Fsimac: A Fault Simulator for Asynchronous Sequential Circuits

Susmita Sur-Kolay[1], Marly Roncken[2], Ken Stevens[2], Parimal Pal Chaudhuri[3], and Rob Roy[4]

[1]Indian Statistical Institute, Calcutta 700035, India – ssk@isical.ac.in

[2]Intel Corporation, Santa Clara CA / Hillsboro OR, USA – mroncken@scdt.intel.com / kstevens@ichips.intel.com

[3]Bengal Engineering College, Howrah 711103, India, – ppc@ppc.becs.ac.in

[4]Mobilian Corporation, Hillsboro, OR, USA – robroy@technologist.com

Abstract

At very high frequencies, the major potential of asynchronous circuits is absence of clock skew and, through that, better exploitation of relative timing relations. This paper presents Fsimac*, a gate-level fault simulator for stuck-at and gate-delay faults in asynchronous sequential circuits. Fsimac not only evaluates combinational logic and typical asynchronous gates such as Muller C-elements, but also complex domino gates, which are widely used in high-speed designs. Our algorithm for detecting feedback loops is designed so as to minimize the iterations for simulating the unfolded circuit. We use min-max timing analysis to compute the bounds on the signal delays. Stuck-at faults are detected by comparing logic values at the primary outputs against the corresponding values in the fault-free design. For delay faults, we additionally compare min-max time stamps for primary output signals. Fault coverage reported by Fsimac for pseudo-random tests generated by Cellular Automata show some very good results, but also indicate test holes for which more specific patterns are needed. We intend to deploy Fsimac for designing more effective CA-BIST.*

1 Introduction

This research originated from the testability study on RAPPID (Revolving Asynchronous Pentium®Processor Instruction Decoder), an asynchronous IA32 instruction length decoding and steering unit, developed at Intel to demonstrate the potential of aggressive self-timed techniques for microprocessor design [14, 15, 13]. The testability results were based on non-invasive Built-In Self Test (BIST) using Cellular Automata (CA). The CA-BIST solution was tuned to the RAPPID design, resulting in excellent stuck-at fault coverage, and enabling us to analyze undetected and untestable faults. Our analysis showed that a majority of the escapees changed the transition delays in the domino gates, which for approximately 5% of the total stuck-at faults could lead to catastrophic behavior [13]. The fault simulator Fsimac, reported here, focuses on both stuck-at and gate-delay faults to fill this gap.

A basic technique to simulate sequential circuits is to unfold the circuit either in time or in space, and simulate the unfolded circuit. We use time-unfolding and assume a *fundamental mode of operation* [16, 1] where signals at the primary inputs (PIs) are allowed to change only after the signals at the primary outputs (POs) have stabilized. For synchronous sequential circuits, the time-frame boundaries typically coincide with the clock boundaries. This is different for asynchronous sequential circuits, where we need to identify the boundaries separately so that we can use the fundamental mode of operation on a frame-by-frame basis until the PO signals stabilize. Identification of the frame boundaries essentially involves identifying feedback loops.

Our *feedback identification* algorithm differs from traditional methods used for instance by [7, 10, 9] for partial scan selection, in that we take a breadth-first rather than a depth-first search approach. The set of feedback cuts is generally larger than the traditional minimum feedback vertex set solutions but, as a pro, results in a lower (typically minimal) number of iteration cycles per frame — which can lead to significant reductions in simulation time.

The core of the fault simulator lies in timing analysis of the frames, which are essentially combinational circuits. Because our target circuits are those of RAPPID, which in some sense resemble extended burst mode machines, we decided to follow the timing simulation approach developed for 3D-style extended burst mode machines [4, 3], using *min-max timing analysis* and *13-valued logic*.

The organization of the paper is as follows. Section 2 gives some basic background on delay and event modeling, and Section 3 shows how complex domino gates are incorporated. Section 4 contains our feedback identification algorithm. Fault coverage and simulation details for Fsimac are given in Section 5, and conclusions in Section 6.

