TIME FLOW MECHANISMS FOR USE IN DIGITAL LOGIC SIMULATION

Stephen A. Szygenda*
Cliff W. Hemming*
John M. Hemphill*
Computer Science/Operations Research Center
Southern Methodist University
Dallas, Texas 75222

Summary

Implementation of a system for the simulation of the time domain operation of a deterministic digital logic net involves consideration of problems different from those encountered in simulation of the time domain operation of a stochastic system. Examination of two different simulation time flow mechanisms illustrates how each technique may be applied to the simulation of logic nets. The design goals for a general purpose logic simulator are examined and the implementation techniques used in TEGAS2 are illustrated.

Introduction

Implementation of a system for the simulation of the time domain operation of a deterministic digital logic net involves consideration of problems different from those encountered in simulation of the time domain operation of a stochastic system.[5] Most general purpose simulation systems designed for simulation of stochastic systems cannot take advantage of the deterministic restrictions inherent in the operation of digital logic nets. The operation of a digital logic net is determined by the parameters of the net itself. Hence, some foreknowledge of the operation of the net may be gained by examination of the net's parameters. This knowledge of the net's operation may possibly influence the design of a logic net simulator. This influence is especially felt in the area of algorithms for the movement and control of time in a simulation model. In this paper, we examine some of the aspects of time domain simulation of digital logic nets.

I. Digital Logic Net Simulation

A digital logic net will be defined to be a set of digital logic elements interconnected by signal paths known as lines. A digital logic element is a device whose output or outputs is a function of either its inputs or current state history or both. These signals can possess different discrete values from a finite set of values. The set of values depends on the type of model used for simulation. For example, in a simple two-valued simulation of a logic net, signals may take on values of either zero (false) or one (true). In a three-valued simulation signals may take on values of zero (false), indeterminate (unknown), or one (true). Other simulation models have different sets of values for the signals.

The system being simulated is defined as the digital logic net itself. Any logic element input line that does not originate as the output of a logic element in the system is known as a primary input. Any output line of a logic element that is not used for input to a logic element in the system is known as a primary output. The state of the system is defined by the values of the lines and the states of the logic elements.

In discussing further concepts of logic net simulation the terminology defined by Pritsker and Kiviat will be used.[1] An event is an action which describes how the simulation system changes state. The occurrence of an event causes the system to instantaneously change state. An activity is an action engaged in by an entity or entities as the system changes from state to state. Activities are started and ended by the occurrence of events.

An event in a digital logic net occurs when a line in the net changes to a value different from the value it possessed before the event occurred. Events can originate either internally or externally to the logic net simulation model. Internally created events are originated by changes in the outputs of a logic element in the net. Externally created events are originated by lines in the nets having their values changed at specific times to new values by the person controlling the simulation.

The logic elements in the net can be thought of as permanent entities in SIMSCRIPT II[3] terminology or as facilities in GPSS[4] terminology. The events in the logic net mark

*Supported in part by ONR-N00178-71-C0148, Naval Weapons Laboratory, Dahlgren, Virginia.

the commencement and termination of activities in the logic net. The logic elements in the nets are the only entities that can support activities. An activity in a logic net is the operation of a logic element evaluating the value of its output or outputs on the basis of its current input or inputs and the current state of the logic element. This activity of evaluation is the only process inherent to the logic element. Activity in a logic element is always initiated by a value change on one of the input lines--an event--and is terminated by the logic element presenting its newly evaluated output value or values, after some increment of simulated time has passed. If the new output values differ from the current values, then the activity terminates by causing an event or events to occur, since it may cause a line or lines to change to a new value. If none of the new output values differ from the current output values, then the activity in the logic element terminates without causing an event to occur.

Hence, the operation of the logic net consists of events and activities which occur in relation to the passage of time. Simulation of the operation of the digital logic net requires the creation and monitoring of these events and activities.

II. An Examination of Two Simulation Time Flow Mechanisms

The implementation for driving a simulation model through the time domain is the time flow algorithm.[5] Often the algorithm is referred to as an implementation of the time flow mechanism. It is known that algorithms combining properties of several mechanisms produce more efficient results in certain instances.[9] We return to this point later.

