Report Number: WUCS-84-3

1984-08-01

Parallel Machines and Algorithms for Discrete-Event Simulations

M. A. Franklin, Donald F. Wann, and K. F. Wong

A number of recent articles have focused on the design of high speed discrete-event simulation (DES) machines for digital logic simulation. These investigations are in response to the enormous costs associated with the simulation of complex (VLSI) digital circuits for logic verification and fault analysis. One approach to reducing simulation costs is to design special purpose digital computers that are tailored to the logic simulation test. This paper is concerned with the architecture of such logic machines. The paper has three principal parts. First, a taxonomy of logic machine architectures is presented. The taxonomy focuses on the central components... Read complete abstract on page 2.

Parallel Machines and Algorithms for Discrete-Event Simulations

M. A. Franklin, Donald F. Wann, and K. F. Wong

Complete Abstract:

A number of recent articles have focused on the design of high speed discrete-event simulation (DES) machines for digital logic simulation. These investigations are in response to the enormous costs associated with the simulation of complex (VLSI) digital circuits for logic verification and fault analysis. One approach to reducing simulation costs is to design special purpose digital computers that are tailored to the logic simulation test. This paper is concerned with the architecture of such logic machines. The paper has three principal parts. First, a taxonomy of logic machine architectures is presented. The taxonomy focuses on the central components of the logic simulation algorithms and on architectural alternatives for increasing the speed of the simulation process. It thus represents a basis for discussing and differentiating between proposed architectures and also results in the identification of several new architectures. Although developed for digital logic simulators., the taxonomy can be used for general DES machines. Second, a performance measure is presented which permits evaluation of DES machines. Finally several DES machine designs are described and categorized using the taxonomy.

PARALLEL MACHINES AND ALGORITHMS FOR DISCRETE-EVENT SIMULATION

M. A. Franklin, D. F. Wann,
and K. F. Wong

WUCS-84-3

Proceedings 1984
International Conference
on Parallel Processing
Bellaire, Michigan
August 21-24, 1984

Department of Computer Science
and
Department of Electrical Engineering
Washington University
St. Louis, Missouri 63130

This work was supported in part by NSF Grant MCS-78-20731 and
ONR Contract N00014-8D-C-0761.

1984 International
Conference on Parallel
Processing

PARALLEL MACHINES AND ALGORITHMS FOR DISCRETE-EVENT SIMULATION*

M. A. Franklin, D. F. Wann, and K. F. Wong
Center for Computer Systems Design
Campus Box 1115
Washington University
St. Louis, MO 63130

ABSTRACT

A number of recent articles have focused on the design of high speed, discrete-event simulation (DES) machines for digital logic simulation. These investigations are in response to the enormous costs associated with the simulation of complex (VLSI) digital circuits for logic verification and fault analysis. One approach to reducing simulation costs is to design special purpose digital computers that are tailored to the logic simulation task. This paper is concerned with the architecture of such logic machines. The paper has three principal parts. First, a taxonomy of logic machine architectures is presented. The taxonomy focuses on the central components of the logic simulation algorithm and on architectural alternatives for increasing the speed of the simulation process. It thus represents a basis for discussing and differentiating between proposed architectures and also results in the identification of several new architectures. Although developed for digital logic simulators, the taxonomy can be used for general DES machines. Second, a performance measure is presented which permits evaluation of DES machines. Finally, several DES machine designs are described and categorized using the taxonomy.

1. Introduction

Recently, a number of efforts have focused on the design of high speed, discrete-event simulation (DES) machines for digital logic simulation. These efforts are in response to the enormous costs associated with the simulation of complex (VLSI) digital circuits for logic verification and fault analysis. Such simulations can consume months of machine time [PFIS82A] and have become an unheralded, but significant, bottleneck in logic design and test vector generation.

These high costs have two origins. First, logic simulation applications are computationally intensive with running times which, (on an empirical basis), appear to grow on the order of O(N**2) to O(N**3) where N is the number of circuit components. Second, the rapid growth in VLSI component/chip densities has required simulation of ever larger circuits. From 1960 until just recently, there has been roughly a doubling of the number of circuits/chip every year, resulting in circuit/chip densities currently approaching 500,000 per chip. While this increase appears to be "slowing down" to a doubling about every two years [ROBI84], this still represents substantial growth which, in conjunction with the order effects associated with the simulation computation, have made logic simulation a major limitation in the overall design process.

Approaches to reducing simulation costs fall into two broad categories. The first is to employ divide and conquer techniques. That is, divide the circuit into smaller, more manageable modules for which simulation costs are not as severe. Currently, this is the predominant approach. Unfortunately, it often has the difficultly of not providing effective tests for the interfaces between modules, and for the problems related to overall circuit integration. The second approach is to design special purpose digital computers tailored to the logic simulation task. Such special purpose computers can have performance orders of magnitude faster than the current software based simulators. Recently, several machines have been proposed and built [KOIK82, PFIS82A, HOWA83A, ZYCA83] and the references indicate those efforts known to us at this writing. Since high performance simulation represents an area of entrepreneurial interest, other unreported efforts are undoubtedly also taking place at this time.

