An Empirical Study of On-Chip Parallelism*

Mary L. Bailey Lawrence Snyder

Department of Computer Science
University of Washington
Seattle, WA 98195

Abstract

This paper presents a methodology for empirically determining the amount of parallelism on a CMOS VLSI chip. Six chips are measured, and the effect of input choice and circuit size is studied. The unexpectedly low parallelism measured here suggests that certain strategies for parallel simulators may be doomed, and earlier efforts to extrapolate parallelism from small circuits to large circuits may have been overly optimistic.

1 Introduction

Simulation has continued to be a bottleneck in circuit design; parallel simulation is a potential solution to this problem. One oft-proposed strategy for parallel simulation is to partition the circuit among many processors, with each simulating its subcircuit [Smi86]. In particular, for switch-level simulation, circuits are often partitioned by transistor groups [Arn85, DB85]. This circuit partitioning strategy assumes that there is sufficient "activity" on the chip to keep all of the processors busy.

However, several instances of chip-level simulators designed using this strategy have not performed up to expectations. For example, Frank estimated a speedup of only 12 with a 64-processor simulator engine, assuming no bus contention [Fra85]. Arnold obtained a speedup of 3.8 for a 6 processor version of PRSIM running on MSPLICE [Arn85], and noticed that on average, each processor was idle 29% of the time. Are these problems due to poor partitioning strategies, or is there a more fundamental problem?

This paper addresses the question: How much parallel "activity" is there on CMOS VLSI chips? We begin by presenting a methodology for measuring the "activity" or parallelism on CMOS chips. We use this methodology to measure the parallelism of 6 circuits. We then discuss the effect of circuit size on parallelism. Finally, we draw our conclusions and discuss future work.

*This research is supported by the Defense Advanced Research Projects Agency, ARPA Contract Number MDA903-85-K-0072

2 Methodology

Before measuring circuit parallelism, we must provide a clear definition for it. Parallelism, as used in computer science, is a digital measurement, while the circuits necessarily have measurable analog properties. For our purposes, circuit parallelism is the average number of transistors switching at a given time. At this point we leave vague the quantization of time, but will return to it later in this section.

There are several ways to measure chip parallelism. Perhaps the most accurate method is to use a SEM probe. However, this requires many probes, one for each node on the chip. One could use a small number of probes and repeat the test until all nodes are measured, but this is impractical. Therefore, direct measurement is not possible.

An alternative, indirect method for measuring a circuit's parallelism would measure the chip's power requirements. Since static CMOS circuits derive power completely from transistor switching, the average power should correlate quite well with the amount of parallelism on the chip. However, because power dissipation depends on capacitive loads, pads and big drivers present in the circuit distort the power measurement. Eliminating these factors is impractical. Thus, external measurements are not viable for measuring circuit parallelism.

A third alternative is to use circuit simulation. Circuit simulation is a long trusted method for gathering information about the behavior of circuits. SPICE [Nag75] is recognized as the best indicator of circuit performance, but it can only be used on small circuits. Since we need to consider larger circuits for our measurements, we eliminated SPICE as a candidate for our circuit simulator, but use it to calibrate the selected simulator.

We chose to use RNL [Ter83], a linear level simulator that can handle circuits that are quite large. RNL is similar to a switch-level simulator, but includes timing information as a part of the simulation output. Given this choice, we considered two metrics for measuring parallelism. The first metric is the average number of events in the input queue at each timestep. The transistors in the queue are those that are changing. Thus, the average queue length is the average number of transistors that are changing. This is similar to the power measurements, but does not suffer from

the distortion of driver sizing, and thus may be the most intuitive measure of parallelism.

The second, alternative, metric is the average number of events executed in a timestep. This defines parallelism as the number of transistors that cross threshold values at a given time. This definition is commonly used when one wishes to measure the potential success of parallel simulators, since the events occurring in a single timestep can be executed in parallel by multiple processors.

Because this investigation was originally driven by the parallel simulation perspective, we use this second metric. In fact, to make the results more appropriate for event-driven simulators, we ignore timesteps where no events occur when computing this average parallelism.

Two issues remain in using this definition of parallelism. First, we must "calibrate" RNL. We do this by comparing it to SPICE. Second, we discuss how the parallelism measurements are affected by the choice of timestep.

2.1 Calibrating RNL

Since SPICE is the preferred simulator for measuring parallelism, but fails to handle large circuits, it is important to compare RNL and SPICE on small circuits to see if they reveal the same information. While we have used RNL at the University of Washington for five years with good success, RNL does not have the universal acceptance of SPICE.

