Report Number: WUCS-85-06

1985-05-01

Collecting Data About Logic Simulation

Roger D. Chamberlain and Mark A. Franklin

Design of high performance hardware and software based gate-switch level logic simulators requires knowledge about the logic simulation process itself. Unfortunately, little data is publically available concerning key aspects of this process. An example of this is the lack of published empirical measurements relating to the time distribution of events generated by such simulators. This paper presents a gate-switch level logic simulator lsim which is oriented towards the collection of data about the simulation process. The basic components of lsim are reviewed, and its relevant data gathering facilities are discussed. An example is presented which illustrates the use of... Read complete abstract on page 2.

Collecting Data About Logic Simulation

Roger D. Chamberlain and Mark A. Franklin

Complete Abstract:

Design of high performance hardware and software based gate-switch level logic simulators requires knowledge about the logic simulation process itself. Unfortunately, little data is publically available concerning key aspects of this process. An example of this is the lack of published empirical measurements relating to the time distribution of events generated by such simulators. This paper presents a gate-switch level logic simulator lsim which is oriented towards the collection of data about the simulation process. The basic components of lsim are reviewed, and its relevant data gathering facilities are discussed. An example is presented which illustrates the use of lsim in gathering data on event distributions and on communications requirements under alternative logic circuit partitionings.

Collecting Data About Logic Simulation

Roger D. Chamberlain and Mark A. Franklin

WUCS-85-06

May 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 ONR Contract N00014-8D-C-0761, NSF Grant DCR-8417709, and NSF Graduate Fellowhip, and the ITT Corporation.

COLLECTING DATA ABOUT LOGIC SIMULATION

Roger D. Chamberlain and Mark A. Franklin

Center For Computer Systems Design
Campus Box 1115
Washington University
St. Louis, Missouri 63130

Keywords: Simulation, Logic Simulation, Logic Simulators, Logic Simulation Machines, Switch Simulation, MOS Simulation, Discrete Event Simulation, Event Lists, Event List Algorithms

Abstract: Design of high performance hardware and software based gate-switch level logic simulators requires knowledge about the logic simulation process itself. Unfortunately, little data is publically available concerning key aspects of this process. An example of this is the lack of published empirical measurements relating to the time distribution of events generated by such simulators. This paper presents a gate-switch level logic simulator *lsim* which is oriented towards the collection of data about the simulation process. The basic components of *lsim* are reviewed, and its relevant data gathering facilities are discussed. An example is presented which illustrates the use of *lsim* in gathering data on event distributions and on communications requirments under alternative logic circuit partitionings.

COLLECTING DATA ABOUT LOGIC SIMULATION *

Roger D. Chamberlain and Mark A. Franklin
Campus Box 1115
Washington University
St. Louis, Missouri 63130

1. Introduction

With the advent of LSI and VLSI technologies, the role of logic simulation in the design and fault analysis of digital systems has steadily grown in importance. The major reason for its critical role in the verification phase of the design cycle is that the costs associated with fixing design errors after chip and system fabrication have been completed are extremely high. In terms of fault analysis, logic simulation plays a central role in developing test vector sets for use in validating the operation of fabricated systems.

Software based logic simulators based on standard discrete event simulation algorithms have been available for some time [1,2,3] and have been used extensively in the verification and fault analysis tasks. With VLSI, however, where hundreds of thousands of logic components may be present on a single chip, the costs associated with logic simulation have grown enormously. Such simulations can consume months of machine time [4] and, with increased chip component densities, have become a significant bottleneck in the overall design cycle.

One response to this problem has been the design and development of a number of special purpose digital processors tailored to the logic simulation task [5,4,6,7,8,9,10]. These processors can often speed up the simulation process by several orders of magnitude over pure software simulation approaches. For a general review of the techniques and architectures employed in these hardware based logic simulators see Blank [11] and Franklin [12].

In attempting to design either hardware or software based logic simulators, questions often arise relating to the nature of the logic simulation task itself. Table 1 contains a number of

* This research has been sponsored in part by funding from ONR Contract N00014-8D-C-0761, NSF Grant DCR-8417709, an NSF Graduate Fellowship, and the ITT Corporation.
The authors are with the Departments of Computer Science and Electrical Engineering, Washington University, St. Louis, MO.

questions indicative of type of issues which must be addressed in the design of a logic simulator. While some of these questions apply only to hardware based simulators, others apply to both simulator types. The author's have found almost no data in the public literature relating to these issues and the lack of such data is a principal motivation for the work reported in this paper.