This paper is concerned with the architecture of such logic machines. It is assumed that it is necessary to have logic simulations which yield both functional and timing information. The paper has three principal parts. First, a taxonomy of logic machine architectures is presented. The taxonomy focuses on the central components of the logic simulation algorithm and on architectural alternatives to speeding the simulation process. It thus represents a basis for discussing and differentiating between proposed architectures and also permits identification of several new architectures. Although restricted to digital logic simulators, the taxonomy can be used for general DES machines. Second, a performance measure is presented which permits evaluation of DES machines. Finally, recent DES machine designs are briefly described and categorized using the taxonomy. The discussion assumes that the reader is familiar with the operation of software based discrete event simulators.

2. Speed Up Techniques

To better understand the proposed taxonomy, it is useful to review some of the options available for increasing the processing speed of logic

* This research has been sponsored in part by funding from ONR Contract N00014-8D-C-0761 and NSF Grant MCS-78-20731.

simulation. Several of these speed-up techniques are given in Table 1. Over the years, software logic simulators have improved in speed due both to better software design (ingenuity) and to the use of faster general purpose computers (technology). Some further increases can be expected as logic simulation codes are tailored to the next generation of vector processors.

To achieve the several orders of magnitude speed increase needed for effective (hopefully interactive) simulation of VLSI based systems, requires use of one or more of the architectural techniques indicated in Table 1. Functional specialization refers to those techniques which take a software task and decrease its execution time by placing it in hardware. The basic sequential nature of the overall simulation algorithm is, however, maintained. Since event list management, function evaluation and net list operations are critical elements in logic simulation, they represent prime candidates for incorporation into hardware. For example, it is possible to design a hardware priority queue which, from the viewpoint of the rest of the simulator, essentially permits event list manipulations in (small) constant time [LEIS81]. Most of the hardware machine proposals contain some specialized devices, most typically in the function evaluation and net list manipulation areas.

Further speed increases can only be achieved by exploiting the inherent concurrency which exists on the one hand in the circuit to be simulated, and on the other hand in the simulation algorithm itself. This observation leads to multi-processor architectures where processors in parallel simulate subsets of the entire circuit and where, within each processor, the simulation algorithm is executed in a pipelined fashion. By placing simultaneously active components on different processors, a high degree of concurrency can be achieved. Provision, of course, must be made for a fast communications mechanism that is needed to transfer circuit state updates between processors. In addition, if many events are to be processed on a processor at a given time step, then pipelining techniques can be successfully used. These techniques, used in conjunction with functional specialization, can provide significant increases in simulation speed. The taxonomy presented below focuses on the concurrency aspects of logic machine design.

ARCHITECTURE
Functional Specialization
1. Special event list hardware
2. Special function evaluation hardware
3. Special hardware for net list operations
Concurrency Exploitation
1. Parallelism.
2. Pipelining.
INGENUITY
Better event list algorithm
More efficient net list data structure
TECHNOLOGY
Use of faster logic family.

Table 1: Logic Simulation Speed-Up Techniques

3. A Taxonomy for DES Machines

The taxonomy presented here attempts to depict the main architectural features without dwelling on fine details. It is comprehensive in that it permits description of currently proposed architectures while identifying several which are not immediately obvious. It uses a shorthand notation similar in spirit to Flynn's notation for parallel machine architecture (e.g. MIMD, SIMD etc.) [FLYN66] and Kendall's notation for queueing systems (e.g. M/M/1, M/D/10, etc.) [KEND53].

The basis for the taxonomy is the observation that every DES algorithm contains three essential components:

1. Time control
2. Event list control
3. Event (function) evaluation.

How these components are implemented may vary from one simulator to another, but they are all present in one form or another in any DES. The time control component determines the progression of simulated time. For obvious reasons, time control is often referred to as clock control. The event list control component is concerned with the proper posting and scheduling of events (in the event list) in increasing simulated time order. The event or function evaluation component performs the processing associated with the accessed events (e.g. evaluation of a logic gate) and determines if any new events are to be scheduled.

A taxonomy notation is presented in Table 2. The taxonomy has four components: two specifying time control characteristics, one for specifying the event list control component, and one for the event evaluation component. The Time Control Mechanism consists of two parts. The first relates to how the machine advances simulated time. The simplest approach is to advance time in Unit Increments (UI). Indeed many of the early software simulators used this approach. In software simulators this approach is inefficient since there are often no events to be processed at a given time. Incrementing the clock on a unit basis, in this case, results in a fixed and unnecessary overhead for each time step. With Event based time Incrementing (EI) the clock is

```
+-----------------------------------------+
| 1. TIME CONTROL MECHANISMS              |
|    a- TIME ADVANCE                      |
|       1) Unit Increment ----------> (UI)|
|       2) Event based Increment -------> (EI)|
|    b- TIME SYNCHRONIZATION              |
|       1) Global Clock ----------> (GC)  |
|       2) Local Clock ----------> (LC)   |
| 2. EVENT LIST ATTRIBUTES                |
|       1) Single List ----------> (SL)   |
|       2) Multiple List ----------> (ML) |
| 3. EVENT/FUNCTION EVALUATION            |
|       1) Single Machine (Serial) -----> (SM)|
|       2) Multiple Machine (Parallel) --> (MM)|
+-----------------------------------------+
```

Table 2: Taxonomy Components

advanced to the time associated with the next event to be processed, skipping over intermediate times when no events are scheduled to occur.