2 Timing models

From a gate-level perspective, the timing simulator in Fsimac operates on combinational circuits. Any remaining sequential logic is inside a gate, e.g. a latch or complex domino gate. Each gate has pre-specified delay bounds indicating the minimum and maximum input-to-output waveform delays, and a 13-valued waveform description that includes the input-to-output behavior regarding hazards. The gates are evaluated in topological order, as determined by the underlying directed graph structure. For all signals, we compute 13-valued logic values as well as *time stamps* for the earliest and latest arrival time, measured with respect to the input change at the PIs and feedbacks. We do this to capture the asynchronous nature of the circuit and to enable validation of relative timing constraints. Sections 2.1-2.2 give more background on delay and waveform models.

2.1 Delay Model

The following three types of delay models are commonly used in the design and analysis of asynchronous circuits:

- **Fixed delay**
  The delay of each component is assumed to be fixed. Due to variations in processing and operating conditions, IC component delays are seldom fixed — and as such, this is not a realistic delay model.
- **Unbounded delay**
  Component delays can take any non-negative value. Asynchronous design methods based on this model are extremely robust to delay variations. In general, though, this model is over-conservative.
- **Bounded delay**
  The delay of each component varies between given lower and upper bounds. This model intends to capture delay variations due to fluctuations in the fabrication process, ambient temperature, power supply, etcetera.

Asynchronous design methods, in particular those for self-timed circuits, make frequent use of the unbounded delay model in order to manage the design complexity and to support a wide range of implementation technologies without the need for re-design [2]. RAPPID makes use of handshake protocols for exactly these reasons, but adds local and relative timing information to enable a smaller, faster and lower power implementation in a given technology [15]. This timing information is made explicit for purposes of verification and re-synthesis or re-design. The measured performance of RAPPID correlates well with the simulation numbers of COSMOS that are based on fixed delays [14], which indicates that fixed delay modeling works for RAPPID as a system performance metric. At the gate level, though, the bounded delay model is the most likely fit for simulating the circuit with regard to relative timing constraints, and for tackling the 5% test coverage gap observed in [13].

Fsimac follows the *min-max timing analysis* approach by [4, 3] which is based on the bounded delay model. We assume that the gate delay bounds are known in advance for a given circuit and process technology. In addition, we assume equal nominal gate delays for rising and falling transitions, and zero wire delays — but these assumptions can be changed without jeopardizing the simulation model. To further simplify the analysis, we conservatively assume that the component delays are uncorrelated.

In other words, our focus is on *lumped gate delays*. We mainly consider *pure delays*, for which the waveforms are shifted in time and do not change shape. However, we model some forms of *inertial delay*, for instance in complex domino gates, where narrow evaluation pulses are suppressed at the output side (see Section 3).

2.2 Waveform Model

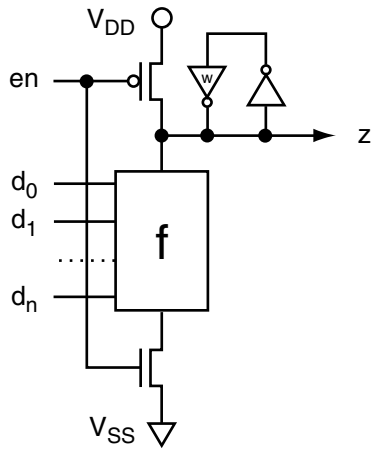
13-Valued waveform logic, originally proposed by [5] makes it possible to deal with hazards during circuit analysis, and to avoid unnecessary event proliferations by abstracting the details of multi-transition waveforms. We adopt the interpretation and notation of [3] and represent signal waveforms as triples $\langle b, m, e\rangle$, with b denoting the begin state of the signal, e the end state, and m the intermediate transition behavior. More precisely:

