42	32
diffeq-MUL2	3	3 / 3	0	0	15	13

Table 1: Experimental Results.

5. Conclusion

Based on a detailed analysis of hazard avoidance techniques for generalized C-elements, a new automatic synthesis method for extended burst-mode circuits implemented with generalized C-elements was devised. The experimental results show that gC circuits for small-to-medium size extended burst-mode specifications are 40% smaller and 30% faster than two-level circuits generated by the 3D syn-

Specification	Output	Tallest Stack			
		Kudva		3D-gC	
		P	N	N	P
chu-ad-opt	lr	4	1	4	1
	dr	2	2	1	1
	zr	4	1	4	1
vanbek-ad-opt	lr	4	1	4	1
	dr	2	3	1	3
	zr	2	3	1	3
sendr-done	DoneS	3	1	3	1
	zzz00	2	2	2	1
sbuf-read-ctl	Ack	3	1	3	1
	RamRdSBuf	2	2	2	2
	BusReq	4	1	4	1
	zzz00	2	2	1	1
q42	a4	2	3	1	3
	r2	2	2	2	2
	zzz00	3	2	2	2

Table 2: Comparison to Kudva's.

thesis tool. For further optimization, technology mapping techniques to optimize average-case delays need to be examined. Finally, because generalized C-elements are composed of series stacks, we need to pay close attention to charge sharing problems. Presently, the output-driving inverter is made large in order to reduce the charge sharing problem. In the future, automatic charge sharing reduction techniques will be investigated.

Acknowledgment

Peter Beerel's comment on don't care set led to a simplification of the synthesis and better results.

References

[1] Peter A. Beerel. *CAD Tools for the Synthesis, Verification, and Testability of Robust Asynchronous Circuits.* PhD thesis, Stanford University, 1994.

[2] Erik Brunvand. The NSR processor. In *Proc. Hawaii International Conf. System Sciences*, volume I. IEEE Computer Society Press, January 1993.

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

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

[5] A. Davis, B. Coates, and K. Stevens. The Post Office experience: Designing a large asynchronous chip. In *Proc. Hawaii International Conf. System Sciences*, volume I, pages 409–418. IEEE Computer Society Press, January 1993.

[6] Paul Day and J. Viv Woods. Investigation into micropipeline latch design styles. *IEEE Transactions on VLSI Systems*, 3(2), June 1995.

[7] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, S. Temple, and J. V. Woods. The design and evaluation of an asynchronous microprocessor. In *Proc. International Conf. Computer Design (ICCD)*. IEEE Computer Society Press, October 1994.

[8] Michael Kishinevsky, Alex Kondratyev, Alexander Taubin, and Victor Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design.* Series in Parallel Computing. John Wiley & Sons, 1994.

[9] P. Kudva, G. Gopalakrishnan, H. Jacobson, and S. M. Nowick. Synthesis of hazard-free customized CMOS complex-gate networks under multiple-input changes. In *Proc. ACM/IEEE Design Automation Conference.* IEEE Computer Society Press, June 1996.

[10] Luciano Lavagno. *Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs.* PhD thesis, U.C. Berkeley, November 1992. Technical report UCB/ERL M92/140.

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

[12] Chris J. Myers and Teresa H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.

[13] S. M. Nowick and B. Coates. Automated design of high-performance asychronous state machines. In *Proc. International Conf. Computer Design (ICCD)*. IEEE Computer Society Press, October 1994.

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

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

[16] 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 Journal of Solid-State Circuits*, 29(12):1429–1439, December 1994.

[17] K. Y. Yun. *Synthesis of Asynchronous Controllers for Heterogeneous Systems.* PhD thesis, Stanford University, August 1994. Technical Report CSL-TR-94-644.

[18] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. E. Dooply, and J. Arceo. A low-control-overhead asynchronous differential equation solver. To appear in *ESSCIRC-96*.

[19] K. Y. Yun and D. L. Dill. Unifying synchronous/asynchronous state machine synthesis. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 255–260. IEEE Computer Society Press, November 1993.

[20] K. Y. Yun, B. Lin, D. L. Dill, and S. Devadas. Performance-driven synthesis of asynchronous controllers. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 550–557, November 1994.