The time flow mechanism governs the simulation of events in the system and controls the flow of simulated time. As a result, the time flow mechanism influences many aspects of the simulation. For example, if the time flow mechanism restricts when events can occur in the time domain, then the simulation model may not present a realistic picture of the system being simulated. Another aspect influenced by the choice of time flow mechanisms is the simulator efficiency. Some time flow mechanisms may execute in less time for a given simulation than other time flow mechanisms because of implementation or computational requirements. For these reasons, we will examine two different time flow mechanisms and their relation to digital logic net simulation.

The first time flow mechanism to be considered is the "next event" mechanism. This can be represented by the algorithm below:

Let $e_1, e_2, \ldots, e_n$ be the events which are scheduled to occur in the system.

Let t_i be the time at which event e_i is scheduled to occur.

Define $E = [(e_1, t_1), (e_2, t_2), \ldots, (e_n, t_n)]$ to be the set of pairs (e_i, t_i) ordered on the value of t_i, in each pair, such that the t_i's are in ascending order. That is, the first member in the set is scheduled to occur at the smallest time t_i, of all the t_i's in the set of pairs. Let X be the current event being simulated. Let T be the current time in the simulated system. Then the next event algorithm is as follows:

1. If the set E is empty, terminate the algorithm.
2. T = f(E) where f is a function that yields the value of the t_i in the first pair in the set E. (Note that this may advance the simulation time T.)
3. X = g(E) where g is a function that yields the identification of the event e_i in the first pair of the set E.
4. Delete the first pair in set E and deallocate storage used for that pair.
5. Simulate the event identified by X.
6. Go to step 1.

Figure 1 illustrates how an implementation of the next event algorithm might appear. In many implementations, the set E of events is maintained in some form of a linked list to allow for easy manipulation of the members. The next event algorithm possesses an attribute conducive to an accurate simulation of a real system. No restriction is placed on the time when an event may occur in the system. The simulation time T is, in effect, driven by the occurrence of events. In step 2 of the algorithm we see that the simulation time is updated from the time of occurrence of the event. Hence, an event can occur at any arbitrary time. Of great implementational importance is that the storage requirement for the algorithm is proportional only to the number of members in the set E. The major drawback to the next event algorithm involves the insertion of new pairs into the set E.

Insertion of a new pair into the set E is an action known as scheduling. Scheduling is used to cause the simulation of a given event at a given time. In the next event algorithm it is necessary that the event-time pair (e_i, t_i) be inserted in set E such that the ordering of set E on the t_i's be preserved. If set E is maintained as a linked list structure as in Figure 1, insertion of the pair involves searching the set E in a linear manner to determine where to insert the event-time pair in the set. For example, in Figure 1, we would search the set E starting from the list head. If we desired to insert the event-time pair (e_k, t_k) and set E contained N event-time pairs such that in each of the N pairs the relation

$t_i < t_k$ existed, then, we would be forced to search through N pairs before determining the point in set E to insert the pair (e_k, t_k). In the case of digital logic net simulation for the next event algorithm it can be shown that the probability of having a search length greater than one to determine where to insert an event-time pair in set E is nonzero except for trivial cases. The proof of this is found in Appendix A. If this probability were zero, then scheduling in the next event algorithm would always allow placing the event-time pair in the same place in set E for every event scheduled. Hence, no searching would be involved in scheduling an event. Since the probability is nonzero, searching is required in scheduling an event. If many events are scheduled to occur during the simulation, intuitively we can expect to search through a number of events to insert a new event. In digital logic net simulation, this searching overhead is critical due to the large number of events that may occur in the net.

The second time flow mechanism to be considered is the fixed time increment mechanism. Using the notation introduced above, the representation of the algorithm is as follows:

Let Δt be a fixed increment. (The size of Δt is designated by the designer of the simulation).

1. If the set E is empty, the algorithm terminates.
2. $T = T + \Delta t$.
3. If T is not equal to f(E) go to step 1.
4. $X = g(E)$.
5. Simulate the event identified by X.
6. Delete the first pair in set E and deallocate storage used for it.
7. Go to step 3.