- $b, e \in \{0, 1, X\}$, where X indicates an unknown signal value which can be either $0, 1$, or change repeatedly.
- $m \in \{0, 1, \uparrow, \downarrow, X\}$, where $\uparrow, \downarrow, X$ indicate a single rising, respectively, falling transition, and potentially multiple transitions.
- **13-valued waveforms**

(constant)	$\langle 1,1,1\rangle$, $\langle 0,0,0\rangle$
(transition)	$\langle 0,\uparrow,1\rangle$, $\langle 1,\downarrow,0\rangle$
(hazard)	$\langle 0,X,0\rangle$, $\langle 0,X,1\rangle$, $\langle 1,X,0\rangle$, $\langle 1,X,1\rangle$
(stabilizing)	$\langle X,X,0\rangle$, $\langle X,X,1\rangle$
(unstabilizing)	$\langle 0,X,X\rangle$, $\langle 1,X,X\rangle$
(undefined)	$\langle X,X,X\rangle$

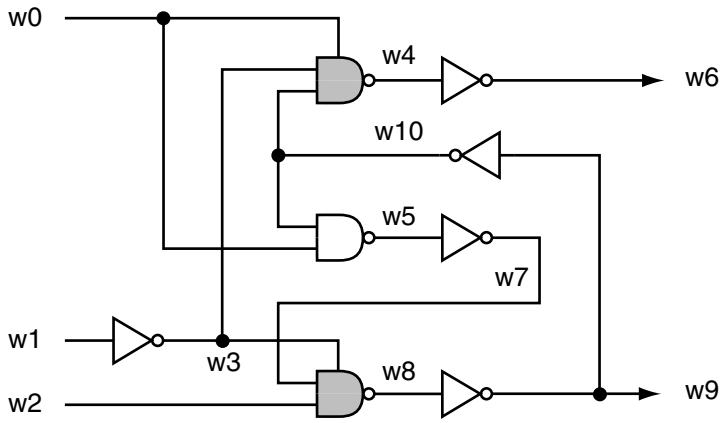
Functions $f(d_0, d_1, \ldots, d_n) : \{0,1,X\}^n \rightarrow \{0,1,X\}$ can be extended to this 13-valued logic domain, assuming $d_i = \langle b_i, m_i, e_i\rangle$ and output result $\langle b_f, m_f, e_f\rangle$, as follows:

- Take all sequences of input waveforms, and compute the corresponding sequences of (single input change) states from $\langle b_0, b_1, \ldots, b_n\rangle$ to $\langle e_0, e_1, \ldots, e_n\rangle$.
- Define $b_z = f(b_0, b_1, \ldots, b_n)$, $e_z = f(e_0, e_1, \ldots, e_n)$.
- For the definition of m_f, we need to monitor f throughout the state sequences. If the value of f stays constant $0\,(1)$, then $m_f = 0\,(1)$. If, on the other hand, f changes from 0 to 1 (or from 1 to 0) exactly once per state sequence, then $m_f = \uparrow\,(\downarrow)$. Otherwise, $m_f = X$.



en	f	z
$111, 0\uparrow 1, 1X1, 0X1, XX1$	$000, 1\downarrow 0, 0X0$	previous z
	$0XX, XX0, XXX$	XXX
	$0\uparrow 1, 0X1, 1X1, XX1, 111, 1X0, 1XX$	$000, 1\downarrow 0$
$000, 1\downarrow 0, 0X0, 1X0, XX0$	any 13-value	$111, 0\uparrow 1$
$XXX, 1XX, 0XX$	any 13-value	XXX

Figure 1 Generic footed domino gate with full-keeper (left) and part of the look-up table with 13-valued logic behavior (right).



