

Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering
Research

Computer Science and Engineering

Report Number: WUCS-85-09

1985-11-01

Statistics on Logic Simulation

K. F. Wong, Mark A. Franklin, Roger D. Chamberlain, and B. L. Shing

The high costs associated with logic simulation of large VLSI based systems have led to the need for new computer architectures tailored to the simulation task. Such architecture have the potential for significant speedups over standard software based logic simulators. Several commercial simulation engines have been produced to satisfy need in this area. To properly explore the space of alternative simulation architectures, data is required on the simulation process itself. This paper presents a framework for such data gathering activity by first examining possible sources of speedup in the logic simulation task, examining the sort of data needed in... [Read complete abstract on page 2.](#)

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research



Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

Recommended Citation

Wong, K. F.; Franklin, Mark A.; Chamberlain, Roger D.; and Shing, B. L., "Statistics on Logic Simulation" Report Number: WUCS-85-09 (1985). *All Computer Science and Engineering Research*. https://openscholarship.wustl.edu/cse_research/853

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

Statistics on Logic Simulation

K. F. Wong, Mark A. Franklin, Roger D. Chamberlain, and B. L. Shing

Complete Abstract:

The high costs associated with logic simulation of large VLSI based systems have led to the need for new computer architectures tailored to the simulation task. Such architecture have the potential for significant speedups over standard software based logic simulators. Several commercial simulation engines have been produced to satisfy need in this area. To properly explore the space of alternative simulation architectures, data is required on the simulation process itself. This paper presents a framework for such data gathering activity by first examining possible sources of speedup in the logic simulation task, examining the sort of data needed in the design of simulation engines, and then presenting such data. The data contained in the paper includes information on the subtask times found in standard discrete event simulation algorithms, event intensities, queue length distributions and simultaneous event distributions.

Statistics on Logic Simulation

**K. F. Wong, M. . Franklin, R. D. Chamberlain
and B. L. Shing**

WUCS-85-09

November 1985

**Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899**

This research has been sponsored in part by funding from NSF Grant DCR-8417709 and ONR Contract N00014-8D-C-0761.

STATISTICS ON LOGIC SIMULATION

K.F. Wong, M.A. Franklin, R.D. Chamberlain and B.L. Shing

Washington University
St. Louis, Missouri

TOPIC AREAS:

1. Simulation
2. Special Purpose Hardware For DA

ABSTRACT:

The high costs associated with logic simulation of large VLSI based systems have led to the need for new computer architectures tailored to the simulation task. Such architectures have the potential for significant speedups over standard software based logic simulators. Several commercial simulation engines have been produced to satisfy needs in this area. To properly explore the space of alternative simulation architectures, data is required on the simulation process itself. This paper presents a framework for such data gathering activity by first examining possible sources of speedup in the logic simulation task, examining the sort of data needed in the design of simulation engines, and then presenting such data. The data contained in the paper includes information on subtask times found in standard discrete event simulation algorithms, event intensities, queue length distributions and simultaneous event distributions.

Statistics on Logic Simulation^{*}

K.F. Wong, M.A. Franklin, R.D. Chamberlain and R.L. Shing

Center For Computer Systems Design
Washington University
St. Louis, Missouri

1. Introduction

The high costs associated with the detection and correction of design errors once a VLSI chip has been fabricated have led to an increased reliance on simulation techniques in the logic design process. Logic simulation is used extensively to initially verify logic correctness and subsequently to develop vectors for testing fabricated chips. As circuit complexity has grown, the time delays and costs of performing logic simulation on standard serial computers have grown until they can consume months of machine time [PFIS82].

These high costs have led to the development of a number of special purpose processors dedicated to logic simulation [DENN82, HOWA83, ZYCA83, VALI84, DAIS85, HEFF85, SILI85]. Such processors typically perform simulations at 10 to 1000 times the speed of standard general purpose computers. The techniques employed in achieving these speedups vary from microcode implementation of simulation algorithms to the development of special purpose logic and multiprocessors tailored to a simulation algorithm. In addition to the approaches found in commercial simulation engines, other possible simulation architectures have also been proposed [BLAN84, FRAN84, ASHO85, HAHN85].

In general, it has been difficult to effectively compare these alternative approaches. There are several reasons for this. First, commercial products in this area often have proprietary designs whose details are not publicly available. Second, developing reasonable performance models over a range of complex architectural alternatives is still more an art than a science. Third, basic data on the simulation process (e.g. event distributions) is difficult and time consuming to obtain, and has not been generally available in the open literature.

^{*} This research has been sponsored in part by funding from NSF Grant DCR-8417709 and ONR Contract N00014-8D-C-0761.

This paper is concerned principally with the third item above, that is obtaining and presenting data on the logic simulation process. Such data, relating mainly to event activity, event list statistics and time distributions is important in determining the effectiveness and sizing of various pipeline and multiprocessor design options, in addition to designing event list scheduling algorithms and hardware.

The section to follow reviews several simulation speedup techniques. In the course of this presentation, the sort of data necessary in evaluating such techniques and architecture alternatives is discussed. Section 3 describes *Lsim*, a UNIX/C* based discrete event logic simulation program which has extensive facilities for data collection, and which was used in the data collection process.