Consider, for example, the type of information required to design an effective event list processing algorithm for use in logic simulation. While many event list algorithms have been developed [13,14,15], performance analysis of such algorithms has generally required that various assumptions be made regarding the way events are generated over the simulation time period. Often, for instance, a uniform time distribution has been assumed even though it is generally believed to be incorrect. These untested assumptions may be critical in evaluating the relative performance of the various algorithms.

1. What is the distribution of events over time during execution of logic simulation? Such information is useful in the design of both hardware and software based event list manipulators.

2. How should the logic to be simulated be partitioned when attempting to perform simulation on a parallel hardware based simulator? In certain designs, improper partitioning may lead to uneven load balancing and excessive interprocessor communications delays.

3. How are events distributed within clock cycles? Such information could be of importance in determining the effectiveness of various pipelining strategies for processing multiple events which occur at the same time slot.

4. How much time is spent in the various phases of the simulation algorithm (e.g. event list operations, netlist operations, logic function evaluation, etc.)? Having this information permits one to concentrate resources on those parts of the simulation process which have the greatest effect on simulation costs.

TABLE 1: Some Logic Simulation Questions

This paper presents the *lsim* logic simulator. *Lsim* has been designed to aid in gathering data about the logic simulation process. Indeed, *lsim* has facilities, in conjunction with UNIX task monitoring functions and the S statistical package [16], to collect and display all of the data required to answer the questions of Table 1. The goal in designing *lsim* was to develop a tool which would aid in data gathering important to the design of advanced hardware based logic simulators.

The remainder of this paper is divided into four sections. Section 2 presents the *lsim* gate-switch level logic simulator and illustrates its use in specifying a simple circuit. Section 3 discusses *lsim's* data gathering facilities. Section 4 presents a more extensive example illustrating the use of *lsim's* data collection facilities. The final section presents summary and conclusions about the use of *lsim* collected data in the design of hardware based logic simulators.

2. An Introduction to Lsim

Lsim is a gate-switch level simulator which can simulate systems containing both the traditional TTL unidirectional type of logic gates, and bidirectional switches of the sort found in MOS circuits. It has been written in the C language, and runs under the UNIX operating system. The remainder of this section contains a summary of *lsim* simulation capabilities and an example illustrating its use. A complete description can be found in Chamberlain [17].

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 though an active transistor to supply or ground. A weak signal is one that is connected to a voltage source though 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. Fault simulation is supported through establishment of stuck-at conditions for selected signal lines.

2.1. Logical States

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). 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 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. The transient states are only utilized in the variable delay timing model. The unit and fixed delay timing models only use the first four stable states.

Logical states (except for z) have one of two signal strengths (strong or weak) associated with them. These strengths are used to resolve the state of a signal when more than one component output is connected to the signal. This is the case when using wired logic connections.

2.2. Bidirectional Components

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 (required for proper MOS logic simulation), 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 Hayes [18,19].

2.3. Delay Models and Timing Specifications

The simplest delay model supported by *lsim* is the unit delay model. Timing issues are completely ignored in this model and all components are assumed to have a delay of one unit. This unit delay is used to provide a mechanism for providing functional simulation of the circuit without the overhead involved with more accurate timing simulations.

The second delay model supported is the fixed delay model. *Lsim* treats each component as having fixed low-to-high and high-to-low propagation times associated with each output. 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. Whenever the component is evaluated and it is determined that an output is to change state (i.e., a new event is generated), the time for this new event is calculated as the current time plus the fixed time associated with the delay through the component.

The most accurate delay model takes into account the fact that not all components can have a fixed propagation delay associated with them, but are more realistically modeled by a variable time range within which the output modification is assumed to take place. The variable delay model uses a minimum and maximum value associated with each of the delay times specified, and signal levels are modeled by the three transient states ("r", "f", and "t") during the time between the known stable states.

In addition to the standard component delays discussed above, setup and hold times may be used to specify input timing requirements for memory elements. Violation of setup and hold time requirements are detected by the simulator and reported as an error condition.

2.4. A Simple Example

An example of *lsim* usage is presented below.

2.4.1. Circuit Specification and Compilation

The first step involved in simulating a circuit is describing the properties of the circuit in a machine readable format. This circuit description must include information such as the gates to be simulated, their interconnections, delays, and other properties. In order to facilitate this description, the *circ* circuit compiler has been developed to translate a text file description of the circuit of interest into a format readable by *lsim*. Figure 1 is the schematic diagram of a simple three gate digital circuit. Figure 2 gives the specification of the example circuit as input to *circ*. A complete description consists of at least the following parts: delay specifications, component type definitions, environment specification, and netlist description.

In this example, there are two delay specifications defined (comdel and memdel). These user selected identifiers are referenced later when defining component types. They associate minimum, maximum, and fixed low-to-high propagation, high-to-low propagation, output enable, and output disable times with the given identifier.