File dbyc.v

```
module dbyc(w0,w1,w2,w6,w9);
input w0,w1,w2;
output w6,w9;
wire w3,w4,w5,w7,w8;
   cdom CDOM3_0(w4,w3,w10,w0);
   nand NAND2_0(w5,w10,w0);
   inv INV1_0(w3,w1);
   inv INV1_1(w6,w4);
   inv INV1_2(w7,w5);
   inv INV1_3(w10,w9);
   cdom CDOM3_1(w8,w7,w2,w3);
   inv INV1_4(w9,w8);
endmodule
```

File dbyc.des

```
CDOM3_0   01*;  /* domino AND */
CDOM3_1   01*;  /* domino AND */
```

Figure 2 Asynchronous circuit *dbyc* from RAPPID with complex domino gates in grey (left) and Fsimac description (right).

To extend a Boolean function, just take its 3-valued logic expansion for $\{0, 1, X\}$ and follow the above procedure. For efficiency, all standard 13-valued functions are pre-computed and kept in a look-up table.

3 Complex Domino Logic

Domino logic uses a single control signal to precharge and evaluate a cascaded set of dynamic logic blocks. The control signal can be a clock, as is typically the case in synchronous design, or a local signal triggered by a self-timed protocol as in [14, 15, 8]. Domino logic is used frequently in high-speed circuit design.

The generic CMOS domino gate shown in Figure 1 has a full-keeper (latch) to maintain the logic value at the output z, when the control signal en is high. The en-controlled n-transistor, called *foot*, ensures mutual exclusion between the precharge (high) and evaluate (low) phases. Part of the look-up table with the 13-valued logic behavior is shown on the right. Row 1 gives the table inputs en, f and the table output z, rows 2–4 describe the behaviors for which we consider the gate to be in evaluation phase, row 5 does the same for the precharge phase, and row 6 addresses the undefined hazardous situation where we cannot tell which phase applies. Note the output suppression of the narrow $(f \wedge en)$ evaluation pulses for $f = 1\downarrow 0,\ 0X0$ in row 1.

Figure 2 shows the circuit schematics *dbyc* from the Byte Controller in RAPPID [14], with the two Fsimac file formats **dbyc.v** and **dbyc.des**. File **dbyc.v** contains the structural Verilog description of the circuit. File **dbyc.des** describes the functionalities of the two domino AND gates with the Boolean functions f given in reverse Polish notation: $01*$, using numeric input names i for d_i in Figure 1.

4 Feedback Identification

Our algorithm for feedback identification, *feedback_detect*, uses a breadth-first traversal of the directed graph $G=(V,A)$ that we generate from the gate-level circuit description in structural Verilog. Vertex set V contains the circuit gates and PIs. Arc set A contains the (directed) wire segments connecting two vertices. For each vertex g, we use A to derive predecessor vertices $fanin(g)$ and successor vertices $fanout(g)$. For each g, we compute an integer variable called *level* to denote the maximum number of gates between the PIs and g, and we keep a Boolean variable called *flag* to indicate that the *level* computation for g has completed.

Two traversal lists, called TRUE_LIST and FALSE_LIST, keep track of the vertices for which *flag* is true and false respectively. Both lists are ordered by *level*. For efficiency, the ordering is descending for TRUE_LIST, and ascending for FALSE_LIST.

Feedbacks are identified and stored on-the-fly in a separate list FEED whenever g has a *level* greater than one of its fan-out gates f, and both levels are partially evaluated. We include such f by ascending *level* in a third traversal list, called EVAL_FEED_LIST.

Each traversal phase starts from TRUE_LIST, proceeds with FALSE_LIST, where the actual feedback identification takes place, and ends with EVAL_FEED_LIST. This continues phase-after-phase until all *flags* are true. Figure 3 shows the resulting feedback list for an example circuit.

Algorithm *feedback_detect* has a linear time complexity, and provides a solution which typically leads to a minimal number of iterations when the circuit is simulated in fundamental mode, i.e. from PI changes until PO stabilization.