Section 4 presents the test case workload of five VLSI designs and the data collection methodology employed. Section 5 discusses the results of the data collection. The final section summarizes the paper.

2. Logic Simulation Speed-Up Techniques and Data Requirements

Numerous options are available to the system designer for accelerating logic simulations through the development of novel architectures and computing structures. These options break down into two broad approaches (see Table 1).

Functional Specialization	- Special event-list hardware - Special function evaluation hardware - Special hardware for net-list operations
Concurrency Exploitation	- Parallelism - Pipelining

Table 1: Logic Simulation Architectural Speed-up Techniques.

The first, **functional specialization**, refers to those techniques which take a component of a standard software based simulation algorithm, and decrease its execution time by placing it

* UNIX is a trademark of AT&T Bell Laboratories.

in hardware. In this approach the basic sequential nature of the original simulation algorithm may be maintained. For example, given the central role played by event list manipulation routines, it is possible to design special purpose hardware which will time order a list of events, and permit insertion and removal of events in essentially a single instruction time. Given data on the time associated with this task when executed as a software routine, the cost effectiveness of developing such special purpose hardware can be evaluated. Given data on event time distributions, the sizing of such special purpose hardware can be optimized.

The second approach, **concurrency exploitation**, refers to designing hardware structures where either the physical parallelism which exists in the circuit being simulated, or the algorithmic parallelism inherent in the simulation algorithm can be utilized to achieve increased execution speed. Physical parallelism can be exploited by partitioning the circuit being simulated and placing partitions on separate processors in a multiprocessor architecture. Effective partitioning, however, requires that a balance be achieved between computational (e.g. functional evaluation) and communication loads (e.g. passing state information between processors) across the available processors.

Algorithmic parallelism, in this context, refers to the ability to create a pipeline of operations related to the basic simulation algorithm. Some of the tasks associated with the pipe would be: removal of an event from the event list, obtaining state information from netlist, performing function evaluation, etc. Such pipelining typically applies at each simulation time instant when an event(s) is available, but would only be effective in providing speedup if multiple events were available at that time. Notice that the more processors available to exploit physical parallelism, the fewer events are available per processor, and thus the less effective the speed up associated with pipelining. Data related to event distributions is thus important in evaluating the tradeoff between these two approaches to parallelism.

The architecture alternatives associated with concurrency exploitation can be further clarified by classifying the space of design alternatives associated with implementation of the

standard event based logic simulation algorithm **. Three classification components common to a wide range of logic simulation architectures can be identified and used to develop a taxonomy (Table 2) of logic simulation architectures [FRAN84].

TIME CONTROL MECHANISMS		
TIME ADVANCE	Unit Increment	Event based Increment
TIME SYNCHRONIZATION	Global Clock	Local Clock
EVENT LIST ATTRIBUTES	Single List	Multiple List
EVENT/FUNCTION EVALUATION	Single Machine	Multiple Machines

Table 2: A Taxonomy of Logic Simulation Architectures

The first component in the taxonomy relates to the **time control methodology** employed and is divided into two parts. The **time advance mechanism**, is concerned with how the simulation clock is advanced during execution of the simulation algorithm. The **time synchronization mechanism** relates to how clocks on separate processors in a multiprocessor are synchronized. In the **unit time increment** approach the clock is advanced by a single uniform time step at each simulation cycle. This is done whether or not any events are present which must be evaluated on that cycle. While this eases the clock distribution problem it also introduces an overhead associated with processing clock times having no event activity. In the **event increment** approach the clock is advanced in a nonuniform manner depending on the future time when the next event is to take place. This eliminates the overhead associated with processing no activity clock times; however, it introduces extra processing associated with scheduling events and distributing time information. Data on the percentage of no-event times is needed to help evaluate this design decision.

The **time synchronization** component is of importance principally when using multiple processors architectures. A global clock scheme simplifies time synchronization since all processors execute in a lock step fashion executing events scheduled for the same time. However, as pointed out earlier, if only a few events are available at each time point, it is possible that a

** We do not consider here architectures derived from purely switch level based algorithms [DALL85].

only a small number of the available processors in the multiprocessor will be active at each time point. The use of a local clock scheme in which processors move ahead to future events at their own pace (subject to various precedence constraints) might allow significantly more events to be processed in parallel [CHAN81, JEFF83].

The second taxonomy component concerns **event list attributes**. The number of event lists can range from one to the number of circuit components. While a single event list is typically associated with single processor approaches, it may also be effective in multiple processor designs. This is especially true if event list maintenance and scheduling is moved into special purpose hardware. Having multiple lists, such as one associated with each processor in a multiprocessor, will reduce overhead associated with communication of events and also reduce event list lengths.

The third taxonomy component, **event/function evaluation**, deals with the number of processors (event/function evaluators) present. The design tradeoffs here relate to balancing speedups associated with increased computational parallelism with increases in communications overhead and system complexity. Related to this design decision are the problems of circuit partitioning, design of the interprocessor communications network, and design of the algorithmic pipelining structure. These decisions require data associated with event activity, various event distributions, and communications impact of various partitioning strategies. Table 3 indicates some simulation machine design decisions and certain data needed to aid in those decisions. The next section discusses a software based simulator used in the data collection process.