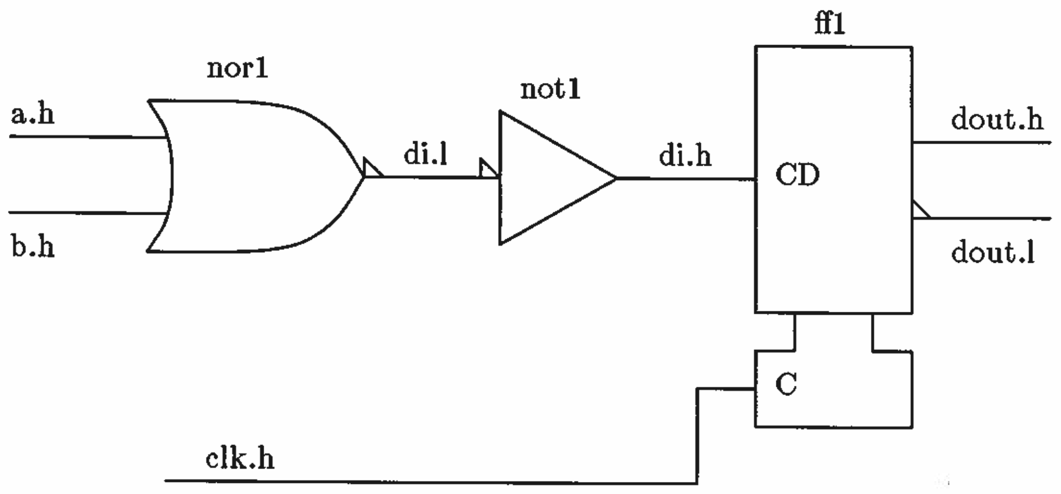

Figure 1. Example circuit

```
# Example circuit specification

begin circuit
      begin delays
            comdel = (8,12,12 $ 2,3,3)ns;
            memdel = (4,6,6 $ 4,6,6)ns;
      end delays;
      begin types
            nmos_inv = (not, dc=(weak, strong), comdel);
            nmos_nor = (nor, 2, dc=(weak, strong), comdel);
            dflip_flop = (dff, st = 3ns, ht = 1ns, memdel);
      end types;
      begin environment
            inputs = (a.h,b.h,clk.h);
            outputs = (dout.h,dout.l);
      end environment;
      begin components
            nor1 = (nmos_nor, inputs = (a.h, b.h), outputs = (di.l));
            not1 = (nmos_inv, inputs = (di.l), outputs = (di.h));
            ff1 = (dflip_flip, inputs = (clk.h, di.h), outputs = (dout.h, dout.l));
      end components;
end circuit;
```

Figure 2. Example circuit specification

There are three component types defined by the user in the circuit description, nmos_inv, nmos_nor, and dflip_flop. They reference the built-in functions not, nor, and dff, respectively. Not and nor are standard combinatorial gates, dff is a level sensitive D flip flop. The other parameters in the type definition indicate the number of inputs, output current drive capability, setup time, hold time, and delay specification to be associated with the type. These component types will be referenced when specifying the logic netlist.

The environment specification defines the primary inputs and outputs of the circuit. The netlist description is where the actual components themselves are described. The components are named, typed, and their connectivity is defined by specifying their input and output signals.

2.4.2. Interactive Simulation

Once a digital circuit has been specified and compiled using *circ,* it is ready to be simulated by *lsim.* After *lsim* has been invoked the user may enter interactive commands to control the simulation process. Some of the more important commands are those which describe the inputs to the simulated circuit and the form of the output required. The following commands set up these parameters for the current example.

```
    lsim> set 0 a.h
    lsim> input b.h 0000000011111111 p
    lsim> input clk.h 00001100 p
    lsim> watch a.h b.h di.l di.h clk.h dout.h dout.l
    lsim> output 1
```

The first command (set) establishes a static low level at the primary input signal a.h. The second two commands define periodic waveforms (designated by use of a "p") to drive the primary input signals b.h and clk.h. The watch command indicates which signals are to be displayed on a periodic basis, and the output command specifies that the output period is to be 1 unit. In this example, the default unit delay model is used and thus the periods referred to in the input and output commands are in terms of these unit times.