The first issue in this calibration process is to determine exactly how the measurements will be taken, and which numbers will be compared. For RNL, an event occurs for one of two reasons: a node is changing between stable states ($0 \rightarrow 1$ or $1 \rightarrow 0$), or a node is changing due to charge-sharing. Charge-sharing may cause nodes to make transitions between stable states and the X state. Node changes from stable states to the X state occur with no delay. However, there may be a delay for the transition from the X state to the stable states. Also, RNL computes realistic delays only for nodes that it considers "interesting". These are all nodes *except* those that connect exactly two transistors in a chain. For the calibration experiments, we trace only the "interesting" nodes, and track events that give rise to stable states.

While RNL has a digital output, SPICE outputs a voltage which varies between 0 and 5 Volts. We first must determine how to "digitize" the SPICE output so that we can easily compare it to the output of RNL. In fact, the important issue is to determine when a signal makes a transition. We used a $4V$ threshold for rising signals, and a $1V$ threshold for falling signals. Thus a rising signal will have a value of 0 until its voltage rises above $4V$ at which time its value will become 1. These thresholds were selected because they provided the best timing correlation with the RNL output. Also, for the SPICE runs, we used a $0.1ns$ time increment in the transient analysis since the internal timebase in RNL is $0.1ns$.

We compared SPICE and RNL on three small circuits. We will discuss one of these, a 2 to 4 decoder, here. The others are discussed in [BS87]. The decoder has 32 transistors and 20 nodes. It is small, and is similar to a larger circuit used in later parallelism measurements. Thus, it was a good candidate for the calibration analysis. Of the 20 nodes in the circuit, 14 were "interesting." The inputs were initially set at 5 Volts and then simultaneously pulled down to 0 Volts. The circuit's reaction to this change ($5 \rightarrow 0$) was measured. In SPICE, each input was modeled as a pulse with $0ns$ rise and fall time. The results are shown in Table 1. The times listed are relative to the time when the select lines became 0.

Time (ns)	SPICE	RNL	Time (ns)	SPICE	RNL
0.0	s_1 = 0	s_1 = 0	1.5	o_4 = 0	
	s_2 = 0	s_2 =0	1.6	56 = 1	56 = 1
0.6	19 = 1	19 = 1	1.9		o_4 = 0
		20 = 1	2.6	o_1 = 1	
0.7	20 = 1		2.7	143 = 0	
	17 = 1	17 = 1	2.9		143 = 0
	18 = 1	18 = 1	3.2		o_1 = 1

Table 1: RNL vs. SPICE in the Decoder

There is a close correlation between RNL and SPICE for all of the signals except o_1 and o_4. These signals change much faster in the SPICE simulation. Each of these signals is the output of an inverter driven by either node 143 (o_1) or 56 (o_4). In RNL, the outputs cannot change before its input changes, thus the input changes first and the output changes a little later. In SPICE, however, the threshold of the n-channel transistors in the inverters have a lower threshold than $4.0V$ and the p-channel transistors have a higher threshold than $1.0V$, so the inverter starts driving its output signal before the input reaches its threshold. Since the input changes slowly compared to the output, the output reaches its threshold *before* the input does.

The other two small circuits, a small shift register and a modified full adder cell used in the Baugh-Wooley multiplier, yield similar results [BS87]. The results of SPICE and RNL are not identical, but are quite similar. Thus RNL provides a reasonable substitute for SPICE in these parallelism measurements.

2.2 Timestep Effects

One concern in using RNL for measuring parallelism is its $0.1ns$ timestep. Because we measure parallelism over a timestep instead of instantaneously, we need to understand the effects of the timestep on our parallelism metric. When considering the plausibility of creating parallel simulators, the $0.1ns$ timestep may be too small. This section presents the effect of changing RNL's timestep.

If larger timesteps give more parallelism, this could be useful for parallel simulators. However, expanding the size of the time step may affect the validity of the simulation. In particular, one must be careful to insure that an event and its predecessor, the event causing this one to occur, are *not* evaluated in the same timestep.

In order to examine this issue, we performed three types of experiments:

1. Retain the $0.1ns$ timebase, and measure the parallelism obtained by grouping the events into larger time slices. The results obtained here will have the same total number of events as in the $0.1ns$ timestep measurements, but may result in two events being evaluated at the same time when in fact one of them causes the other event.