DESIGN DECISION	NEEDED DATA
Event list scheduling device design	Percentage of time on standard serial algorithm taken with event list maintenance and scheduling
Event list scheduling device design	Distribution of event list length
Number of multiple processors	Distribution of number of events during clock cycles when at least one event is present
Pipeline depth and design	Distribution of number of events during clock cycles when at least one event is present, normalized by the number of processors present
Unit versus event time increment	Percentage of clock times where there are events to be processed
Circuit partitioning strategies	Communications bandwidth required using various partitioning heuristics

Table 3: Required Data for Certain Design Decisions

3. Lsim: A Unix/C Based Logic Simulator

3.1. Basic Lsim Features

Data on the simulation process was gathered using the *lsim* gate-switch level logic simulator. *Lsim* is a UNIX/C simulator based and was designed to ease the task of collecting data on the simulation process [CHAM85, CHAM86]. It can simulate systems containing both the traditional TTL unidirectional type of logic gates, and bidirectional switches of the sort found in MOS circuits.

Lsim models circuits with signal values being represented by one of seven logical states:

STABLE STATES	TRANSIENT STATES
1 high	r rising
0 low	f falling
z high impedance	t transition to/from high impedance
x undefined	

To properly model circuits which include bidirectional gates, pass transistors, wired logic connections and tri-state outputs, a "strength" is associated with each signal in addition to its

logical signal. *Lsim* uses two strengths, strong and weak, corresponding to a high and low current drive capability. A strong signal is one that is connected directly to the power supply, ground, or through an active transistor to supply or ground. A weak signal is one that is connected to a voltage source through a resistance, such as a depletion mode pullup transistor.

Timing analysis is supported at three different levels, a unit delay model in which every gate is assumed to have a delay of one simulated time unit, a fixed delay model where gate delays are modeled by fixed low-to-high and high-to-low propagation times, and a variable delay model in which gates have variable delays specified by a maximum and a minimum value. In addition, enable and disable times (i.e. switching times for the setup and removal of a high impedance state on a component output) may also be specified. The data presented in this paper were obtained with the fixed delay model.

The seven logical states associated with signal lines are divided into two major types, stable states (1,0,z,x) and transient states (r,f,t). Stable states apply to all timing models. The "1" and "0" states are used to model high and low voltages respectively. The "z" state is used to model the high impedance output of components that have tri-state outputs. The "x" state is used when little is known about the voltage level of the signal. Transient states only apply to the variable delay model and are used to represent intermediate states during a transition between stable states. The "r" and "f" states are used during a transition from low to high, and from high to low respectively. The "t" state is used during a transition to or from a high impedance state.

There are several components supported by *Lsim* that differ from the normal unidirectional gate model that is common in gate level simulators. These components, the pass transistor and resistor are capable of propagating signals in two directions. Internally to *Lsim*, these components are handled by creating, in effect, two parallel unidirectional components that are connected back to back. This construction is hidden from the user, who simply refers to one terminal of the component as the input and the other terminal as the output. The algorithms

for processing bidirectional components and handling multiple strength signals follow those proposed by HAYE82. More details on *lsim* and examples of its use can be found in CHAM85 and CHAM86.

3.2. Data Collection Facilities

Lsim has features to collect information related to three basic items: events, timing of subtasks in the *lsim* program, and communications across user defined circuit partitions. An event refers to a discrete action performed by the simulator, such as the modification of the logical state of a component output, or the periodic display of signal states to the user. Each event has a time associated with it which indicates when during the simulation that event is to occur. Events are stored in an event queue which is used in event scheduling and (lowest time value) event retrieval. *Lsim* collects the following statistical data on events:

- the number of events associated with each component in the circuit
- the number of events in the event queue
- the times between events in the event queue

In addition to the data mentioned above, there is a provision for *lsim* to send out a record to a file each time event queue activity occurs. Each record contains fields indicating event type, current simulated time, scheduled time of the event, and whether an event insertion, removal or deletion occurred. The resulting data file can be analyzed using the *S* statistical analysis package.

The UNIX profiling utilities may be used to obtain data on the execution times associated with various tasks involved in simulation. The utilities provide information that tells the number of times that subroutines have been called as well as cpu times for the subroutines themselves. The subroutine calls can then be classified into a set of general tasks that comprise the simulation. The task classifications used are found in Table 4.

