A model and implementation of a universal time delay simulator for large digital nets

by STEPHEN A. SZYGENDA, DAVID M. ROUSE and EDWARD W. THOMPSON*

University of Missouri
Rolla, Missouri

INTRODUCTION

Although simulation of logic circuits has been attempted in the past, only those which simulate completely combinational circuits have performed with any degree of success for various types of logic. This is primarily due to the number of simplifying assumptions that can be made for combinational circuits. For example, in combinational circuits, one need not consider timing of the signals, since any given input vector will always propagate to a stable state value. Also, since purely combinational circuits do not contain internal states, the user need not define or initialize these values, as must be done (and done meaningfully) for sequential circuits. A systems study of simulation and diagnosis for large digital computing systems has been performed. The results of that study have led to the implementation to be described. The model, which has been adopted, permits the user to select the level of detail most appropriate to his requirements, and thus not hamper him with overly restrictive assumptions. The advantages of this model are the following:

1. It does not require the specification of feedback loops in sequential circuits.
2. The ability to reset all feedback lines to any value at any time is not required.
3. It provides for the detection of hazards and races.
4. Simulation is not restricted to particular types of circuits.
5. Besides having the capability of simulating gate level circuits, it can also simulate at a functional level. This results in a savings of time and storage, permitting the simulation of large circuits.

Actually, three models are presented, two being subsets of the other. It is felt that these models will be useful tools in analyzing logic circuits, generating tests, and providing experience in determining a desirable level of detail for simulation.

In the next section of this paper, features are defined which appear to be desirable for a general purpose system simulator. This is followed by a description of the means utilized to achieve these desired features. In particular, a detailed discussion of the models adopted for the simulation is provided. It is felt that the heart of any simulator is the basic simulation model. The effectiveness of the simulation is directly related to the ability of the model to accurately describe the physical systems being simulated. Therefore, the adopted models are of extreme importance and will be described in sufficient detail to substantiate their effectiveness.

A description of the simulator implementation follows the simulation model discussion. (This implementation is used in a total simulation and diagnosis software system.) Modes of operation and simulation optimization techniques are also described. Since a table driven simulator was implemented, a discussion of its implementation will be given.

DESIRED FEATURES OF A SIMULATION MODEL

A primary goal of this simulation package is that it possesses the ability to simulate any of the common modes of logic operation. The handling of asynchronous sequential circuits presents the most difficulty, in that circuit timing must be accurately described. This rules out the unit delay assumption used by many existing simulators (in particular, all space-leveled, compiler-driven simulators). Therefore, this model allows a variable time delay for the elements being simulated.

In the model used by Seshu, which is frequently used for sequential circuits, the only race analysis performed is that of checking feedback line values. All
feedback lines are assumed to be broken at some point and a race is declared if, during any pass through the circuit, more than one feedback line changes value. One feedback line at a time is declared a winner and its value is then propagated through the circuit to determine its effect on the outputs. The race becomes critical if different stable states are reached, depending upon the order in which the feedback values were changed.

This type of model has three major deficiencies:

1. The system is very sensitive to the selection of feedback lines and the point at which they are broken.
2. A few feedback loops can produce numerous races, requiring excessive time and storage for evaluation.
3. Race analysis performed in this manner does not detect static or essential hazards.

Also, many of the races which may be detected are physically impossible. One of the goals of the model being presented will be to overcome these deficiencies.

The model used by Seshu also makes the assumption that feedback lines can be reset to any desired value at any time, even in the presence of faults. It is felt that this assumption is not necessarily valid, and the assumption is not made in the model being presented.

The speed of simulation ( compilation and execution) is an important measure of a good simulator. Therefore, maximum speed is also an objective. However, the requirement for maximum speed can, and should, be sacrificed for more accurate simulation results, when required. Therefore, the speed of this simulator is a function of the level of detail required of the simulation.

Another simulation goal is the use of a minimum amount of storage. The importance of this goal is twofold. First, minimization of storage requirements will enable the simulator to handle larger circuits; and second, its use will not be limited to large computing facilities. Appropriate segmentation can help reduce the storage requirements, but only with a sacrifice in speed. For these reasons, both minimization of storage requirements and software system modularity are of primary concern.