Figure 2 illustrates a common manner of implementing the fixed time increment algorithm. Since the fixed time increment algorithm only has knowledge of points in time that are integral multiples of Δt, the structure reflects this. A single dimensioned array TQ of pointers is maintained. The pointers point to lists of event identification information. Each location in array TQ represents a distinct point in time. For example, the location indexed by I in TQ represents a simulation time equal to $I*\Delta t$. I is an integer by definition of the algorithm. In this manner, set E can easily be maintained for the fixed time increment algorithm. The time flow algorithm and the scheduling mechanism can then make use of set E.

The fixed time increment mechanism possesses a property that can seriously degrade the validity of the simulation. Since events can occur only at points in time that are integral multiples of Δt, if the model requires a relative continuum of time in order to present an accurate real world view then Δt must be very small for simulation accuracy. A very important implementational consideration (factor) is that the storage requirements are proportional to the sum of two quantities. The first quantity is the number of members in set E. The second quantity is the total time interval being simulated divided by Δt. This is due to the fact that the length of array TQ is proportional not only to the total time length being simulated but also to the inverse of Δt. However, the fixed time increment algorithm possesses an important advantage in the area of scheduling events. To insert an event-time pair (e_i, t_i) into set E, one need only add the event identification information for e_i to the list pointed to by the array TQ location indexed by t_i. Hence, scheduling events requires a fixed overhead in time and involves no searching, unlike the next event algorithm. In a simulation with a high number of events occurring with respect to the passage of simulated time, the fixed overhead involved with scheduling in the fixed time increment algorithm is conducive to a rapid simulation.

The integral approximation to a time continuum imposed by the fixed time increment algorithm may not be as serious as it seems, for information concerning the time it takes a given real logic element to evaluate its output from its input--the propagation delay--is not usually provided as an exact figure. Delays through actual logic elements are usually specified to fall within some minimum and maximum values by the manufacturer. Hence, careful selection of Δt for the fixed time increment simulation can lead to accurate results without the scheduling overhead that can be incurred in the next event algorithm.

III. Event Occurrence in Digital Logic Nets

Before a meaningful simulation can be performed, one must consider the real-world item that is being modeled in the simulation. In this case, our interest is centered on digital logic nets. Let us first consider the characteristics of individual types of logic nets.

A combinational logic net is one in which the outputs are determined solely by the present value of the inputs.[2] Combinational nets tend to settle to a final state within a finite time after excitation of the inputs to the net. Since all of the events in the net may occur in a relatively short period of time, it is possible to find a rather high density of events occurring in a logic net, in relation to time.

A sequential logic net is one whose outputs are dependent on the past history of the net as well as the current inputs to the net. A synchronous sequential system employs clocks to synchronize the time at which the system transitions from one stable state to another. All internal combinational values are assumed to be stable at each clock transition providing synchronization. An asynchronous sequential net has no synchronization and thus "free runs."[2]

Since in a synchronous sequential logic net all changes occur with the "clock" all events occur within some finite time after the "clock" occurs. Hence, in a synchronous sequential net, event occurrence tends to cluster around occurrence of the "clock."

In an asynchronous sequential logic net, event occurrence is determined solely from the structure of the logic net. Thus very little can be said in general about event occurrence in such nets.

IV. The Design Goal of a General Purpose Logic Net Simulator

There are several important goals to be achieved in a general purpose logic net simulator. First, the simulator should place no restrictions on the length of the time interval being simulated. Second, it should employ a time flow mechanism strategy that will allow for efficient scheduling of a high density of events with respect to time. The simulator should make efficient use of storage and should have small overhead in the setup time for execution of each event.

V. TEGAS2--A General Purpose Logic Net Simulator

The time flow algorithm of TEGAS2 represents an extension of the one used in TEGAS.[6,7,8] The basic simulator driving time flow mechanism is the fixed time increment mechanism. Immediately, the observation is made that storage limitations severely restrict the time interval over which a simulation can be performed using the pure fixed time increment strategy. TEGAS2 employs a combination of fixed time increment and next event strategies to circumvent the time interval problem encountered in the pure fixed time increment mechanism. The fixed time increment algorithm implemented in TEGAS2, used to drive the execution of events, utilizes a single dimension array Z that is N locations long. Figure 3 details much of this description. In the current version N is equal to 100. Array Z contains pointers to the lists of events scheduled to occur. Array Z is indexed from zero using the simulation time T modulo N. Another variable is used to represent the current time cycle contained in the array. This variable will be called S. Hence the time interval being simulated is from A to B where A = S*N and B = (S+1)*N-1. So that an event found by the pointer X into the array Z is occurring at simulation time T = (S*N)+X. Any event scheduled for time less than or equal to B is scheduled in array Z in the standard fixed time increment scheduling method. All events scheduled to occur after time B are stored in a linked list ordered on time. This list structure constitutes the next event algorithm portion of TEGAS2. Always at time B an event is scheduled which in effect advances the simulated time interval held in array Z. This event is known as the event list update. This event increments S by one and removes all events from the ordered linked list that will occur within the new time interval from A to B. These events are linked to array Z so they may be executed by the fixed time increment portion of TEGAS2.