Lsim also provides facilities for collecting communications information across circuit partitions. That is, given a user defined partitioning of the circuit to be simulated, *lsim* will

collect data on the number of times state information is passed across partitions. This data, although not available at this time, will be of use in evaluating the speedup potential associated with exploiting physical parallelism, and the bandwidth requirements required of interconnection networks used in multiprocessor configurations.

event queue manipulation	insertion, retrieval, or deletion of events from the event queue
functional evaluation	determination of component output values given component input values
netlist operations	propagation of component output changes to the inputs of other components, in effect, searching the connectivity of the circuit
data collection output	time spent collecting and displaying data being collected
startup operations	time spent loading and initializing the simulation
other overhead	time that could not be easily classified as one of the other tasks

Table 4: Simulation Subtask Classification

4. The Benchmark and Data Collection Methodology

Data on the simulation process was obtained by applying test sets containing random test vectors to five circuits. The five circuits were: 1) a stop watch, 2) a priority queue, 3) an associative memory, 4) a Radiation Treatment Planning (RTP) chip, and 5) a crossbar switch. This section briefly describes these circuits and the methodology employed in collecting data on the simulation process.

The stop watch circuit determines the elapsed time between a start and a stop signal. The priority queue can be used as an event list manipulation device. It stores 48-bit records, each divided into four fields, and retrieves the record whose first field contains the smallest value. The associative memory functions like a normal random access memory as well as a memory in which records can be retrieved by content (i.e. those matching a specified pattern).

The RTP chip implements an algorithm used in cancer treatment planning, which calculates the radiation dosage at a specified point. The crossbar switch provides an interconnection network between four input and four output ports.

These circuits reflect a mix of characteristics (Table 5) and are the product of five graduate student design teams. The two most prevalent VLSI technologies (nmos and cmos) and clocking schemes (synchronous and asynchronous) are represented. The circuit sizes range from approximately 650 transistors to 6500 transistors. The priority queue, associative memory, and crossbar switch were designed so that they could be scaled to larger versions as required (assuming no pin or power limitations). The test circuits were kept small enough to insure that simulation run lengths were reasonable and disk storage availability was adequate. The Switches and Gates columns in Table 5 indicate the number of *lsim* bidirectional switch and unidirectional gate blocks used in defining the circuit (the Total entry is the sum of these columns). The right column reflects the total number of transistors in each circuit.

The test set size was picked to insure that the statistics gathered reflect the sort of test sets one might use during design verification. In each case, the size of the test set was chosen so that the aggregate statistics remained stable over at least five run intervals and most components experienced at least one output change. For example, if the test set contained 25 test vectors, statistics were displayed after every 5 test vectors. If the average queue lengths fluctuated more than one percent, the number of test vectors was increased to approximately 50 and the simulation was repeated. The run lengths were increased until the fluctuations

Circuit	Tech.*	Type*	Switches	Gates	Total	Approx. Trans.*
Stop watch	nmos	sync	216	131	347	650
Assoc. memory	nmos	async	296	454	750	1700
Priority queue	cmos	sync	2960	720	3680	5100
RTP chip	nmos	sync	1422	1746	3169	6100
Switch	nmos	async	0	2096	2096	6300
Average			979	1029	2008	3970

* Technology, synchronous, asynchronous, Approximate number of transistors
 Table 5: Circuit Characteristics.

stabilized to within a range of about one percent. Once this first criterion was met, the number of test vectors was increased further if less than 95 percent of the components experienced at least one output change.

The test vectors were applied to the test circuits using *lsim's* program interface. In this technique, special test vector generation subroutines were written in the C language and dynamically linked to the normal *lsim* routines. These routines supplied the inputs necessary to simulate a stream of random test vectors.

5. The Data

5.1. Subtask Time Distributions

The first data to be considered relates to the relative subtask times associated with the standard discrete event oriented logic simulation (Table 6). Time spent in the data output operation has not been included since this will vary greatly depending on the amount of data being collected. For example, if data is collected about every event that occurs during the simulation, this task alone could consume as much 40 percent of the execution time. Startup operation time is also omitted from this percentage data in normalizing for variations in test set and circuit size.

Circuit	Percentage Time			
	queue manipulation	functional evaluation	netlist operations	other overhead
Stop Watch	19	36	31	14
Assoc. Memory	12	38	39	11
Priority Queue	22	33	35	10
RTP Chip	39	25	20	16
CB Switch	25	35	20	20
Average	23	33	29	14

Table 6: Subtask Execution Percentages

From this data, the speedup that can result from various types of hardware specialization can be evaluated. The data indicates, however, that there is no single subtask which is a critical

bottleneck. That is, unless all aspects of the algorithm are improved, large speedups will not be achieved. For example, if an infinitely fast device were designed to process events, the most that could be gained is a speedup of about 23 percent. Note that a fairly efficient timing wheel based event list algorithm is used in *lsim* [ULRI78].

5.2. Event Intensity Data

To get a broad measure of simulation activity over time, it is worthwhile noting the fraction of time points during which no activity takes place. That is, given a resolution of say 1 nanosecond, this is the percentage of nanosecond time points when no events are scheduled. As shown in the first column of Table 7, at most of the time points there is no activity. Related to the idle time percentage is the idea of circuit intensity. Intensity corresponds to the percentage of gates which change state on average over the simulation. The second column of Table 7 shows the percentage of gates which change state averaged over all non-idle time points (i.e. points where at least one event occurs). The third column shows the percentage of gates which change state averaged over all time points (idle and non-idle) in the simulation.