The status of all the signals being watched can be determined at any time during the simulation through the use of the status command. Figure 3 shows the results of issuing the status command. The "x" indicates that the logical states of the associated four signals are undefined. The states of the other three signals were established by the previously executed set and input commands. The numbers in the final column are included to associate a signal name with the appropriate column of the simulation output (see Figure 4 where a.h is associated with column 1, b.h with column 2, etc.).

```
lsim> status
a.h                   = 0 (watched)  (1)
b.h                   = 0 (watched)  (2)
di.l                  = x (watched)  (3)
di.h                  = x (watched)  (4)
clk.h                 = 0 (watched)  (5)
dout.h                = x (watched)  (6)
dout.l                = x (watched)  (7)
lsim>
```

Figure 3. Sample Terminal Session

Once the signals for the primary inputs have been established, the simulation is ready to begin. The initial implementation uses a simple output format which is oriented towards use of inexpensive alphanumeric terminals. Figure 4 below shows the results of simulating the system for 24 time units (3 clock periods). The numbers found at the top of each column of output correspond to the signals being watched as indicated in Figure 3. The periodic inputs can be seen in columns 1 (a.h), 2 (b.h), and 5 (clk.h). The remaining signals can be seen to change 1 unit after their respective component inputs change.

At this point the user may proceed in a number of directions. The current simulation may be continued, the delay model could be changed to either fixed delay or variable delay, the current state of the simulation could be stored in a disk file, a previously saved simulation could

```
lsim> start
units     1   2   3   4   5   6   7
0      |0  |0  | x | x |0  | x | x |
1      |0  |0  |   1| x |0  | x | x |
2      |0  |0  |   1|0  |0  | x | x |
3      |0  |0  |   1|0  |0  | x | x |
4      |0  |0  |   1|0  |   1| x | x |
5      |0  |0  |   1|0  |   1|0  |   1|
6      |0  |0  |   1|0  |0  |0  |   1|
7      |0  |0  |   1|0  |0  |0  |   1|
8      |0  |   1|   1|0  |0  |0  |   1|
9      |0  |   1|0  |0  |0  |0  |   1|
10     |0  |   1|0  |   1|0  |0  |   1|
11     |0  |   1|0  |   1|0  |0  |   1|
12     |0  |   1|0  |   1|   1|0  |   1|
13     |0  |   1|0  |   1|   1|   1|0  |
14     |0  |   1|0  |   1|0  |   1|0  |
15     |0  |   1|0  |   1|0  |   1|0  |
16     |0  |0  |0  |   1|0  |   1|0  |
17     |0  |0  |   1|   1|0  |   1|0  |
18     |0  |0  |   1|0  |0  |   1|0  |
19     |0  |0  |   1|0  |0  |   1|0  |
20     |0  |0  |   1|0  |   1|   1|0  |
21     |0  |0  |   1|0  |   1|0  |   1|
22     |0  |0  |   1|0  |0  |0  |   1|
23     |0  |0  |   1|0  |0  |0  |   1|
Simulation halted at time = 24 units.
lsim>
```

Figure 4. Simulator Output

be input, or the session could be terminated.

3. Data Collection Facilities

A major motivation for the implementation of *lsim* was the desire to investigate the simulation algorithm itself. The remainder of this section discusses *lsim's* data collection features for monitoring the simulation task.

3.1. Event

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 that tells when during the simulation that event is to occur. Events are stored in a data structure called the event queue, which handles the scheduling of events as well as the retrieval of the event with the lowest time value. The following statistical data is collected by *lsim* about the event queue while data collection is enabled:

the number of events associated 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 an event is scheduled, retrieved from the event queue, or deleted from the event queue. Each record created contains four fields organized as follows:

field	contents and values
1	insertion ("1"), removal ("0"), deletion ("-1")
2	current simulated time (picoseconds)
3	scheduled time of the event (picoseconds)
4	event type

Insertion and removal of events correspond to the normal processing of events associated with discrete event simulation. Event deletion occurs when certain types of errors (e.g. "spike" errors) are encountered. The current simulated time is recorded before the event is processed for event removal records. The event type identifies the action to be performed by the simulator (e.g. component output modification, display output, etc.) that this event represents. Records are written as ASCII strings with individual records separated by new lines. The resulting data file can be analysed using the *S* statistical analysis package.

3.2. Partitioning

A common approach to achieving high speed logic simulation is to develop parallel hardware architectures and associated parallel simulation algorithms. The general idea here is to partition the logic to be simulated and place each partition on a separate processor. Simulation can, hopefully, proceed in a parallel fashion on the multiple processors thus achieving a computational speedup.

In order to examine various parallel architecture options, the effects of alternative logic partitioning schemes must be examined. A poor partitioning can result in either overloading certain processors, overloading the interprocessor communications network, or both.

Lsim supports the collection of communication requirements between components in different partitions in an attempt to measure quantitatively the interprocessor (interpartition) communication that takes place. Each component may be assigned a partition (if unspecified, it defaults to zero), and each time a component output changes and the resulting signal is propagated to another component, an interpartition count is updated. This information is stored in a square N x N matrix (N is the total number of partitions) whose entries contain a count of source partition (row number) to destination partition (column number) communications. At the end of the simulation, each entry in the matrix tells the amount of communication required between the various partitions. If the overhead involved with communicating across partitions is large, it is obviously advantageous to have most of the communications requirements show up on the major diagonal, meaning the communication was completely within a partition.