The simulation package should be as flexible and versatile as possible, but at the same time it should be easy to understand and implement. Therefore, one need only be concerned with those options which are pertinent to the task at hand. For example, the option of simulating faults is available with or without race analysis.

Another additional goal is the capability of easy adaptation of new element types. Hence, one need only supply a new description for element evaluation, and the program then adds to, or updates, the existing specifications. Therefore, this method should not require a large amount of reprocessing.

The system should also have the capability of multiple fault insertion, as well as fault insertion on inputs and outputs of functional modules.

MEANS OF ACHIEVING THE DESIRED FEATURES

There are two approaches that can be taken for digital simulation. One is the approach of modeling the entire circuit as one unit and, therefore, with one macro-model. This would be the technique used for a compiled simulator, since a compiled simulator levels and transforms the circuit into a form that can be dealt with collectively. The other approach is that of modeling the entire circuit by breaking it into smaller blocks, which can be individually modeled according to their type. This approach can be accomplished with a table driven simulator. A table driven simulator deals directly with elements, in that the circuit description is explicitly specified during simulation. Therefore, it determines, during simulation, what elements are to be evaluated next and then uses one generalized routine to evaluate all elements of any one type.

The second method was chosen for this simulator since the first contains undesirable features, such as the need of pre-leveling, location and breaking of feedback loops, inability to handle various sequential circuits effectively, etc. Although the second approach is more general, and can handle a large majority of circuit types, it does have the disadvantages of being slower and requiring more storage for certain cases. Therefore, it is these latter two disadvantages that are of great concern in this simulator structure. Solutions to these problems will be discussed later.

The ability to simulate sequential circuits is inherent in a table driven simulator, in that it dynamically levels the circuit during simulation. This is done by determining, from the circuit description, what elements need to be reevaluated due to a change in the value of a signal.

The ability to simulate asynchronous circuits relies on the ability to accurately represent the time associated with evaluation of signal values. By including the propagation time of each element in its description, and then using this parameter during simulation to order the evaluation procedure, this time factor can be accurately represented. This means that one must have the ability to accept propagation delays of different lengths instead of making a unit delay assumption.

Another side feature of a table driven simulator is that of not being required to consider feedback lines
by special, cumbersome, and inaccurate methods. These methods include explicit specification of what lines are feedback lines and the ability to reset these to any value. These two conditions are produced when an attempt is made to represent an entire sequential circuit by one model, as is done in a compiled simulator structure. The feedback specification problem is not present in a table driven simulator, in that no special case need be made concerning feedback lines. The latter case, which is commonly referred to as a reset assumption, is avoided since simulation occurs directly from a given accessible state without requiring reinitialization of the state during simulation.

As mentioned earlier, speed and storage will be a primary concern in this simulator. Three techniques will be employed to reduce these problems to an acceptable level. They are: (1) selective trace simulation, (2) parallel simulation, and (3) functional simulation.

Selective trace is a technique used in conjunction with table driven simulators which provides the ability to evaluate only those elements which have a potential of changing. For example, one need not reevaluate a gate output if all the input signals are the same as they were when it was last evaluated. Thus, simulation becomes a process of tracing changes, and their effects, through the circuit.

Signal values can be stored in one of three manners. First, one bit of a machine word could be used to represent the value of a signal. Second, each bit of a machine word could be used to represent a different signal value. Third, each bit of a machine word could represent different values of the same signal for different input vectors or different fault conditions.

The first of these three techniques is extremely inefficient in storage handling. The second approach is hard to execute in Fortran (the implementation language for the system) since it would require bit manipulation. Therefore, the third approach was taken. For this technique, \( n \) different input vectors (where \( n \) is the number of bits in the machine word length), or fault conditions, can be simulated with the same speed and storage required for the first approach. This is referred to as parallel simulation, since \( n \) unique simulations occur in parallel. The effect of this approach is to divide the required simulation time by a factor of \( n \).