2. Change RNL's timebase by changing the queuing delay (Δt, the number of $0.1ns$ units) of events. When an event is queued, if the new timebase is n times the old timebase, its Δt is changed to be:

$$\Delta t = \begin{cases} 0 & \text{if } \Delta t_{old} = 0 \\ n & \text{if } 0 < \Delta t_{old} < n \\ n(\Delta t_{old}/n) & \text{otherwise} \end{cases}$$

   where "/" is integer divide. This appropriately changes the timebase while guaranteeing that events dependent on each other will not occur in the same timestep.

3. Use the switch-level option in RNL to get parallelism data for a unit-delay time model. One would expect that the second experiment's data will approach the value obtained in this experiment as the timebase gets larger.

The circuit data we present here is from a 16×16 Baugh-Wooley multiplier, an instance of a generator developed at the University of Washington by Wayne Winder [Sys87]. It is a signed multiplier designed in static CMOS, and is purely combinatorial in nature. The test data consisted of 20 sets of random inputs, with each set consisting of a pair of inputs to initialize the circuit followed by a pair of inputs for the parallelism measurement.

The results for this experiment are shown in Figure 1, with 95% confidence intervals [BJ77]. The dashed confidence intervals are the average parallelism results for the "bucket" timebase measurements (experiment 1), and the solid line confidence intervals are the average parallelism results for the modified timebase measurements (experiment 2). The dotted lines at the top of the figure shows the confidence interval for the switch-level timebase (experiment 3).

The "bucket" experiments show parallelism increasing linearly as a function of timebase. The parallelism for the modified experiments increases also, but starts to level off around the switch level measurements. The curves differ due to the data dependency built into the modified experiments. In the modified timebase experiments the parallelism is ultimately limited by the data dependencies of the events, unlike the "bucket" experiment where the parallelism will continue to increase as the timebase increases.

For specific inputs, the parallelism in a given modified timebase can exceed the parallelism for the analogous switch-level simulation, and vice versa. For example, if three events occur in the order i, j, k, and i causes j, increasing the timestep can place i, j, and k in the same timestep. The modified timebase experiment will move j to the next timestep because it is caused by the earlier event, i, in the same timestep. This reverses the evaluation order of j and k, which may generate a different sequence of subsequent events. Thus different timesteps may result in more or fewer total events.

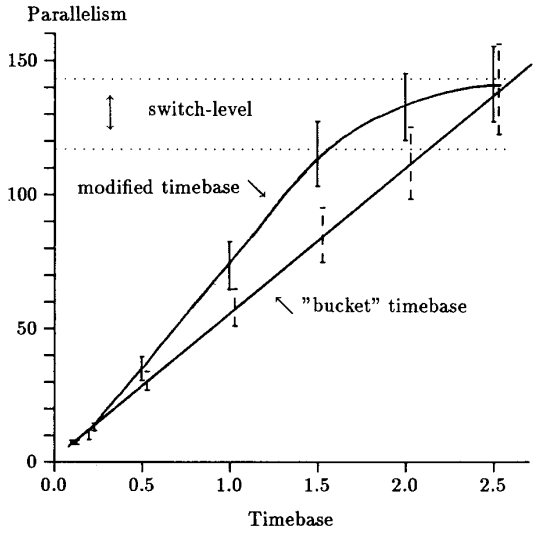

Figure 1: Effects of Timebase on Average Parallelism in the Baugh-Wooley Multiplier

Our measurements show that parallelism will increase as the timestep increases, and that the modified timebase measurements will converge to the parallelism measured by a unit-delay algorithm. Increasing the timebase does not dramatically change the measured parallelism, but may affect the accuracy of the simulation. In this paper, we use the more accurate $0.1ns$ timebase for our measurements.

3 Parallelism of Circuits

The above metric was applied to six CMOS circuits developed at the University of Washington. Two of the circuits have over 20,000 transistors; the others are much smaller. For all of the circuits, we used extracted circuits for the simulation so that the actual topology of the circuits would be reflected in the measurements. The results are shown in Table 2.

The Quarter Horse is a 32-bit RISC microprocessor [HJK*85]. It has 32 general purpose dual-ported registers, two internal busses, an ALU, shifter, memory address register, and a program counter structure with PLA control. In addition, it has an LSSD for testing purposes. As our test data we used a single run of a character load instruction, which takes 18 PLA cycles. The designers thought this instruction was highly parallel.