3.3. Task Execution

The standard UNIX profiling utilities can be used to determine the cpu times of various tasks involved in the 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 current task classifications being used are the following:

event queue manipulation
functional evaluation
netlist operations
user output
other overhead

The event queue manipulation includes the insertion, retrieval, or deletion of events from the event queue. Functional evaluation is the determination of component output output values given component input values. The netlist operations refer to the propagation of component output changes to the inputs of other components, in effect, searching the connectivity of the circuit. The user output refers to the time spent providing the periodic display of signal states. The other overhead represents the time that could not be easily classified as one of the other tasks.

3.4. Output Format

To demonstrate the output format of the data collection facilities, the circuit of Section 2 is simulated with data collection enabled. The resulting terminal session is shown in Figure 5. Most of the data reported is self explanatory. Note that the times shown in the example are in units since a unit delay timing model was used. Note also that the default of a single partition (partition 0) was used, resulting in an interpartition communications matrix containing only a single entry.

```
% lsim circuit.ls
Simulator state input from file circuit.ls
Current simulated time = 0 units.
lsim> input clk.h 00001100 p
lsim> set 0 a.h
lsim> input b.h 0000000011111111 p
lsim> collect on
lsim> halt 64
lsim> start
Simulation halted at time = 64 units.
lsim> collect off
The number of events processed for each component in the circuit is:
nor1                  8           ff1                  16
not1                  8
The average number of events in the event queue = 2.500
The standard deviation is 0.707
The maximum number of events in the queue = 5
The total number of events processed = 56
The total number of events deleted from the queue = 0
The average time between events = 1.143 units
The maximum time between events = 2 units
The minimum time between events = 0 units

The summary of communication across partitions is as follows:
          0
     0    78
lsim>
```

Figure 5. Data collection output

4. Example Simulation

In this section the simulation task oriented data collection capabilities of *lsim* are illustrated with a more extensive circuit example. The circuit to be simulated contains about 1000 transistors and functions as a stopwatch controller. It was designed as part of an undergraduate VLSI design course using NMOS technology and was subsequently fabricated and shown to operate as specified.

4.1. Circuit Description

The function of the stopwatch is to keep track of the time between successive activations of the run/stop signal (which would be connected to a button) in order for the user to measure the elapsed time between two observed events. In addition, a clear input signal is provided to allow the user to reset the time to an initial zero state. The time base, a two phase non-overlapping clock, is generated externally to the chip.

Output from the stopwatch is provided in the form of four binary coded decimal digits, representing the four digits in a time display ranging from 0 to 99 seconds with a precision of .01 sec. There are only two control inputs to the chip. The clear signal initializes the time to zero by clearing all the internal counters, and the run/stop signal is responsible for starting and stopping the timing operation. The first time the run/stop signal undergoes a low-to-high transition, the stopwatch starts running and continues until a second low-to-high transition on the run/stop signal.

The block diagram for the stopwatch is shown in Figure 6. The control inputs are "clr", the clear signal, and "rs.h", the run/stop signal. "Phi1" and "phi2" are the two phase clock inputs. For this data collection exercise the clock period was set at 200 ns. The outputs are "d0" through "d15". "c0" through "c4" are carry signals that tell the counters when to increment. There are two major component types that comprise the stopwatch, an edge detector and six bcd counters. These are defined as macros in *lsim*.

The edgedet component is a rising edge detector. It detects the rising edge of the "rs.h" input and converts it into a "run.h" signal. "Run.h" is a logical high when the stopwatch is running and is a logical low otherwise. The bcdcount components are synchronous counters that count from 0 to 9 when their "count" inputs are high. "Run.h" is the "count" input for bcdcount1, and "c0" through "c4" are the "count" (ci) inputs for bcdcount2 through bcdcount6, respectively.