Another important implementation feature is called functional simulation. This is the grouping of a number of logic elements together and then expressing the group by its function. Thus, one need only store and evaluate the function in order to simulate the represented logic. An example of functional simulation would be the representation of an adder by storing and executing a simple add instruction, instead of storing and executing the large number of logic elements used to form an actual adder circuit. Therefore, it can be seen that functional simulation enhances simulation speed and reduces storage. The ability to implement functional simulation is compatible with a table driven simulator structure, since it models the circuit by modeling elements, regardless of the evaluation procedure used to model the element. For this reason, changing or adding element types is a simple task which involves changing only the evaluation procedure and its respective pointer.

Fault insertion is also simplified since it now becomes a matter of simply providing elements having the same characteristic as a faulty element.

When faults are automatically inserted, fault collapsing is used to reduce the number of possible faults. This is done by inserting only one fault of a group of faults which always produce the same outputs for any input combination. For example, all stuck-at-0's on the input of an "and" gate appear the same as a stuck-at-0 on the output of that gate. Therefore, the stuck-at-0's on the input need not be simulated if a stuck-at-0 on the output is simulated, since they all produce the same response and are therefore repetitious.

To further increase the accuracy of simulation, an ambiguity interval can be associated with each signal. This is a result of an inability to specify exactly when a given signal will actually make a transition from one state to the next. The requirement of an ambiguity interval comes from the fact that gates of the same type could have different propagation times. Therefore, the time delay of a gate would be represented as a minimum value plus an ambiguity region. By considering this ambiguity, race and hazard analysis can be performed during simulation.

It is not always desirable to perform race analysis, since it requires greater simulation time and storage. Therefore, there are different modes of simulation, including straight simulation and simulation with race analysis. Straight simulation will be referred to as the Mode 1 simulator, and simulation with race and hazard analysis will be referred to as the Mode 3 simulator. In order to obtain modularity in simulation and consistency in circuit modeling, Mode 1 will be implemented as a subset of Mode 3. A Mode 2 simulation is also available. This is a three valued simulation which can be used for simulation initialization. It indicates and propagates information as to whether or not a signal is defined at a given time. This is accomplished with the use of a third value. An example of the Mode 2 simulation is given in Figure 1. Here, each signal can either be 1, 0, or I (I indicates Indeterminate). Before time 0, all signals are unknown and, therefore, in an indeterminant (D) state. At time 0, A and B are changed to a 1. As a result of this change in A and B, the value of C and D must be reevaluated. C is evaluated to be 1,
Figure 1—Mode 2 simulation

at $t = 5$. The change in $A$, at $t = 0$, causes a reevaluation of $D$. However, since $C = I$ at $t = 0$, then $D = I$ at $t = 10$ due to the change in $C$ having not yet propagated to $D$. An evaluation table for $C$ and $D$ is indicated in Figure 2. However, the change in $C$ at $t = 5$ causes $D$ to be evaluated again. Hence, $D$ becomes 1 at $t = 15$ since both $C$ and $A$ are known.

An example of the Mode 3 simulation is depicted in Figure 3. $A$ and $B$ are input signals of initial value 0 and 1, respectively. $C$ and $D$ are the output of inverters, which have a propagation delay of 4 and an ambiguity of 2. $E$, which is the set signal to an $S-R$ Flip Flop, is the logical "and" of $C$ and $D$. At $t = 0$, $A$ changes to 1 which produces a change in $C$ to 0 through an ambiguity region from $t = 4$ to $t = 6$. This means that the change of $C$ could occur sometime between $t = 4$ and $t = 6$. If $B$ changes at $t = 1$ (possibly due to an ambiguity in the circuit feeding $B$), then $D$ changes value as indicated. Notice that the value of $E$, as a result of $C$ and $D$ being 1 between $t = 5$ and $t = 6$, is a potential error region. This, along with ambiguity and minimum delay of the "and" gate, produces the results indicated for $E$. Since $E$ is setting the Flip Flop, this potential error region sets the Flip Flop to a potential error value, which is the resulting state of $Q$. From this example, it can be seen how the ambiguity in propagation delay is handled, as well as how Mode 3 simulation handles potential error regions and how these potential error regions can result in essential hazards.