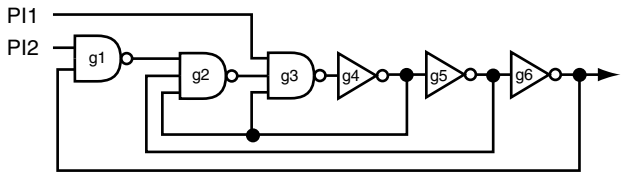


Figure 3 (feedback_detect example) After the first **repeat** phase, FALSE_LIST = EVAL_FEED_LIST = $\langle\,\rangle$, TRUE_LIST = $\langle g1, g3\rangle$, the *flags* are *true* only for $g1, g3$ and the *levels* for $g0 \ldots g6$ are $1, 4, 1, 2, 3, 4$. After the second **repeat** phase, the *level* values are unchanged but all *flags* are now *true*, and the algorithm terminates with FEED = $\langle\,(g2 \rightarrow g3), (g4 \rightarrow g3), (g6 \rightarrow g1)\,\rangle$.

Algorithm feedback_detect

```
/* Input                                                              */
/*      Directed graph G = (V, A)                                     */
/* Output                                                             */
/*      List of feedbacks FEED ⊆ (V → V)                              */
/*      level(g) for g ∈ V                                            */
/* Initially                                                          */
/*      level(g) = 0 and eval(g) = false, for all g ∈ V               */
/*      flag(g) = true, for primary inputs g ∈ V ∩ PI                 */
/*      flag(g) = false for gates g ∈ V − PI                          */
/*      TRUE_LIST = FALSE_LIST = ⟨ ⟩                                  */
/*      EVAL_FEED_LIST = FEED = ⟨ ⟩                                   */

Procedure comp(g);
/* computes level(g) and flag(g), for g ∈ V */
begin
  level(g) :=
    max {level(f)+1 | f ∈ fanin(g) ∧ (g → f) ∉ FEED} ∪ {1}
  flag(g) :=
    ⋀ {flag(f) | f ∈ fanin(g) ∧ (g → f) ∉ FEED} ∪ {true}

  if flag(g) then
    add g to TRUE_LIST by descending level
    delete g from FALSE_LIST and EVAL_FEED_LIST
  else
    add g to FALSE_LIST by ascending level
    eval(g) := true
  fi
end

/* main program */
forall g ∈ V ∩ PI do
  forall f ∈ fanout(g) do comp(f) od
od
repeat
  while TRUE_LIST ≠ ⟨ ⟩ do
    g := head(TRUE_LIST)
    TRUE_LIST := tail(TRUE_LIST)
  forall f ∈ fanout(g) do
    if (g → f) ∉ FEED then comp(f) fi
  od
  while FALSE_LIST ≠ ⟨ ⟩ do
    g := head(FALSE_LIST)
    FALSE_LIST := tail(FALSE_LIST)
    forall f ∈ fanout(g) do
      if (level(g) > level(f) ∧ eval(f) ∧ (g → f) ∉ FEED)
      then
        add (g → f) to FEED
        add f to EVAL_FEED_LIST by ascending level
      else comp(f)
      fi
    od
  od
  while EVAL_FEED_LIST ≠ ⟨ ⟩ do
    g := head(EVAL_FEED_LIST)
    EVAL_FEED_LIST := tail(EVAL_FEED_LIST)
    comp(g);
  od
until flag(g) = true, for all g ∈ V
```

Circuit Characteristics						Stuck-at Fault Simulation			Transition Fault Simulation		
Circuit	**Gates**	**PIs**	**POs**	**Internal Wires**	**Feedbacks**	**Faults**	**Tests**	**Coverage %**	**Faults**	**Tests**	**Coverage %**
Exs3d1	51	1	7	37	25	132	15	71 .97	51	4	86.27
							30	83.33		6	88.23
							50	99.24		12	92.16
							75	99.24		15	92.16
Exs3d2	57	7	3	51	8	222	15	42.79	57	15	59.69
							30	56.76		30	59.69
							75	78.83		50	77.19
							125	84.68		60	80.12
Exs3d3	19	1	3	13	6	46	4	67.39	19	15	78.94
							7	91.30		25	89.47
							13	100.0		50	89.47
										60	94.73
Exs3d4	10	3	2	2	2	26	3	84.0	10	5	70.0
							5	100.0		9	80.0
							15	100.0		15	80.0
MUX	7	3	2	2	2	26	3	73.08	7	5	71.42
							5	80.77		33	85.71
							15	100.0		46	100.0
dbyc	8	3	2	5	1	24	3	62.5			
							7	70.83			
							9	75.0	work in progress		
							11	83.33			

Table 1 Fsimac fault coverage for single stuck-at and gate-delay faults on circuits from 3D, Tangram and RAPPID.