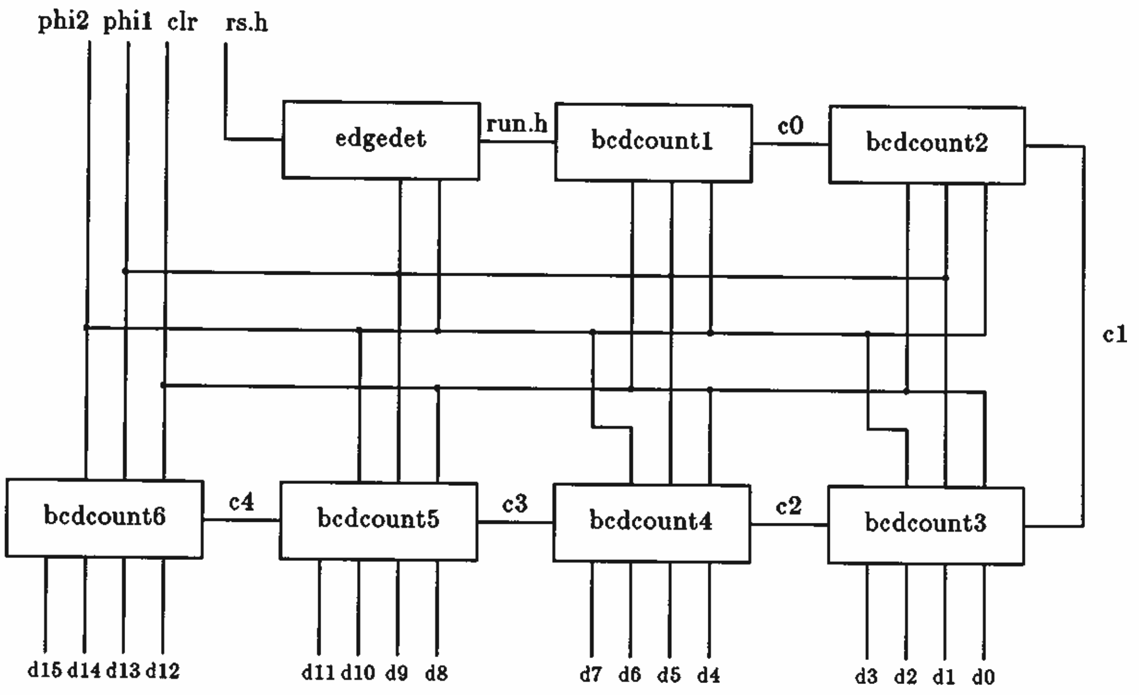

Figure 6. Stopwatch block diagram

4.2. Event Information

Figures 7 and 8 illustrate two types of event oriented information which can be obtained using *lsim*. Figure 7 shows the distribution of future events that were generated over an entire simulation run. Every time a new event is scheduled, the time difference between the current simulation time and the new event time is calculated. This figure is a histogram of all the time differences which have been calculated during the circuit simulation.

Several points of interest should be noted. First, almost all events which are scheduled are within a 200 ns time interval from the current time. An implication of this is that a simple timing wheel event processing algorithm with a relatively small number of entries will be able to

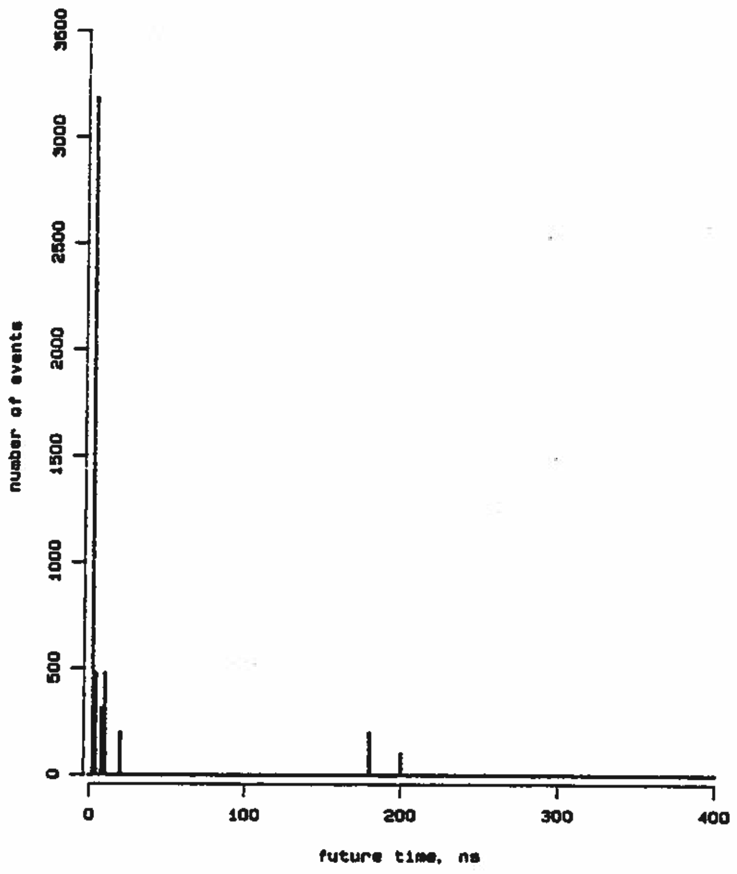

Figure 7: Scheduled Time of Events

process events efficiently in constant (as opposed to linear or log N) time. The distribution of events is also what we might expect with a digital system which has a 200 ns clock period. That is most events are scheduled for times shortly after the clock period begins, or they are scheduled for the start of the next clock period (about one clock period in the future).