Since there may be several processors present, the question of time synchronization must be addressed. That is, is there a single global clock (GC) which is communicated to all the processors, or is the clock distributed (LC), with each processor controlling its own version of simulated time? In the latter case, care must be taken to ensure that the processors maintain synchronization; if not simulation errors can occur.

With several processors present, it is also possible to have the event list located either on a single processor (SL) (with the processor perhaps broadcasting the next event to be processed), or distributed across several or all of the processors (ML). The time associated with event list processing may be reduced by having multiple lists on multiple machines. Furthermore, it may be possible to more fully exploit pipelining within each processor with a distributed event list.

The final element in the taxonomy relates to event/function evaluation and designates whether the evaluation is performed on a Single Machine (SM) or on Multiple Machines (MM). For added speed, these machines may be specialized to perform logic function evaluations quickly. However, that is not specified explicitly in this taxonomy.

A machine architecture can now be specified by a four tuple of the following form:

```
TIME ADVANCE /
  TIME SYNCHRONIZATION /
    EVENT LIST ATTRIBUTES /
      EVENT EVALUATION
```

Note that with this taxonomy there are sixteen possible architectures.

3.1. Single Machine Architectures

As an example of use of the taxonomy, a standard serial machine running a "primitive" unit increment clock simulator would be designated as:

```
UI/GC/SL/SM
 |  |  |  |
 |  |  |  |--- Single Machine
 |  |  |------ Single List
 |  |-------- Global Clock
 |----------- Unit time Increment
```

Since, with a single machine, a global clock is identical to a local clock, this notation is equivalent to UI/LC/SL/SM . Likewise, the standard serial machine software simulator using event based timing increments would be denoted as:

EI/GC/SL/SM or EI/LC/SL/SM

Thus, ordinary serial processor based simulators comprise four of the sixteen architectures possible within this taxonomy. The remaining twelve, however, yield a variety of interesting possibilities.

3.2. Multiple Machine Architectures With Global Clock

As pointed out earlier, a natural approach to exploiting parallelism is to divide the circuit to be simulated across several processors, with each processor being responsible for evaluating events associated with its assigned circuit subset. Assume for this discussion that a global clock is used to minimize time synchronization problems. There are four possible architectures which have a global clock:

EI/GC/SL/MM and UI/GC/SL/MM
EI/GC/ML/MM and UI/GC/ML/MM

The first category, EI/GC/SL/MM, is illustrated in Figure 1. The single clock in the diagram with the +E over it indicates that the architecture uses a global clock with event based timing increments. In this situation, one of the processors is designated as a master processor. This master processor contains the global event list and increments its clock, based on the time of the next event in its list. As events are taken from the list they are sent over the communications network to the multiple processors for evaluation. The processors perform this evaluation in parallel, and then return new event and state change information to the master, which, in turn, updates its global event list. The cycle is then repeated.

If, at a given time point, there are a number of events which require processing, and if this event processing can be distributed across the parallel processors, then an increase in simulation speed may result. Such an architecture could also make good use of functional specialization in the event list processing and in the function evaluation areas. Note though, that while centralized event list processing may simplify overall operation, it may also produce a performance bottleneck. Other issues which must be addressed here, and in the remaining architectures, relate to communications network design and logic partitioning.

The UI/GC/SL/MM architecture is similar to the EI/GC/SL/MM architecture discussed above, except that a unit time increment is employed. It would, however, likely have lower performance due to the overhead associated with processing time increments that have no associated events.

Consider next the UI/GC/ML/MM architecture shown in Figure 2. This organization has a unit increment global clock with multiple event lists and multiple machines. One implementation approach would have the master processor send a control message to the n slave processors signaling each clock tick. On receipt of the clock signal the function evaluation process would begin. Each slave processor would in turn perform all the evaluations on its local event list which are associated with the current time step. Those evaluation results which impact subcircuits located on the other processors are now communicated to the appropriate processors. When these tasks are completed, each slave returns a completion signal to the master processor. After all slaves have reported completion, the master processor proceeds to the next clock tick (a simple clock increment) and the cycle then repeats. Note that each slave keeps track of the activity associated with circuit elements assigned to it, and events associated with these elements are kept on the local event list.

The UI/GC/ML/MM architecture achieves its speed-up through parallel function evaluation and parallel event list manipulation using a simple time management scheme. Functional specialization and pipelining can also be used for further speed increase. As will be seen in Section 5, most logic simulation machines which have been proposed or built are variants of this architecture. Clearly, the related architecture EI/GC/ML/MM is also possible. Though not requiring unit time incrementing, this approach would require that additional event time information be returned from the slave processors to the master so that the global clock could be properly updated. The simple completion signal approach associated with the UI/GC/ML/MM architecture avoids these complications, but at the expense of having clock ticks where no action takes place.

3.3. Multiple Machine Architectures With Local Clocks

The taxonomy suggests the following four multiple machine, local clock architectures.