The IIR digital filter was designed by Hyong Lee [Lee85]. It includes a 16×16 multiplier, a 32-bit ripple adder, a 9-bit ripple counter, a 17 stage, 16-bit shift register, four 3 stage, 16-bit shift registers, and a PLA. Here we measured

Circuit	Transistors	Nodes	Parallelism (S.D.)						Percent Serial (S.D.)	
			Percent		Average		Maximum			
Quarter Horse, 32-bit RISC Microprocessor	24,068	10,500	0.06%		6.3		140		60%	
IIR Digital Filter	27,360	14,399	0.04%		6.4		280		53%	
8 × 8 Baugh-Wooley Multiplier	2,162	1,083	0.26%	(0.046)	2.8	(0.50)	22	(7.3)	38%	(8.6)
8 × 8 Booth Multiplier	2,013	1,088	0.31%	(0.031)	3.4	(0.34)	41	(12)	36%	(3.8)
8-stage, 16-bit Shift Register	1,536	1,048	2.4%	(0.23)	25	(2.4)	69	(6.0)	3.4%	(4.9)
4 to 16 Decoder	208	110	2.9%	(0.91)	3.2	(0.47)	11	(3.0)	42%	(9.6)

Table 2: Circuit Parallelism.

one macrocycle containing 401 microcycles.

The other three circuits are instances of generators, programs that produce families of circuits. All were developed at the University of Washington. The Baugh-Wooley multiplier is a 8 × 8 instance of the Baugh-Wooley multiplier generator used in the timestep measurements above.

The Booth multiplier is a generator developed as a group project in a VLSI design class. Its design is based on the modified Booth multiplier using the sign generate method described in [Ann86]. The multiplier has a static CMOS multiplier plane and clocked pipeline registers between the multiplier plane and the final adder. An additional carry resolve unit is placed at each row in the multiplier plane to compute the carry generated by unnecessary low order bits. The final carry is then used as the *carryin* to the final 18-bit adder. The final adder is a precharged Manchester carry adder with carry bypass.

The shift register is a CMOS generator developed by Smaragda Konstantinidou [Sys87]. It uses two-phase non-overlapping clocks. The shift register latch is a master-slave dynamic latch implemented with two clocked inverters.

The decoder is a static gate style CMOS generator written by Bill Yost [Sys87]. It is parameterized by the number of select lines.

For these instances, we averaged 20 sets of random inputs. Each set consisted of enough random inputs to initialize the circuit followed by a random input for the measurement.

For each circuit in Table 2 we measured the percentage of parallelism, average parallelism, maximum parallelism, and fraction of serial steps. The percentage of parallelism is the percentage of nodes that are changing, or the average parallelism divided by the number of nodes in the circuit. This allows easy comparison of the parallelism of different sized circuits. In this table the standard deviations are shown in parentheses, and calculated values are shown to two significant digits.

The resulting parallelism measurements are remarkably small. For instance, for the Quarter Horse, if we used 139 processors, and there was no overhead for synchronization and communication, the speedup would be only 6.3. The Digital filter faces a similar fate; the speedup is slightly larger, but at the cost of many more (430) processors. The surprise in these circuits is the amount of serial activity. Both show over 40% of serial activity, during which only one processor can be doing useful work. In fact, the only circuit with much parallelism is the shift register, which was selected for this reason. Even here, only 2.4% of the nodes are changing at any instance.

These results are comparable to those reported by Ed Frank [Fra85]. He found that the potential parallelism for the proposed Fast-1 processor ranged from 4.1 to 192.1 with a mean of 29.2. The measure he used was essentially the same as our definition: he took the total number of instructions executed by the single processor version of the Fast-1 and divided this by the number of parallel simulation steps. The number of parallel simulation steps is roughly equivalent to the number of distinct timesteps in RNL. The difference in the measured parallelism is explained by the fact that the Fast-1 is a unit-delay simulator, and in Section 2.2 we saw that the parallelism values for switch-level simulation can be up to an order of magnitude greater than the values we measure using a $0.1ns$ timebase.

4 Influence of Circuit Size

We now consider the effects of circuit size on parallelism using the two multipliers and the shift register shown in Table 2. For each circuit and size we used 20 random data sets and computed the mean and standard deviation for each circuit instance. Each data set consisted of random numbers generated by successively using the UNIX function *random* to obtain bit values, and combining these to form random numbers of the appropriate lengths. For each data set, we first supplied enough inputs to obtain an output, and then supplied an additional input for the parallelism measurement. In the case of purely combinatorial circuits this meant using one input set to initialize the circuit and a second set for the measurement.