Figure 8 shows the time position of all events generated over the entire simulation, but scaled to indicate their location within the 200 ns clock period. As expected, most events occur near the beginning of the clock period with hardly any events being scheduled for the second half of the period. There are several implications of this distribution. First, any simulation algorithm which determines whether there are events to process on a clock tick by clock tick basis will encounter many clock ticks when there are no events available to process. These types of simulation algorithms (and their corresponding hardware architectures) thus have an overhead (possibly high) associated with processing clock times at which no action (events) take place.

Second, the type of data found in this figure permits one to determine the effectiveness of multiple processor pipelined architectures for hardware logic simulation. One way in which hardware simulation speeds can be increased is to pipeline the processing of events that occur at the same clock tick. If there are Ni events which occur at clock tick i, and if these events are evenly distributed across M processors, then each processor has Ni/M events to process at time i. If P represents the number of stages in an event processing pipeline, then the pipeline will only be effective if Ni/M is larger then P over a sufficient number of i time points (i.e. the pipeline is full a reasonable amount of time). The data contained in this figure (at a somewhat higher resolution) allows one to make this sort of determination. A complete performance model of this type of architecture is now being developed.

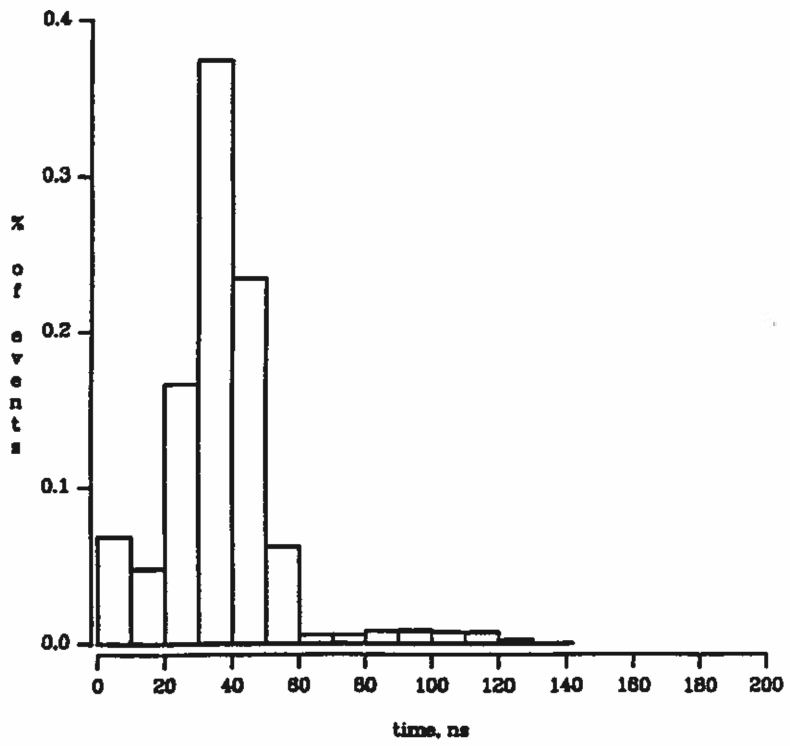

Figure 8: Distribution of Events Within A Clock Period

4.3. Circuit Partitioning

An important problem which must be considered when dealing with multiple processor based logic simulators relates to determining how circuit components are to be allocated to the individual processors. Figure9 show the communications impact of two alternative partitionings of the stopwatch circuit across eight partitions (processors). In the first case (upper part of figure) partition 0 corresponds to the input lines, partition 1 to the output lines, and the remaining partitions to the major blocks found in Figure 6 (edgedet, bcdcount1, bcdcount2, etc.). The results show a good deal of communications from the input (partition 0) to all the other partitions, light communications from the four output blocks to the output lines and between the blocks and their neighbors, and the bulk of communications within each block itself (i.e. large numbers on the main diagonal). Given that communications within a partition has very low cost (i.e. results in little delay), this partitioning scheme is probably an effective one.

This is contrasted with the second case where components have been randomly allocated to eight partitions. In this case reasonable size entries appear in almost every matrix location indicating that there is communications between all of the processors. Given a processor interconnection network which has limited bandwidth, such a component allocation could result in a serious performance bottleneck. This type of information is now being utilized to evaluate effects of different partitioning algorithms on the performance of various hardware architectures.