EI/LC/SL/MM and UI/LC/SL/MM
EI/LC/ML/MM and UI/LC/ML/MM

The EI/LC/SL/MM architecture is illustrated in Figure 3. This is a somewhat unusual architecture in that there is a combination of a global event list and local clocks. The local clocks in this case do not appear to enhance performance since the single event list will, in effect, set the time for the entire system of clocks by determining the next event (and its associated time) to be processed. That is, the presence of the single event list appears to have the effect of a global clock. Of course, the master processor, which has the single event list and some knowledge of event processing times, could possibly direct the slave processors to perform event evaluations for different future times. This would add more complexity to the simulation algorithm, and on the surface, does not appear to be promising. The UI/LC/SL/MM architecture appears even less promising and is not pursued here.

The EI/LC/ML/MM architecture is shown in Figure 4. This system has all the properties of a true MIMD machine running tasks in an asynchronous manner. An EI/LC/ML/MM simulator views the simulation as a set of autonomous, communicating processes [CHAN81, PEAC79]. Once again, subsets of the entire circuit would be mapped on the different processors. Time synchronization and state information exchange could be achieved through message passing between the processors. Each message could have a local clock time stamp associated with the processor sending the message. With one algorithm, local clocks would only be advanced to the time associated with the lowest time stamp over all of its incoming processes. This would prevent the processor from evaluating events which may be in its local event list, but which are too far ahead in time for proper processing. Other algorithms would permit the processors to evaluate as many events as possible, performing rollback maneuvers when synchronization errors occur [JEFF83].

Potential speed up in this situation is obtained through parallel function evaluation, parallel event list manipulation and distributed time management. Distributed time management allows circuit component states to be evaluated as each input value changes. Thus, states can be evaluated at the earliest possible time. In a global clock scheme, time generally cannot be advanced until all slave processors have finished their processing for the current time frame. That is, in the global clock case, the speed of the

simulator is governed by the longest processing time in each time frame. Against these possible speed advantages associated with the EI/LC/ML/MM simulator is the complexity and overhead associated with distributed process management. The performance issues here are not yet fully understood. The related UI/LC/ML/MM architecture would probably have a somewhat simpler distributed time management scheme, but at the cost of eliminating a good deal of potential overlap in processor function evaluation.

The four remaining architectures are as follows:

EI/GC/ML/SM and UI/GC/ML/SM
EI/LC/ML/SM and UI/LC/ML/SM

It is left to the reader to determine if these remnants of the taxonomy are worthwhile.

4. A Performance Measure

In order to compare the relative performance of simulation machines, a performance measure must be developed. In this section certain variables affecting performance are identified and a performance measure is proposed. The measure presented can form the basis for making comparisons among several of the machines described in the literature. The performance measure, P, is derived from analyzing the cost of simulation and includes factors relating to the simulation machine speed, machine cost, waiting time cost, and simulation quality. The proposed performance measure is:

$$P = Ts*(Cm + Cw) + fq$$
$$= Ts*(Cm + fw(Ts)) + fq$$

where
Ts = simulation time (total real time to execute a simulation for a given problem)
Cm = machine cost/unit time
Cw = waiting cost/unit time = fw(Ts)
fq = simulation quality function

Consider the various factors in this relation. The longer the running time, Ts, of the simulation, the larger is the cost. The coefficient relating P and the simulation time is composed of the sum of two costs, the machine usage cost (Cm) and the cost to the user for waiting (Cw) until the simulation is finished. But observe that the waiting time cost is usually not linearly related to the simulation time, but grows in some exponential manner, that is fw(Ts). The function fw(Ts) accounts for the penalty of, say, waiting for long simulation turnaround times or the opportunity loss associated with delays in introducing a product. A third property which is hard to quantify, and therefore is not included in our performance measure, is related to the quality of the simulator (e.g., unit delay timing, inertial delay, number of states). The parameter values for the machine cost component can be supplied either by measurements of existing systems or by estimates obtained by simulation models of the alternative architectures.

The simulation execution time (Ts) can be expressed in terms of more basic parameters:

$Ts = Ne/Re$ where
Ne = number of events generated
Re = speed of the machine (events/unit time)

The number of events, Ne, can be written as:

$Ne = Ng*I*Nc$ where
Ng = size of the problem (number of gates)
I = intensity (fraction of gates that change state in an average clock cycle)
Nc = number of clock cycles in the simulation

The speed of the machine (Re) is a function, fe, of the problem size (Ng), the intensity (I), and the architecture (hardware/software) (A), of the machine.

$Re = fe(Ng, I, A)$

P, the performance measure, can now be written as:

$$P = Ng*I*Nc*[Cm + fw(Ts)] / fe(Ng, I, A)$$

Thus, we can compare hardware simulators in terms of P if we are given

1) problem characteristics (Ng, I, Nc)
2) machine characteristics (A, fe)
3) machine costs (Cm)
4) waiting time costs (fw)

For example, suppose the problem characteristics are Ng = 10,000 gates, an intensity of I = .1, and Nc = 1,000 clock cycles [ZYCA83]. Then, the number of events generated in a simulation cycle is $Ne = Ng*I*Nc = 1,000,000$ events. When the waiting time cost is neglected, the performance measure P can be computed for several existing systems.