5 Fault simulation

The inputs for Fsimac are (1) a gate-level circuit description in structural Verilog, (2) minimum and maximum gate delay bounds, and (3) a sequence of test stimuli. We first simulate the fault-free circuit, and then the faulty circuits with a single fault each, and we compare the simulation results between the latter and the first. When no fault list is specified, we take all single stuck-at and gate-delay faults. We use the standard stuck-at 0/1 fault model on gate inputs and outputs. A gate-delay fault can either shrink or grow the minimum and maximum input-to-output waveform delays for the gate.

The simulations are done frame-after-frame, where the frame boundaries are the feedbacks identified by algorithm *feedback_detect* given in Section 4. To initialize the circuit, the designer can specify a set of initial signal values. The unspecified PIs and inputs from feedbacks are set to 0 and the circuit is simulated to obtain stable logic values for the initially unspecified feedback wires. When initialized, we simulate the current frame, using min-max timing analysis and using the first test stimuli as PI values. When the PO results become stable, we take the final output value of fanin gate g for every feedback $(g \rightarrow f)$ as the next input value for the corresponding fanout gate f, and set the PI values to the next test stimuli. Then we simulate the next frame, etcetera — until either the tests are exhausted or all faults detected.

For stuck-at faults, we compare the logic PO values for the circuit with the fault against the logic PO values for the fault-free design. We do this at the end of each simulated time frame. If the results do not match, the fault is detected. For a gate-delay fault, we additionally check the time stamps for the PO signals at the end of each time frame: i.e., we (1) compare the signal orderings by minimum delay bound, and (2) verify if the minimum delay bound is lower than the corresponding maximum bound for the fault-free simulation. If either one fails, the fault is considered detected.

We benchmarked Fsimac on 3D circuits [18], on Philips handshake circuits from Tangram [17, 12], and on Intel circuits from RAPPID [14, 15]. The test sequences were generated pseudo-randomly using Cellular Automata (CA) [6]. Table 1 gives results for 3D circuits Exs3d1. . .Exs3d4, Tangram circuit MUX, and RAPPID circuit dbyc. We injected all single stuck-at faults, and single gate-delay faults with a fixed increase (beyond comfort level) in both delay bounds. The results are largely validated by hand. Note that a significant increase in the number of tests does not always lead to a significant increase in fault coverage. This can mean that either a significant portion of the uncovered faults are untestable, or more specific test patterns are needed. Fsimac cannot do much about the former situation, but we have reason to believe that Fsimac can help with the latter, and coach the CA into generating more design specific tests.

6 Conclusion

Fsimac is a gate-level fault simulator for asynchronous sequential circuits that use relative timing in addition to delay-insensitive design. We support min-max timing analysis, using bounded gate delays and 13-valued waveform logic for combinational as well as sequential gates, such as Muller C-elements and complex domino gates. This analysis is performed on the time-unfolded circuit, after cutting feedback loops. Our feedback identification algorithm is new, and differs from existing approaches in that it uses breadth-first rather than depth-first search. The corresponding feedback cuts result in a lower (typically minimal) number of iteration cycles during timing analysis.

The fault simulator handles single stuck-at 0/1 faults on gate inputs and outputs, and gate-delay faults that either shrink or grow the minimum and maximum input-to-output waveform delays of a gate. We validated the coverage results for Fsimac on benchmark circuits from 3D, Tangram (Philips), and RAPPID (Intel).