5. Summary

This paper has described the *lsim* logic simulator, focusing on the data collection facilities available for obtaining information about the logic simulation algorithm itself. An example was provided illustrating the use of *lsim* in collecting data on event distributions over an entire simulation run and event activity within a clock cycle. In addition, the powerful component partitioning analysis capability was discussed. Research is currently being pursued which is using *lsim* to generate simulation task data on a benchmark of circuits. This data is to be used in the development of realistic models of hardware architectures tailored to the logic simulation

	0	1	2	3	4	5	6	7
0	408	16	809	1653	1649	1649	1649	1649
1	0	0	0	0	0	0	0	0
2	0	0	29	6	0	0	0	0
3	0	0	0	4051	76	0	0	0
4	0	0	0	0	767	12	0	0
5	0	4	0	0	0	216	4	0
6	0	4	0	0	0	0	208	4
7	0	4	0	0	0	0	0	208
8	0	4	0	0	0	0	0	0

functional block component allocations

	0	1	2	3	4	5	6	7
0	141	88	186	51	87	104	47	102
1	74	155	10	129	112	35	15	79
2	109	60	6	53	36	102	157	5
3	39	37	9	40	36	19	48	20
4	17	88	87	52	4	48	17	54
5	246	37	37	20	58	12	101	129
6	405	1529	518	695	512	808	210	1025
7	25	50	119	85	218	45	30	43
8	1018	661	663	610	620	1145	355	466

random component allocations

Figure 9 : Communications Across Circuit Partitions

algorithm. From this it is hoped that faster and more effective simulation machines can be developed.

REFERENCES

1. Szygenda, S. A., "TEGAS2 - Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic," Proceedings of the 9th Design Automation Workshop, Dallas, Texas, June 26-28, 1972, pp. 116-127.

2. Control Data Corporation, LOGIS User's Manual, Control Data Corporation, Santa Clara, California, 1980.

3. VLSI Design Staff, "1985 Survey of Logic Simulators," VLSI Design, Vol. 6, No. 3, March 1985, pp. 74-80.

4. Pfister, G. F., "The Yorktown Simulation Engine: Introduction," Proceedings of the 19th Design Automation Conference, June 1982, pp. 51-54.

5. abramovici, M., Levendel, Y.H., and Menon, P.R., "A Logic Simulation Machine," IEEE Tran. on Computer-Aided Design, 2, 2, April 1983, pp. 82-94.

6. Koike, N., Ohmori, K., Kondo, H., and Sasaki, T., "A High Speed Logic Simulation Machine," Digest of Papers COMPCON, Spring 1983, March 1983, pp. 446-451.

7. Denneau, M. M., "The Yorktown Simulation Engine: Architecture and Hardware Description," Proc. of the 19th Design Automation Conf., June 1982, pp. 55-59.

8. Howard, J.K., Malm, R.L., amd 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.

9. Zycad Corp., The Zycad Logic Evaluator: Product Description, Zycad Corp., N. Roseville, MN., 1983.

10. Valid Corp., "Realfast Simulation Accelerator," Product Description, 1984

11. Blank, T., "A Survey of Hardware Accelerators Used in Computer-Aided Design," IEEE Design and Test of Computers, 1, 3, August 1984, pp. 21-39.

12. Franklin, M. A., Wann, D. F., and Wong, K. F., "Parallel Machines and Algorithms for Discrete-Event Simulation," Proceedings of the 1984 International Conference on Parallel Processing, August 1984, pp. 449-458.

13. Ulrich, E., "Event Manipulation for Discrete Simulations Requiring Large Numbers of Events," Comm. of the ACM, 21, 9, Sept. 1978, pp. 777-785.

14. Ulrich, E., "Table Lookup for Fast and Flexible Digital Logic Simulation," Proc. of the 17th Design Automation Conf., June 1980, pp. 560-563.

15. McCormack, W.M., and Sargent, R.G., "Analysis of Future Event Set Algorithms for Discrete Event Simulation," Comm. of the ACM, 24, 12, Dec. 1981, pp. 801-812.

16. Becker, R. A., and Chambers, J. M., S An Interactive Environment for Data Analysis and Graphics, Wadsworth, Inc., Belmont, CA, 1984.

17. Chamberlain, R.D., "LSIM: A Gate-Switch Level Logic Siimulator," M.S. Thesis, Dept. of Computer Science, Washington Univ., St. Louis, MO. 63130, May 1985.

18. Hayes, J.P., "A Unified Switching Theory with Applications to VLSI Design," Proc. of the IEEE, 70, 10, Oct. 1982, pp. 1140-1151.

19. Kawai, M. and Hayes, J. P., "An Experimental MOS Fault Simulation Program CSASIM," Proc. of the 21st Design Automation Conf., June 1984, pp. 2-9.