Consider two systems: 1) the TEGAS simulator running on a VAX 11/780, and 2) the ZYCAD LE-1001 logic simulation machine. Table 3 compares the P values for the two systems and shows that the ZYCAD system has a higher performance measure than TEGAS running on a VAX 11/780. The machine cost per unit time Cm was computed by amortizing the cost of the hardware and software over a five-year period assuming two shifts per day and 200 days per year of operation. In estimating the hardware/software costs, typical values were used assuming only cost recovery.

System	Cm ($/hour) Hard-ware	Soft-ware	Re (events/sec)	P ($)
TEGAS-VAX 11/780	40	12	5,000	2.80
ZYCAD LE-1001	48	0	500,000	.03

Table 3: A Comparison of Two Simulation Systems

5. A Review of Logic Simulation Machines

Four hardware logic simulators will be examined in detail in this section and their location in the taxonomy structure will be determined. Each of these machines has been reported in the literature and three of them have been manufactured. A brief summary of simulator features is provided, and since the amount of information available on each feature varies, some descriptions are more complete than others.

5.1. The Yorktown Simulation Engine - IBM

The Yorktown Simulation Engine (YSE) was first described in a series of three papers presented at the 1982 Design Automation Conference [PFIS82A,DENN82,PFIS82B]. The architecture is shown in Figure 5 which was reproduced from [DENN82]. As indicated in this figure, the YSE can have up to 256 processors (MM), plus an array simulator used for emulating main memory, cache, or register files. The processors communicate with each other during simulation over a high speed crossbar network of size 256 x 256 x 3 bits. The YSE models four logic states (0,1, Undefined(U), and high impedance(Z)). A logic state (2-bits) plus a parity bit is transferred via the crossbar. There are no provisions for logic strengths; thus switch level simulation requires the user to provide a gate level equivalent description. The design supports only unit delay simulation. The simulation algorithm is based on making multiple passes over the combinational network and propagating signals through a single element level. Thus, gates are evaluated independent of their activity.

If the logic design is implemented so that it contains gates in identifiable ranks (e.g., in a PLA, the AND gates would all be considered in Rank 1 and the OR gates in Rank 2), then the execution speed can be increased by evaluating the gates in a given rank only once during a simulation cycle. This "rank order" technique can only be used with unit delay simulation. For example, suppose a network contains N logical elements and elements are organized by rank with a total depth (i.e., number of ranks) equal to D. Then, a machine which does not use the rank order technique would need to make NxD evaluations to determine the next state of the network. With the use of rank ordered simulation the YSE reduces this to N by treating each gate in a given rank only once.

The machine has a total gate capacity of 2 million gates and can evaluate approximately 3 billion gates per second. The characteristics of a gate (or element) are specified during initialization by loading the appropriate processor memory with a truth table that describes the logic function of the element. Each processor has an instruction memory that contains 8K of 128 bit instructions.

Before beginning the simulation, the logic design is partitioned into the 256 processors (2 partitioning algorithms are described in reference [PFIS82B]) and a partition is assigned to each processor. Each processor has its own list of instructions that have been prepared by the host computer prior to simulation. The host computer determines the necessary interconnection configuration for the crossbar switch and also provides this to the processors.

Note that another logic simulation machine was developed somewhat earlier by IBM San Jose. This effort is reported in [BURG83].

5.2. The Hardware Logic Simulator - Nippon Electric Corporation

The HArdware Logic (HAL) simulator was described in the Spring 1983 COMPCON [KOIS83], and further information was given at the June, 1983 Design Automation Conference [SASA83]. Its basic configuration is shown in Figure 6 which was reproduced from reference KOIK83.

The HAL is similar to the YSE configuration in that it supports only unit delay, has a global clock, has multiple processors (29 + 2 special processors for simulating memory), and has multiple lists. Its major difference, beyond the smaller number of processors, is that it maintains separate event lists in each processor and evaluates only those elements that have input signal changes. This is in contrast to the YSE which evaluates gates independent of their activity. Because only active elements are evaluated, the connectivity and interconnection network path is not computed statically prior to simulation, but is determined dynamically during evaluation and is distributed across the processors via a banyan network. Thus, output changes from gate i in processor m are transmitted to gate j in processor n using a store and forward protocol. Arbitration at the network input handles those cases where two or more processors are requesting the same output port (i.e., input to the same element). The banyan network is of size 32 x 32 with a 23 bit data/control path.

Each processor can model up to 10,000 gates for a total capacity of nearly 300,000 gates. Processing speed is about 1.2 billion gate evaluations per second.

5.3. Bell Telephone Laboratories Hardware Simulator

An architecture for a proposed hardware simulator was presented and analyzed in the December, 1982 Bell Systems Technical Journal [LEVE82]. A subsequent article describing the application of this architecture for fault analysis appeared in LEVE83. Although this architecture has not been taken to the fabrication stage, recent discussions with one of the authors (Levendel) indicates that an implementation is still being considered.

A block diagram of the proposed simulator is shown in Figure 7 and was reproduced from LEVE82. This architecture has multiple processors and two types of interprocessor communication. An algorithm that can be used to partition the network is described.