There is ample room for improvement in the capacity and speed of the fault simulation procedure. Capacity and speed were not our inital focus, which was (1) being able to fault simulate aggressive self-timed circuits like RAPPID, and (2) capture coverage gaps dropped by existing fault simulators based on fixed delay models. The latter two focus points are orthogonal to the former efficiency issues. Therefore, it should be relatively easy to adopt existing solutions for reducing memory requirements and run times [1, 11], and incorporate them in Fsimac.

Our ultimate goal is to use this fault simulator for designing more efficient Built-in Self Test (BIST) based on Cellular Automata (CA). In RAPPID, we tuned the CA-BIST solution by hand to better fit the design needs. In future, we would like to use Fsimac to do the tuning automatically.

Acknowledgements Our special thanks go to Supratik Chakraborty, whose work on min-max timing analysis was used as starting point for Fsimac. We thank the students Kaushik Patra, C.V. Krishna, Rajatish Mukherjee, Souvk Ghosh, Ananda Sarkar, Shameek Ghosh, and Chitta Haty for their efforts in implementing this fault simulator.

References

[1] M. Abramovici, M. Breuer, and A. Friedman. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[2] G. Birtwistle and A. Davis, editors. *Asynchronous Digital Circuit Design*, Workshops in Computing. Springer-Verlag, 1995.

[3] S. Chakraborty, D. Dill, and K. Yun. Min-max timing analysis and an application to asynchronous circuits. *Proceedings of the IEEE*, 87(2):332–346, Feb. 1999.

[4] S. Chakraborty, D. Dill, K. Yun, and K.-Y. Chang. Timing analysis for extended burst-mode circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 101–111. IEEE Computer Society Press, 1997.

[5] T. Chakraborty, V. Agrawal, and M. Bushnell. Delay fault models and test generation for random logic sequential circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 165–172, 1992.

[6] P. Chaudhuri, D. Chowdhury, S. Nandi, and S. Chattopadhyay. *Additive Cellular Automata: Theory and Applications - Volume I*. IEEE Computer Society Press, 1997.

[7] K. Cheng and V. Agrawal. A partial scan method for sequential circuits with feedback. *IEEE Transactions on Computers*, C-39(4):544–548, April 1990.

[8] W. Hwang, G. Gristede, P. Sanda, S. Wang, and D. Heidel. Implementation of a Self-Resetting CMOS 64-Bit Parallel Adder with Enhanced Testability. *IEEE Journal of Solid-State Circuits*, 34(8):1108–1117, Aug. 1999.

[9] M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Saldanha, and A. Taubin. Partial-scan delay fault testing of asynchronous circuits. *IEEE Transactions on Computer-Aided Design*, 17(11):1184–1199, Nov. 1998.

[10] D. Lee and S. Reddy. On determining scan flip-flops in partial-scan designs. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 322–325, Nov. 1990.

[11] T. Niermann, W.-T. Cheng, and J. Patel. PROOFS: A fast, memory-efficient sequential circuit fault simulator. *IEEE Transactions on Computer-Aided Design*, 11(2):198–207, Feb. 1992.

[12] M. Roncken and E. Bruls. Test quality of asynchronous circuits: A defect-oriented evaluation. In *Proc. International Test Conference*, pages 205–214, Oct. 1996.

[13] M. Roncken, K. Stevens, R. Pendurkar, S. Rotem, and P. Chaudhuri. CA-BIST for asynchronous circuits: A case study on the RAPPID asynchronous instruction length decoder. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 62–72, 2000.

[14] 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 *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 60–70, 1999.

[15] K. Stevens, R. Ginosar, and S. Rotem. Relative Timing. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 208–218, 1999.

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

[17] 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 of Computers*, 11(2):22–32, Summer 1994.

[18] K. Y. Yun and D. Dill. Automatic synthesis of 3D asynchronous state machines. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 576–580, 1992.