In general, the mode 3 simulator is used to propagate potential error regions to provide determination of the existence of essential hazards. The unique characteristic of the Mode 3 simulator, that does not exist in the other modes, is that it carries regions instead of

![Logical "and"

![Logical "or"
single values. A technique similar to this has been described by D. L. Smith. A spread is depicted with the use of a New Value (NV), as well as a Current Value (CV), along with the Potential Error value (PE). During simulation, this ambiguity region is represented by simulating the earliest possible transition point (the CV) and the latest possible transition point (the NV). The potential error is then a logical function of the CV, NV, and the PE. With respect to the simulator, this evaluation procedure simply appears as another element type.

Sequential logic circuits containing global feedback loops are extremely difficult to simulate, even for the simplest of design philosophies. Whether accurate simulation is achieved, most often depends upon the design employing a special type of sequential action, which is consistent with the particular simulator being used, or the person simulating the circuit must have a very intimate understanding of the circuit operation. These two circumstances are more often the exception rather than the rule. In many design environments, as a result of practicality, as well as necessity, these conditions are rigidly forced upon the user by their simulation structures. In order to alleviate this problem, the largest possible user flexibility was a goal for this simulator. One of these degrees of freedom is presented in the following example, which shows how race analysis of an asynchronous sequential circuit can be performed.

A Transition Table and State Table are given in Figure 4 for a simple sequential circuit which has been implemented in Figure 5. A race can be seen to exist between states C and D for the input vector 01 and stable state B. Whether this race is a result of improper design, characteristics of a circuit containing a faulty element, or an intentional risk, is unimportant. The important thing is that the simulation of this circuit is capable of revealing sufficient information to determine how the circuit will, or could, act when physically implemented. Figure 6 shows the response of X1 changing from a 1 to a 0 according to the unit delay assumption (the circuit is in stable state 01, with \( X_1X_2 = 11 \)). From Figure 6, it can be observed that the circuit makes a transition from state B to state D, a seemingly definite and satisfactory result.

By considering the circuit response as indicated in Figure 7, which uses more accurate delay time information (as given by the minimum delay in Figure 5) for each gate, it is observed, through this type of simulation, that all isn’t as simple as indicated by the previous simulation. It can be seen that, as the accuracy of propagation time becomes closer to that of the physical circuit, the race between states D and C comes closer to actuality. It appears here as the simultaneous transition of \( Y_1 \) and \( Y_2 \). This type of simulation is one which might be performed by Mode 1 simulation.
The curiosity raised by using a more accurate representation for the delay time can be satisfied by also considering an ambiguity time (as indicated in Figures 5 and 8) associated with each gate. The state variable \( Y_1 \) could change anywhere between \( t = 10 \) and \( t = 13 \). This is a result of the possible variation in time delays.

Figure 5—Example sequential circuit

Figure 6—Results for circuits with unit time delays

Figure 7—Results for circuits with variable time delays

Figure 8—Results for circuits with variable time delays and ambiguity
of the inverter, “and” gate, and “or” gate, along the propagation path $X_1$. Similarly, $Y_2$ could change between $t = 10$ and $t = 12$. The value of $F$ is essentially the “and” of $Y_1$ and $Y_2$, since $X_1$ appears as a constant 1. However, the “and” of the two ambiguity regions for $Y_1$ and $Y_2$ is not only another ambiguity region in $F$, it is also a potential error region as well. In actuality, $F$ may or may not produce the momentary 1 spike between $t = 14$ and $t = 17$. Note that a transition region is concerned with the question of when a transition will take place. However, a potential error region is concerned with whether a transition could take place.

Thus, from this example, it can be seen that a variation in propagation delay is enough to produce a critical race condition from a seemingly stable design. This condition is detected in Mode 3 simulation when the potential error flag is set for the state variable $Y_2$, as indicated by the shaded area in Figure 8.

This example demonstrates some of the problems that could be encountered when simulating sequential circuits. It also shows how these problems can be handled through the various modes of simulation available in this simulator.