The processors are all commercial microprocessors (examples using the Intel 8086 and the Am29116 are described). Those processors that model elementary gates are termed "simple evaluators", and those that model higher functional elements (e.g., an adder) are termed "functional evaluators". A crossbar is used for communication between simple evaluators because simple evaluations are carried out rapidly and information transfer between simple evaluators requires a high bandwidth channel. A conventional parallel bus is used for communication between function evaluators because they take approximately 30 to 50 times longer than simple evaluators and information transfer requires a low bandwidth channel. Since this is a proposal for a simulator rather than a description of an implementation, only a theoretical evaluation of the architecture is presented in the paper. For example, curves (see Figure 18 of [LEVE82]) are developed illustrating how the communication time and processing time (per active element) are affected by the number of processors and by the type of interconnection network (bus or crossbar).

The simulator differs from the IBM YSE and the NEC HAL machines in that it can simulate systems in which the propagation delay of elements is not limited to unit delay. In the Bell simulator, general LH and HL delays can be used (e.g., 15 ns, 7 ns).

Based on various processor and interconnection network assumptions, estimates of machine performance are presented in LEVE83. For example with 256 processors, the authors compute that processing rates of 10 million events per second could be achieved.

5.4. The Zycad Logic Evaluator

At the time of this writing, it appears that there is only one hardware logic simulator that is available on the commercial market (although other companies are rumored to have active projects). This is the Zycad Corporation's Logic Evaluator (LE 1001 and LE 1002) [ZYCA83]. A block diagram of this system is shown in Figure 8 which was reproduced from a "new product" description in IVER82.

The Zycad architecture is basically the same as the Bell architecture, with the exception that no crossbar switch is used. There are a maximum of 16 processors which communicate via a single high speed (250 megabits/sec) bus. Communication to the host computer is via a slower bus. The logic system to be simulated must be described in a language called ZIF but no compilation is needed (as with YSE and HAL). The system can model 3 logic levels (0,1,X) and 3 logic strengths (Strong, Resistive, Weak). Thus, both gate level and switch level simulations can be performed as well as modelling buses, tristate, and open collector configurations. Since each processor can accommodate 65,000 three input gates, 1 million gates can be simulated with 16 processors. An event list is used in each processor and a rate of approximately 16 million events per second can be achieved with 16 processors.

5.5. Logic Machine Summary

Table 4 summarizes several of the important features of the four logic evaluators that have been discussed above and their place in the architecure taxonomy is indicated. In addition, the performance of a representative single machine, software based simulator [SZYG72] is also listed.

6. Conclusion

The initial steps in providing a systematic approach to evaluating alternative DES machine architectures has been taken. A taxonomy which highlights the implementation of the main DES features has been presented. Existing logic simulation machines are easily incorporated into the taxonomy. A performance measure which can be used to rank alternative architectures was presented. Several existing machines were compared using this measure. Finally, the detailed structure of three contemporary logic simulation machines were reviewed.

Given the high costs associated with VLSI level logic simulation and fault analysis, it seems clear that a host of logic simulation machine proposals will be forthcoming in the near future. These standalone and attached processor simulators will revolutionize logic design, making the task largely an interactive process.

References

ABRA82 Abramovici, M., Levendel, Y. H., and Menon, P. R., "A Logic Simulation Machine," Proceedings of the 9th Annual Symposium on Computer Architecture, April 1982, pp. 148-157.

BART80A Barto, R. L., et al., "Architecture for a Hardware Simulator," Proceedings of the IEEE Conference on Circuits and Computers, October 1980, pp. 891-893.

BART80B Barto, R., and Szygenda, S. A., "A Computer Architecture for Digital Logic Simulation," Electronic Engineering, 55-624, September 1980, pp. 35-66.

BURG83 Burggraff, T., et.al., "The IBM Los Gatos Simulation Machine Hardware," Proc. IEEE Inter. Conf. on Comp. Design (ICCD'83), October 1983, pp. 584-587.

CHAN81 Chandy, K. M. and Misra, J., "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," CACM, 24.4, April 1981, pp.198-206

COMF81A Comfort, J. C., "Discrete-event Simulation Computer with Event-set Manipulation Performed by Pipelined Set of Microprocessors: Design Using Simulation," Winter Simulation Conference, 2, 1981, pp. 591-597.

COMF81B Comfort, J. C., "The Simulation of a Microprocessor Based Event Set Processor," Simulation Symposium, 1981, pp. 17-33.

DENN82 Denneau, M. M., "The Yorktown Simulation Engine: Architecture and Hardware Description," ACM IEEE 19th Design Automation Conference, June 1982, pp. 55-59.

FLYN66 Flynn, M. J., "Very High-Speed Computing Systems," Proceedings of the IEEE, 54, pp. 1901-1909.

HOWA83A 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), October 1983, pp. 580-583.

HOWA83B Howard, J. K., et.al., "Using the IBM Los Gatos Logic Simulation Machine," Proc. IEEE Inter Conf. on Comp. Design (ICCD'83), October 1983, pp. 592-594.

IVER82 Iverson, W.R., "Modular hardware simulates logic," Electronics, 55, 15, July 28,1982, pp. 125-126.

JEFF83 Jefferson, D., "Virtual Time," Proceedings of the 1983 Parallel Processing Conference, pp. 384-393.

KEND53 Kendall, D. G., "Stochastic Processes Occurring in the Theory of Queues and their Analysis by Means of the Imbedded Markov Chain," Annuals of Mathematics and Statistics, 24, 1953, pp. 338-354.