Circuit	Idle Time	Intensity: Non-idle Time	Intensity: Total Time
Stop Watch	99	3.1	.027
Assoc. memory	89	.9	.102
Priority Queue	84	1.4	.224
RTP Chip	84	.55	.087
CB Switch	71	.04	.01
Average	85	1.2	.09

Table 7: Event Activities and Intensities (Percentages)

The general picture that emerges is that logic simulation is an activity where, during most of the simulation time points nothing is happening and, when there is activity, it involves a small fraction of the circuit being simulated. The conclusion is that for special purpose simulation architectures to be effective, they must take advantage of the localities of activity which occur in both time and space. Luckily, since we are interested in large circuits, small

percentages may still yield enough activity so that the speedup techniques of specialization and parallelism can be effective if they are applied at non-idle time points. Given this result, the queue length and event simultaneity statistics discussed in the next section, unless otherwise specified, are based on measurements taken at non-idle time points. Note that the CB Switch has the highest non-idle time yet has the lowest intensities. This is due in part to the testing protocol where the switch was loaded to only 50% capacity, and to the switch design (i.e. a circuit switched pipelined design). This results in many non-idle time points where there is very little activity at each point.

5.3. Event Queue Length Distribution

The length of the event queue will vary during the simulation. The distribution associated with queue length yields information on how long one can expect event lists to grow and this information is useful in designing efficient software and hardware based algorithms for event manipulation. For example, what should be the size of a hardware unit specialized to event queue function? If made too small, then overflow conditions will often arise with a likely associated time penalty. If made too large, such a device could be costly but yield little in added performance.

Table 8 summarizes benchmark queue length data. Figures 1 and 2 present density curves for two of the circuits. These show the fraction of time during the simulation corresponding to different queue lengths.

Circuit	Prob. Queue Length ≤ Table Entry			Average Over		Max. queue size
	.9	.95	.99	non-idle time	total time	
Stop Watch	48	79	122	18.9	3.6	143
Assoc. memory	32	34	63	9.2	3.3	375
Priority Queue	209	217	242	68.1	28.6	810
RTP Chip	67	123	164	29.9	11.4	225
CB Switch	5	6	7	2.9	2.3	13
Average	72	92	120	25.8	9.8	313

Table 8: Queue Length Statistics

Note that queue sizes are modest, with average sizes (over non-idle time points) being less than 30 entries, and the probability of queue length less than 90 being greater than .9 . This is shown in Figures 2 and 3 which give the distributions for the Priority Queue and Associative Memory Circuits. Both distributions show a sharp drop off after relatively small queue sizes. If one assumes that, over a variety of chip designs, simulation queue length varies directly as the number of transistors in the design, then the average numbers in Tables 5 and 8 can be used to obtain queue lengths to be expected for larger circuits. For instance, in logic simulations of 100,000 transistor circuits (25 times the average circuit of Table 5), 90% of the time the event queue would have a length less than about 1800. This will vary, of course, depending on the characteristics of the individual circuit being simulated.

5.4. Event Simultaneity

The data on intensities is further refined in Table 9 and Figures 3 and 4. Table 9 is concerned only with those time points during which one or more events are processed. In the first column, for example, there is a 90% probability that the number of events at a given non-idle time point will be less than the table entry. Figures 3 and 4 show the general fast dropoff in number of simultaneous events after the first few entries. They do, however, also demonstrate that there are apparently a few instances of intensive activity where many simultaneous events occur.

Circuit	Prob. # of Simultaneous Events ≤ Table Entry			Average Over		Max. # sim. events
	.9	.95	.99	non-idle time	total time	
Stop Watch	26	44	72	11	.1	82
Assoc. memory	23	32	35	7	.8	346
Priority Queue	194	200	217	55	8.6	802
RTP Chip	38	121	142	18	2.7	160
CB Switch	3	4	6	1.7	.5	9
Average	57	80	94	18.5	2.5	280

Table 9: Simultaneous Event Statistics

Although the statistics indicate that on average relatively few events occur in parallel, this number scales as the size of the circuit being simulated grows. If we assume, for instance, that the average number of simultaneous events scales linearly with circuit size, then a 100,000 transistor circuit will, on average, have about 463 (25×18.5) simultaneous events to process at each non-idle time point. Though not pursued here, it is clear that this affords many opportunities for exploiting parallelism and pipelining in the design of special purpose simulation architectures.

6. Summary and Conclusions

This paper discussed some factors which are of importance in the design of a hardware based logic simulator. A summary of architecture approaches for achieving high performance logic simulation engines was presented along with a description of a software simulator, *lsim*, which has been used as a tool for gathering data on the simulation process. Data was presented on the relative execution times for important simulation subtasks when implemented in software. In general, the data indicates that no single aspect of the simulation algorithm represents a central bottleneck in the simulation process. To achieve high speed, therefore, all subtasks must be made more efficient. Statistics on the length of event queues were also presented. These indicate that most of the event queue lengths are of reasonable size. Statistics on idle time, event intensities and the number of simultaneous events which occur during a simulation were also given. These indicate that logic simulation is characterized by a very small percentage of the circuit components being simultaneously active over a small number of time points. For large circuits, however, these small percentages yield sufficient simultaneous activity so that parallelism and pipelining techniques can be successfully exploited.