To use these three modes of simulation, one need only specify which mode is desired, so that the appropriate evaluation routines will be used. Since the only difference is in the evaluation routines, the same circuit description can be used for Mode 1, 2, or 3 simulation.

One important feature of this approach, with respect to race analysis, is that race analysis occurs concurrently for nested races. Therefore, only one simulation must be performed for $n$ nested races, as compared to as many as $2^n$ simulations for other approaches.

SYSTEM IMPLEMENTATION

The first major implementation decision was the choice of a programming language in which the simulator would be written. Assembler, Fortran and PL/1 were considered. Utilizing an assembler language could result in a little faster execution, with somewhat less storage required. However, Fortran was chosen since it would be easier to implement and is considerably more machine independent. Although PL/1 has some seemingly desirable features, they were sacrificed for the more commonly acceptable Fortran and the small decrease in execution time and storage. It was also desirable for this simulator to be acceptable for use on smaller machines, which have limited storage and compiler facilities. For these reasons Fortran was considered more desirable than PL/1.

The basic simulator consists of three tables, the Time Queue Table (TQ), the System Description Table (SDT), and a table which contains the Current Value of each signal (CV). The time queue table contains events that occur at time $t$, where $t$ is the index of the time queue table. The system description table contains pointers to the evaluation routines used to determine the output values of the element, pointers to the fan in and fan out, and also contains the number of fan outs for each signal.

Using these three tables, simulation is performed as follows:

1. All values that exist in the time queue, at the current simulation time, are transferred to the current value table, thus causing any projected changes in value to take effect.
2. If the new value entered in the current value table is different from the old value, then all elements that are immediately affected by this change are reevaluated. (This is accomplished by following a fan out list.)
3. The results of these reevaluations are projected into the time queue at the current time plus the minimum propagation delay of the signal.
4. The current time is incremented until an entry is found in the time queue, and then the process is repeated again.

This process is restated in a flow chart form in Figure 9.

In addition to this basic structure, other tables are used for optimization of both speed and storage. For example, functions are evaluated indirectly, via the Function Description Table (FDT), which also specifies additional parameters used in the evaluation routine. These parameters are: function type, time delay, number of inputs, and bus length. This permits a minimum number of evaluation routines that must be provided for simulation. By use of the FDT table, the same routine would be used for a 2 input “and” gate with a time delay of 5, as would be used for an 8 input “and” gate with a time delay of 8.

To keep the size of the TQ from becoming prohibitively large, a Macro Time Queue Table (MTQ) was implemented to store events which occur at large time intervals, relative to the largest propagation delay for any gate. The Time Queue was then made cyclic in coordination with the MTQ, where each cycle of the TQ advances the MTQ one step.

Some gate level elements, such as flip-flops, as well as functional elements, have multiple outputs. To be able to simulate this type of element, an additional entry is provided in the SDT Table, which is used to chain the output together.

An ultimate goal of this system simulator is to
possess capability of simulating elements other than actual gates, such as functional modules. Since functional modules are just as apt to be dealing with busses as with single lines, the ability to specify bus lines collectively would make functional module specification an easier, as well as a more meaningful task. For this reason, the capability of specifying busses collectively has been implemented with the use of a paging scheme, where Bus Value (BV) is a group of pages and Bus Value State (BVS) is a table which indicates use, length and location. Therefore, the CV of a bus is an indirect pointer to a page, which contains the actual values of the bus signals.

It can be seen that the System Simulator is independent of the type of function used to evaluate the output signals of the element. For this reason, any type of element can be simulated which can be described in a discrete value system. Therefore, the power of module simulation is directly proportional to the kinds of module descriptions which are permitted.

In an attempt to cover the widest range of module descriptions, five types of descriptions are permitted. These are: (1) gate elements, (2) standard functional modules, (3) compiled gate modules, (4) computer design language modules, and (5) Fortran modules.

Evaluation procedures for gate elements are defined by gate type. Thus, one can specify gate elements, to the system simulator, by giving the gates fan in and fan out. This would be used primarily for circuits which could be specified at the gate level.