KOHN83 Kohn, J., et.al., "The IBM Los Gatos Simulation Machine Software," Proc. IEEE Inter. Conf. on Comp. Design (ICCD'83), October 1983, pp. 588-591.

KOIK82 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.

LEIS81 Leiserson, C.E., Area-Efficient VLSI Computation, PhD Thesis, Dept of Computer Science, Carnegie-Mellon Univ., Pgh. Pa., October 1981

LEVE82 Levendel, Y. H., Menon, P. R., and Patel, S. H., "Special-Purpose Computer for Logic Simulation Using Distributed Processing," The Bell System Technical Journal, December 1982, pp. 2873-2909.

LEVE83 Levendel, Y.H., Menon, P. R., and Patel, S. H., "Parallel Fault Simulation Using Distributed Processing," The Bell System Technical Journal, December 1983, pp. 3107-3137.

PEAC79 Peacock, J. K., et.al., "Distributed Simulation Using a Network of Processors," Computer Networks, 3,1, February 1979, pp. 44-56.

PFIS82A Pfister, G. F., "The Yorktown Simulation Engine: Introduction," ACM IEEE 19th Design Automation Conference, June 1982, pp. 51-54.

PFIS82B Pfister, G. F., and Kronstadt, E. P., "Software Support for the Yorktown Simulation Engine," ACM IEEE 19th Design Automation Conference, June 1982, pp. 60-64.

ROBI84 Robinson, A. L., "One Billion Transistors on a Chip," Science, 223, January 20, 1984, pp. 267-268.

SASA83 Sasaki, T., Koike N., Ohmori K., and Tomita K., "HAL: A Block Level Hardware Logic Simulator," 20th Design Automation Conference, June 1983, pp. 150-156.

SZYG72 Szygenda, S. A., "TEGAS2 - Anatomy of a General Purpose Test Generation and Simulation Syetem for Digital Logic," Proceedings of the 9th Design Automation Workshop, Dallas, Texas, June 26-28, 1972, pp. 116-127

VANA71 Van Ausdal, A. W., "Use of a Boeing Computer Simulator for Logic Design Confirmation and Failure Diagnostic Programs," Advanced Astronautics Science, 29, June 1971, pp. 573-594.

ZYCA83 Zycad Corp., "The ZYCAD Logic Evaluator: Product Description," Zycad Corp., N. Roseville, MN. , 1983

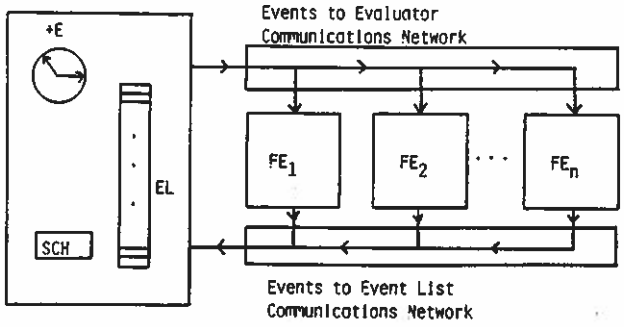

Figure 1: The EI/GC/SL/MM Architecture

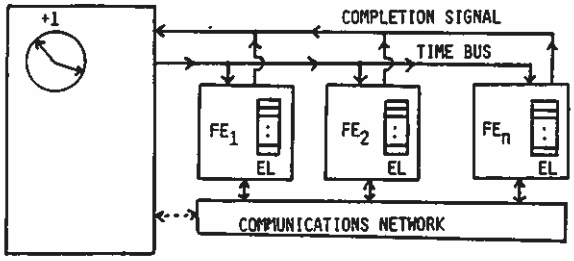

Figure 2: The UI/GC/ML/MM Architecture

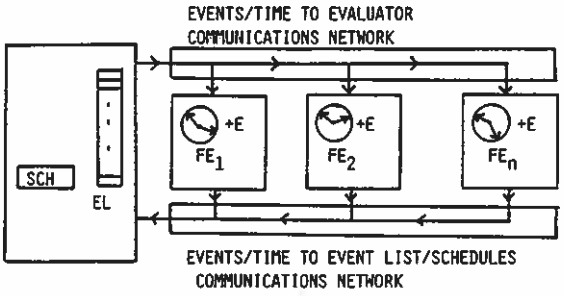

Figure 3: The EI/LC/SL/MM Architecture

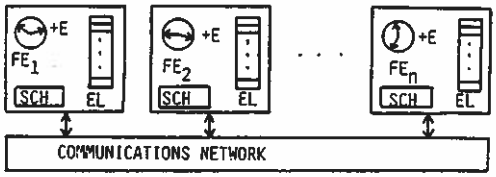

Figure 4: The EI/LC/ML/MM Architecture

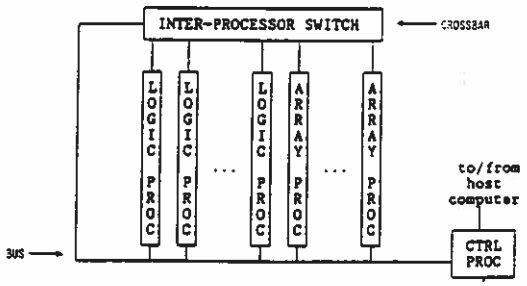

Figure 5: IBM YSE Architecture (from DENN82)