Size		Parallelism (S.D.)						Percent	
$(n \times n)$	Transistors	Percent		Average		Maximum		Serial (S.D.)	
4×4	594	0.54%	(0.082)	1.6	(0.24)	6.8	(3.2)	64%	(12)
8×8	2,162	0.26%	(0.046)	2.8	(0.50)	22	(7.3)	38%	(8.6)
16×16	8,178	0.18%	(0.039)	7.2	(1.6)	79	(24)	14%	(4.0)
24×24	18,034	0.16%	(0.016)	14	(3.0)	190	(41)	8.6%	(2.7)
32×32	31,730	0.15%	(0.024)	24	(3.8)	320	(3.9)	5.7%	(1.2)

Table 3: Baugh-Wooley Multiplier: Random Inputs

Size		Parallelism (S.D.)						Percent	
$(n \times n)$	Transistors	Percent		Average		Maximum		Serial (S.D.)	
8×8	2,013	0.31%	(0.031)	3.4	(0.34)	41	(12)	36%	(3.8)
16×16	6,867	0.21%	(0.020)	7.8	(0.74)	110	(30)	20%	(2.2)
24×24	14,665	0.19%	(0.016)	14	(1.2)	230	(43)	14%	(1.8)
32×32	25,407	0.17%	(0.0076)	23	(1.0)	340	(46)	11%	(1.4)

Table 4: Booth Multiplier: Random Inputs

For each experiment we again measured the percentage of parallelism, average parallelism, maximum parallelism, and fraction of serial steps. As before, the standard deviations are shown in parentheses and calculated values are shown to two significant digits.

Each circuit family was represented by a group of circuits, parameterized here by input size. All of the circuits in the same family are similar in design and construction.

For the Baugh-Wooley multiplier, five instances were generated ranging from a 4×4 multiplier to a 32×32 multiplier. The data set consisted of two sets of x and y inputs, the first set for initializing the circuit, and the second set for measuring the parallelism. The results of these simulations are shown in Table 3. Note that even though the average parallelism increases as the size of the circuit increases, the percentage of parallelism decreases for the first three instances and then stays constant(statistically). Thus, one cannot predict the parallelism of the 32×32 multiplier by multiplying the percentage of parallelism of the 4×4 instance by the number of transistors in the larger instance.

For the Booth multiplier four instances were generated, ranging from 8×8 to 32×32. A 4×4 instance was not generated due to limitations in the generator software. For these experiments, each data set consisted of three pairs of x and y inputs. Two input sets were needed to initialize the circuit due to the presence of the pipeline between the multiplier plane and the adder. Table 4 shows the results of these simulations. The average parallelism is higher in all of the Booth instances than in the corresponding Baugh-Wooley instances with the exception of the 16×16 instance. However, the percentage of parallelism is higher in the Booth multiplier for *all* instances. As in the Baugh-Wooley multiplier, the parallelism of the larger instances cannot be accurately computed by extrapolating from the percentage of parallelism of the smaller instances.

Five instances of the shift register were tested. Three instances had 16 bits with varying stages, and three instances had 8 stages with different numbers of bits. Each data set for the shift register consisted of $s + 1$ data points, where s is the number of stages in the instance. The results of the simulations are shown in Table 5. For the 16-bit instances, we see that the increase in parallelism grows linearly with the number of stages. Analogously, parallelism for the 8-stage instances grows linearly with the number of bits. The parallelism per 1,000 transistors is not statistically different for any of the instances. Thus, for shift registers, using the parallelism per transistor for a small instance to estimate the parallelism for a larger circuit should provide a realistic result.

The results in Tables 3 through 5 reveal a problem with estimating parallelism as a linear function of small circuits. For instance, Wong and Franklin [WF87] report parallelism experiments using a gate/switch level simulator and circuits ranging from 650 to 8,000 transistors. They scaled the parallelism values for 100,000 components and obtained parallelism measurements (the average number of simultaneous events) ranging from 80 to 3,294. As predicted by our results, the largest circuit had the smallest scaled parallelism while the smallest circuit had the largest. Our results show that the percentage of parallelism is typically not constant

Size			Parallelism (S.D.)						Percent	
Bits	Stages	Transistors	Percent		Average		Maximum		Serial (S.D.)	
8	8	768	2.5%	(0.32)	13	(1.7)	36	(4.0)	8.1%	(7.1)
16	4	768	2.4%	(0.29)	13	(1.6)	36	(4.2)	1.9%	(4.0)
16	8	1,536	2.4%	(0.23)	25	(2.4)	69	(6.0)	3.4%	(4.9)
16	16	3,072	2.4%	(0.18)	50	(3.8)	130	(9.0)	2.6%	(4.9)
32	8	3,072	2.4%	(0.16)	50	(3.4)	130	(8.7)	0%	(0)

Table 5: Shift Register: Random Inputs

and usually decreases as the circuit size increases. Thus the large parallelism measurements reported in that study may be overly optimistic.