Similar to the gate modules are the standard functional modules, in that they are predefined routines which can be used by specifying an element as a standard functional module type. An example would be an n-bit 2's complement adder. Thus, to define a complete adder, one need only give its function type, the number of bits being added, and its time delay. This feature was provided in order to eliminate much of the trivial task of redefining common functional modules, and also to provide faster, more efficient, system routines.

To permit the initial design specification to become the initial functional representation for design verification on a macro-level, a computer design language is allowed for module description. This is done by compiling this description into an equivalent Fortran subroutine, which has the inputs equivalenced to the appropriate current value table, and the outputs queued in a scratch array. In order to provide sequential control modules at a design language level, sequential modules can be described in a flow table form of expression. Thus, sequential control variables can be generated by such a sequential module.

Also, to provide the capability of compiled simulation, one can specify modules at the gate level and have these transformed into compiled code for simulation. This is desirable for purely combinational modules which can be simulated faster, or with less storage, in a compiled simulation fashion.

Fortran module descriptions can be used as a means of generating new, efficient, standard functional modules. They can also be used to generate special modules.
whose function cannot be described easily by a high level system design language.

To make this system more usable, one must have a means of redefining modules without requiring the reprocessing of the complete system description. This is done by automatically defining boundary elements for each module of the system.

A boundary element is an element which is placed at each input and output of a module to isolate that module from the rest of the circuit. Each boundary element has one input and one output, and the output value equals the input value. Taking this approach permits changes in the module definition without changing descriptions outside the module. This is depicted in Figure 10 where B1, B2, B3, and B4 are boundary elements for module M2. Gate G1 fans out to G2 and B1 when module M2 is expressed functionally. But, if module M2 is expressed at the gate level, without boundary elements, then the fan out of G1 would be to G2, G3, and G4. Without boundary elements, it would be necessary to change the fan out description of G1 (which is outside the module), if the user wants to change M2 to a gate representation. But, with the boundary elements inserted, the fan out of G1 (to G2 and B1) is still the same independent of the type of expression used for M2. This could be of extreme importance, when a number of design groups are using the same high level description for the complete system, while considering their own particular section at the gate level.

For fault insertion, two tables are used. The Fault Table (FT) indicates the type of fault and which leads of the gate with which it is concerned. The Logical Fault Mask Table (LFMT) indicates in which bit (or subject machine) the fault is present.

During Mode 1 simulation, the value of each signal is stored in the full word array CV. The values in the CV are updated by transferring the appropriate value, which is indicated indirectly in the time queue, to the CV, at the time indicated by the time queue description. The same simulation structure exists for Mode 2 simulation, with the use of different evaluation routines for element output evaluation. An Indeterminant Value Array (IV) is used for storing the third value necessary for Mode 2 simulation.

Due to the necessity of being able to process time intervals, in Mode 3 simulation, a few minor modifications must be made in the simulator structure. This, however, is not apparent to the simulator user. Storage must also be provided for the Current Value (CV), New Value (NV), and the Potential Error (PE), along with the more detailed evaluation procedures used to determine these quantities for each element.

CONCLUDING REMARKS

The simulator described in this paper has been programmed in Fortran IV on an IBM 360/50, with 256K bytes of core storage, at the University of Missouri-Rolla. The size of a system that can be simulated is a function of the available memory capacity of the host machine. For 256K bytes of storage, approximately 3000 elements could be simulated, using the 10th phase of the system. (An element can range from simple gate elements to complex functional elements.) This would be reduced to approximately 2000 elements for Mode 3 simulation. For trial simulation runs, a running time of 100 μs/pass·fault·element has been obtained, utilizing Mode 1 simulation. This, however, is fairly circuit dependent.

It is felt that this simulator represents a uniform systems approach to simulation and diagnosis. Versatility of the models utilized has resulted in this capability. The system not only allows various types of simulation for different hardware implementations, but also provides the ability to handle the total system as a collection of subsystems. Thus, each subsystem can be simulated according to the particular type of circuit involved, the requirements imposed upon the circuit, and the most applicable simulation technique for the particular subsystem.

REFERENCES

1 S A SYGENDA

A software diagnostic system for test generation and simulation of large digital systems

From the collection of the Computer History Museum (www.computerhistory.org)