Work in the area of modeling the performance of various architectural alternatives is under way at Washington University. Related research on the circuit partitioning problem and on associated problems in the design of more general purpose simulation machines for VLSI design automation is also being pursued.

References

- [ASHO85] Ashok, V., Costello R., and Sadayappan P., "Distributed Discrete Event Simulation Using Dataflow," *Proc. 1985 Int. Conf. on Parallel Processing*, IEEE Computer Society Press, 1985, pp. 503-510.
- [BLAN84] Blank, T., "A Survey of Hardware Accelerators Used in Computer-Aided Design," *IEEE Design and Test of Computers*, 1:3 (Aug. 1984), pp. 21-39.
- [CHAM85] Chamberlain, R.D., "Lsim: A Gate-Switch Level Logic Simulator," M.S. Thesis, Dept. of Computer Science, Washington University, St. Louis, MO., May 1985.
- [CHAM86] Chamberlain, R.D., and Franklin, M.A., "Collecting Data About Logic Simulation," *IEEE Trans. on Computer-Aided Design*, (in press).
- [CHAN81] Chandy, K.M. and Misra J., "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Comm. of the ACM*, 24:11 (Apr. 1981), pp. 198-206.
- [DAIS85] Daisy Systems Corp., "The MegaLOGICIAN," Product Description, Mountain View, CA, 1985.
- [DALL85] Dally, W. and Bryant, R., "A Hardware Architecture for Switch-Level Simulation," *IEEE Trans. on Computer-Aided Design*, CAD-4:3 (July 1985), pp. 239-250.
- [DENN82] Denneau, M.M., "The Yorktown Simulation Engine: Architecture and Hardware Description," *Proc. 19th Design Automation Conf.*,
- [FRAN84] Franklin, M.A., Wann, D.F., and Wong, K.F., "Parallel Machines and Algorithms for Discrete-Event Simulation," *Proc. 1984 Int. Conf. on Parallel Processing*, Aug.
- [HAHN85] Hahn, W. and Fischer, K., "MuSiC: An Event-Flow Computer for Fast Simulation of Digital Systems," *Proc. 22nd Design Automation Conf.*, July 1985, pp. 338-344.
- [HAYE82] Hayes, J. P., "A Unified Switching Theory with Applications to VLSI Design," *Proc. of the IEEE*, 70:10 (Oct. 1982), pp. 1140-1151.
- [HEFF85] Hefferan, P.M., et. al., "The STE-264 Accelerated Electronic CAD System," *Proc. 22nd Design Automation Conf.*, 1985, pp. 352-358.
- [HOWA83] Howard, J.K., Malm, R.L., and Warren L.M., "Introduction to the IBM Los Gatos Logic Simulation Machine," *Proc. IEEE Inter. Conf. on Comp. Design (ICCD'83)*, Oct. 1983, pp. 580-583.
- [JEFF83] Jefferson, David, "Virtual Time," *Proc. Inter. Conf. on Parallel Processing*, 1983, pp. 384-394.
- [PFIS82] Pfister, G. F., "The Yorktown Simulation Engine: Introduction," *Proc. 19th Design Automation Conf.*, June 1982, pp. 51-54.
- [SILI85] Silicon Solutions Corp., "The Mach 1000 Simulation Engine," Product Description, Menlo Park, CA, 1985.
- [ULRI78] Ulrich, E., "Event Manipulation for Discrete Simulations Requiring Large Numbers of Events," *Comm. of the ACM*, 21:9 (Sep. 1978), pp. 777-785.
- [VALI84] Valid Corp., "Realfast Simulation Accelerator," Product Description, 1984.
- [ZYCA83] Zycad Corp., "The Zycad Logic Evaluator," Product Description, N. Roseville, MN., 1983.