5 Conclusions and Future Work

We have described a methodology for measuring the parallelism of CMOS VLSI circuits. We selected one of the parallelism definitions, the one most suited to the parallel simulation problem, "calibrated" the simulation tool, and measured the parallelism of six circuits. Three of the circuits were further analyzed to see how circuit size affects the resulting parallelism.

Parallelism in these circuits is remarkably low. It is unlikely that the circuits are not representative, either by the choice of circuits or by virtue of the design methodology at the University of Washington, since our results are comparable to those found by Frank. We also show that the percentage of parallelism is not necessarily constant over circuit size, and in fact often decreases as the circuit size increases. Measurements taken on small circuits may not be reliable predictors of the parallelism performance of larger circuits.

If chips are not very parallel, the conventionally accepted method for speeding up simulations through parallelism may be doomed. Specifically, the method of partitioning a circuit among processors may not be appropriate for parallel switch-level simulation, except, possibly, for simulators using a very small number of processors. These results go a long way toward explaining the poor observed performance of parallel event-based simulators. Moreover the implication is that the results apply, to a lesser extent, to gate-level event-based simulations for the same reasons.

There are several areas for future investigation. We are exploring alternative definitions for parallelism. Additional chips, including chips from other sources, should be tested to validate our results.

Most importantly, we need to discover why chips are not parallel. Are there better design methodologies or specific circuit designs that produce chips with more parallelism? Are parallel chips indeed better performers? Intuitively more activity should imply more performance for a given design. Is this intuition correct?

Finally, we may be able to use parallelism to assist at the architectural design level. One of the problems with the two large chips we examined is that at the architectural level, there is only one functional block active at any time. Pipelining should help here, and thus raise the overall parallelism.

References

[Ann86] Marco Annaratone. *Digital CMOS Circuit Design*. Kluwer Academic Publishers, Norwell. Mass., 1986.

[Arn85] Jeffrey M. Arnold. *Parallel Simulation of Digital LSI Circuits*. Master's Thesis, Massachusetts Institute of Technology, 1985.

[BS87] Mary L. Bailey and Lawrence Snyder. *Measurement of On-Chip Parallelism in CMOS VLSI Circuits*. Technical Report TR87-11-03, University of Washington Department of Computer Science, 1987.

[BJ77] Gouri K. Bhattacharyya and Richard A. Johnson. *Statistical Concepts and Methods*. John Wiley and Sons, Inc., New York, 1977.

[DB85] William J. Dally and Randal E. Bryant. A Hardware Architecture for Switch-Level Simulation. In *IEEE Transactions on Computer-Aided Design*, vol. CAD-4, no. 3, pp. 239-250.

[Fra85] Edward H. Frank. *A Data-Driven Multiprocessor for Switch-Level Simulation of VLSI Circuits*. PhD Thesis, Carnegie-Mellon University, November 1985.

[HJK*85] S. Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, and C. Yang. The Quarter Horse: A Case Study in Rapid Prototyping of a 32-bit Microprocessor Chip. In *Proceedings of the International Conference on Computer Design: VLSI in Computers*, pp. 261-266. IEEE, 1985.

[Lee85] Hyong Lee. *A Variable Digital Filter Design in 3um CMOS*. Master's Thesis, University of Washington, 1985.

[Nag75] L. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, Technical Report UCB ERL-M250, Electronics Research Laboratory, University of California, Berkeley, 1975.

[Smi86] Robert J. Smith II. Fundamentals of Parallel Logic Simulation. In *Proceedings of the 23rd Design Automation Conference* , pp. 2-12. IEEE, June, 1986.

[Sys87] Northwest Laboratory For Integrated Systems. *VLSI Design Tools Reference Manual Release 3.1*. Technical Report TR87-02-01, University of Washington Department of Computer Science, 1987.

[Ter83] Christopher J. Terman. *Simulation Tools for Digital LSI Design*. PhD Thesis, Massachusetts Institute of Technology, September 1983.

[WF87] Ken Wong and Mark A. Franklin. Performance Analysis and Design of a Logic Simulation Machine. In *Proceedings of the 14th Annual International Symposium on Computer Architecture*, pp. 49-55. IEEE, 1987.