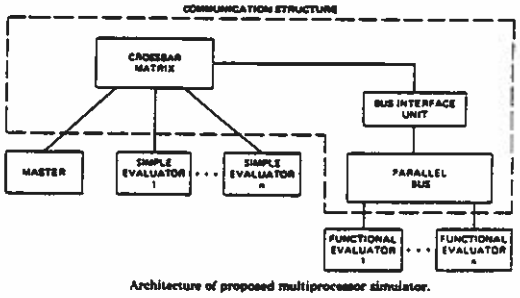

Figure 7: Bell Architecture (from LEVE82)

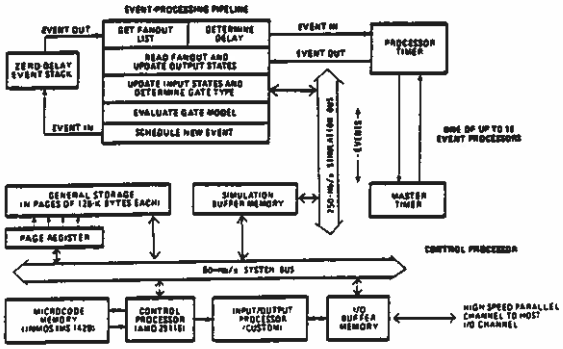

Figure 8: ZYCAD Architecture (from IVER82)

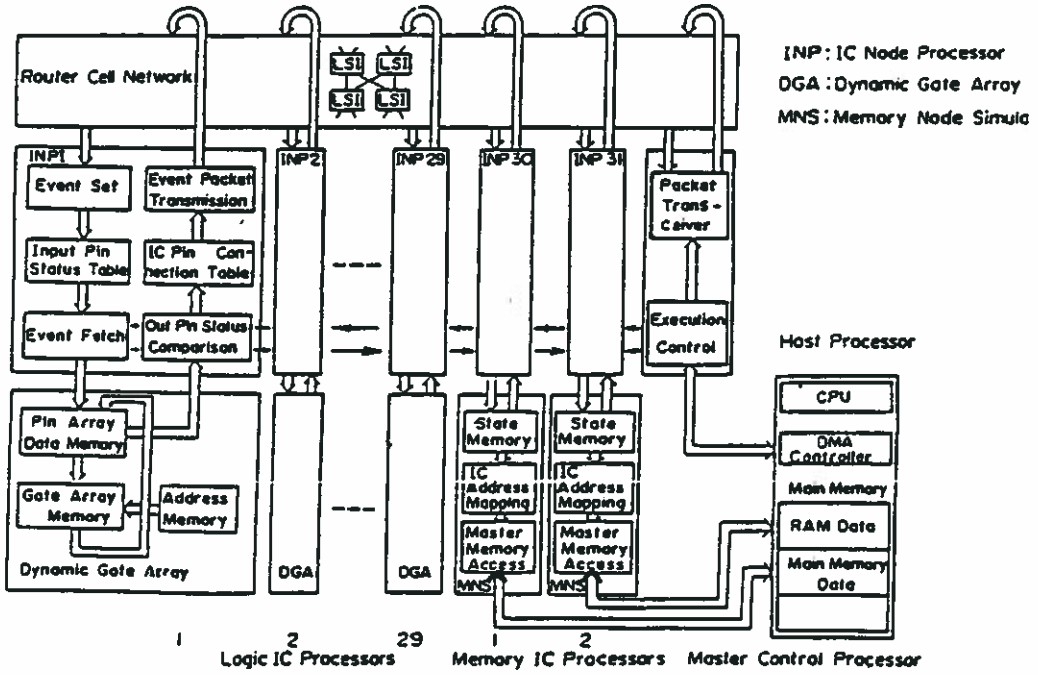

Figure 6: NIPPON Electric Corporation Architecture (from SASA83)

	SOFTWARE		HARDWARE			
	Mainframe Computer	Super Mainframe Computer	Bell	IBM	Nippon	ZYCAD
Function (Event) Evaluation	---	---	Micro-processor	Special Pur-pose Proc.	Dynamic Gate Array	Special Pur-pose Proc.
Number of Processors	1	1	256	256	30	16
Communications Network	---	---	Bus & Crossbar	Crossbar	Banyan	Bus
Preprocessor Complexity	---	---	Moderate	Very High	High	Moderate
Speed (Events/Sec)	5-10K	90K (Cyber 176)	3-18M	3B*	1.2B*	1M to 16M
Capacity (gates)			?	2M	3M	1M
Capacity (memory)			?	?	2M Bytes	3M Bytes
Test Vector/Fault Capabilities	Y/N	Y/N	N	N	N	Y
Status	Built	Built	Paper Study	Prototype Built**	Under Construc-tion	Several Delivered
Category	EI/GC/SL/SM	EI/GC/SL/SM	EI/GC/ML/MM	UI/GC/ML/MM	UI/GC/ML/MM	EI/GC/ML/MM

* Subject to interpretation (gate evaluations/second)
** Excluding Memory (Array Simulation)

Table 4: Current Logic Simulator Characteristics