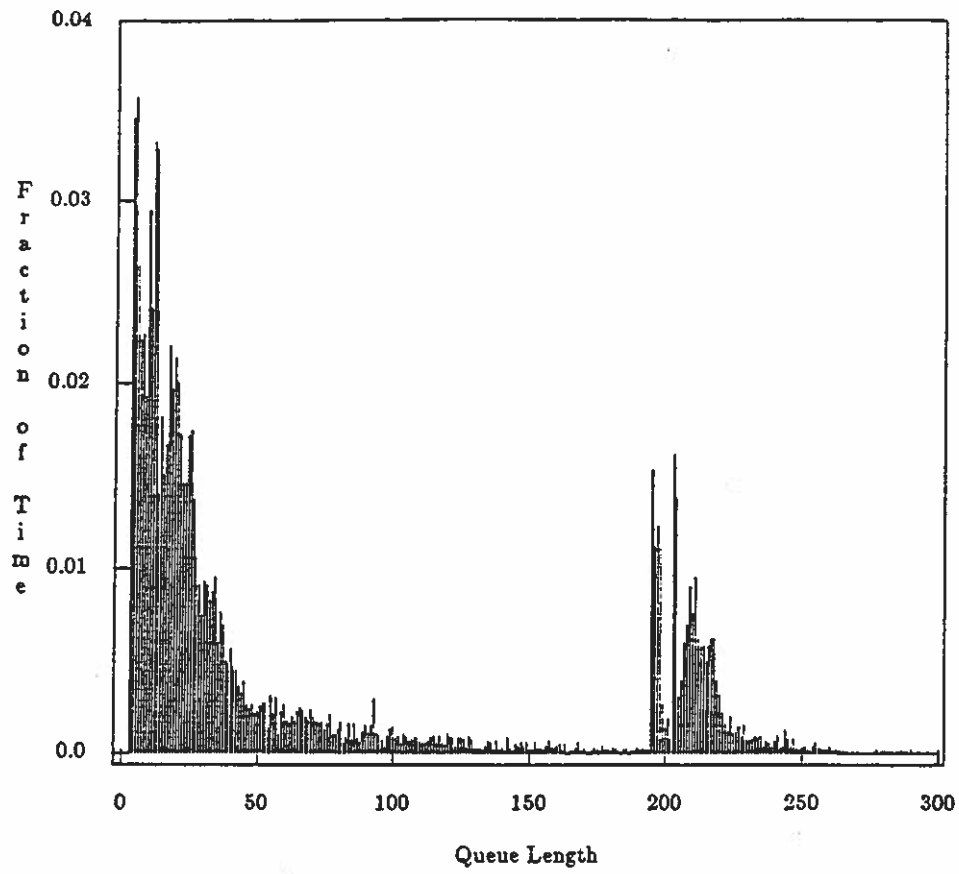


Figure 1: Priority Queue - Queue Length Distribution (non-idle time)

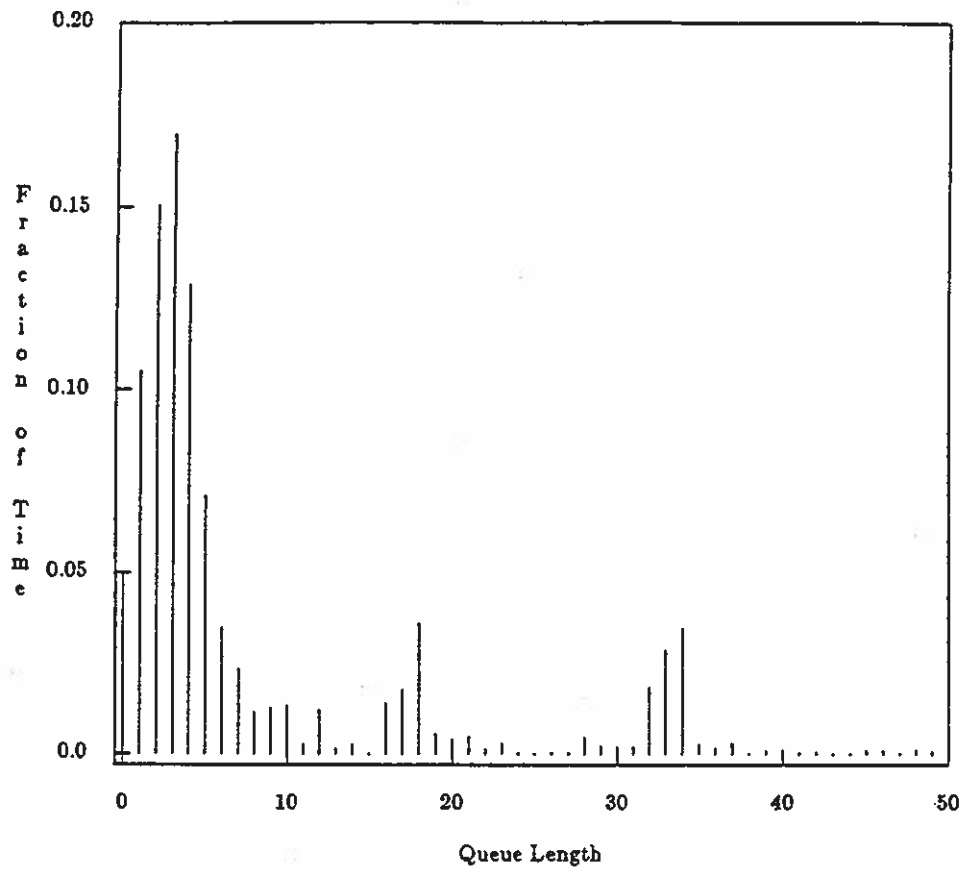


Figure 2: Associative Memory - Queue Length Distribution (non-idle time)

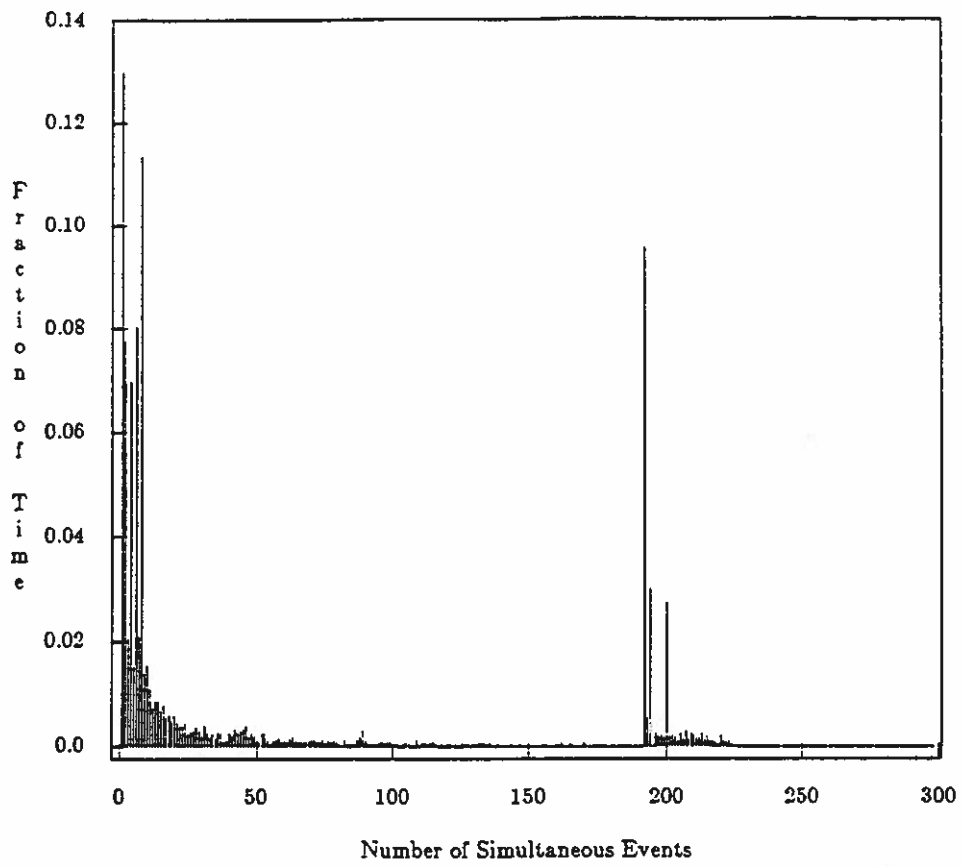


Figure 3: Priority Queue - Simultaneous Events Distribution

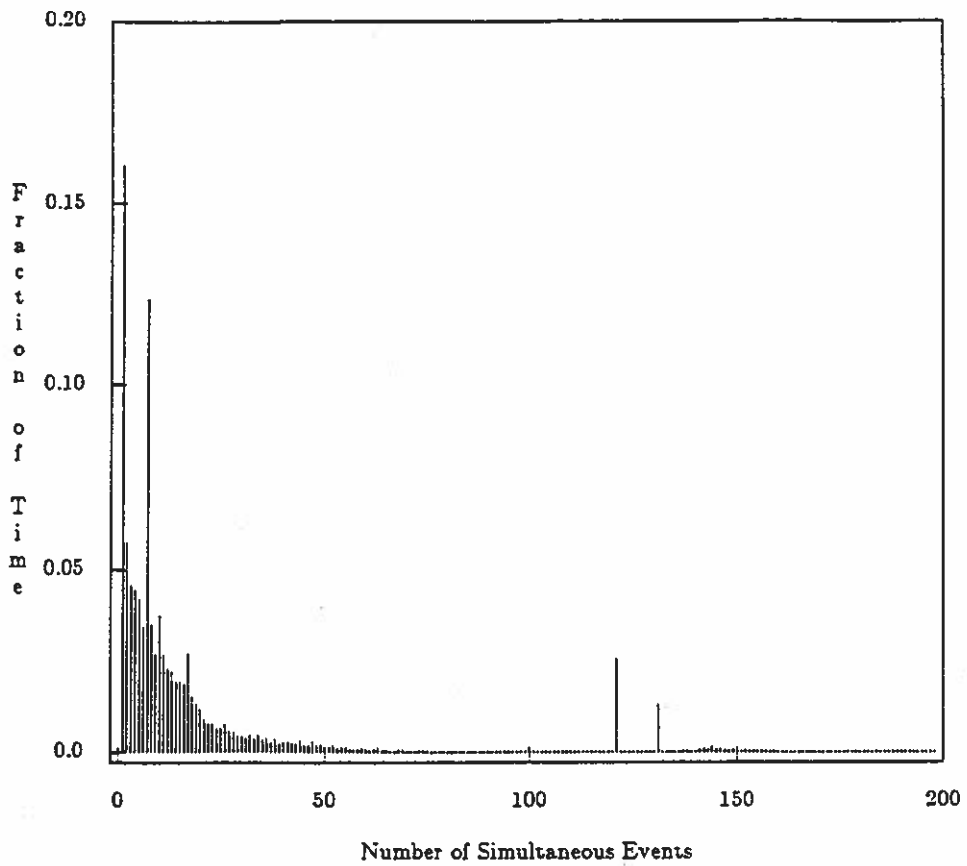


Figure 4: RTP Chip - Simultaneous Events Distribution