This mixed set of time flow mechanisms provides an algorithm for a simulation time interval of 34359738637 units with storage used proportional only to the number of events scheduled. TEGAS2 has been used to simulate logic nets of size greater than five hundred gates with very reasonable execution times. Experience with TEGAS2 indicates that this approach to a time flow mechanism for a general purpose logic net simulator is valid. The original TEGAS[6,7,8] had a simulation time interval limitation of 10000 and storage requirements were proportional to both the number of events scheduled and the maximum time interval that could be simulated. The strategy used in TEGAS2 has overcome these problems. Appendix B details typical simulator execution times.

APPENDIX A

Theorems Concerning Proofs of Scheduling in the Next Event Algorithm

Let $U = [u_1, u_2, \ldots u_n]$ be the set of all types of events that can occur in the logic net being simulated.

Condition A.1--Given the set E from the Next Event Algorithm.

Condition A.2--Let all events in the deterministic logic net simulation be scheduled in the following deterministic manner:

Let d_i be a quantity > 0 associated with each u_i.

When an event of type u_i is scheduled, it may only be scheduled to occur at time X where X is defined as follows:

$X = T + d_i$ where T is the current time of simulation.

Condition A.3--Let $P(u_i)$ be the probability that an event of the type u_i will be scheduled during the simulation.

Then $P(u_i) > 0 \; \forall \; u_i \in U$

Theorem 1--Given conditions A.1, A.2, and A.3. If $d_i = d_j$, $i \neq j \; \forall \; u_i, u_j \in U$ then for the next event algorithm to insert an event-time pair (e_i, t_i) into set E and to retain the ordering of set E one need only insert (e_i, t_i) after the current last member in set E.

Proof--Set E is ordered in ascending order on the t_i in each pair. In order to insert the pair (e_i, t_i) at the end of set E and to retain the ordering of set E one must insure that t_i is

always greater than or equal to t_k where t_k is the time of the last event-time pair in set E. To prove this theorem we need only prove that t_k is never greater than t_i. We can prove this as follows:

By condition A.2 all events scheduled are scheduled in the same manner. Also, all d_i's are greater than zero so the time in the simulator always advances. As a consequence, the largest value t_k could have is equal to $T + d_k$ where T is the current simulation time. The value t_i is equal to $T + d_i$. Since $d_i = d_k$ then $t_i = t_k$. This value of t_k can only occur if the event e_k was scheduled at the same simulation time as event e_i. If the event e_k was scheduled at simulation time X less than the current simulation time T then the relation $t_k = X + d_k < T + d_i = t_i$ would hold. This relation is also allowable under our theorem. The event e_k cannot have been scheduled at a simulation time X greater than current simulation time T since the simulation time always advances. Hence, the event-time pair (e_i, t_i) would have been placed in the set E before the event-time pair (e_k, t_k). Q.E.D.

Condition A.4--The probability is nonzero that the set E will contain at some point in the simulation an event-time pair of type u_i and an event-time pair of type $u_j \ni d_i \neq d_j \Rightarrow$ Probability $(d_i < d_j \vee d_j < d_i) > 0$.

Theorem 2--Given conditions A.1,A.2,A.3 and A.4. In the next event algorithm there exists a nonzero probability that a search of length greater than one (that is, the first place in set E inspected is not the correct place of insertion for the event-time pair) will be needed to locate the proper place to insert an event-time pair being scheduled. This assumes that the initial point of inspection and search is initiated from the last member of the set E.

Proof--For the search length to be greater than one, the t value of the event-time pair (e_i, t_i) being inserted into set E should be such that making it the last member of set E will destroy the ordering of set E. This will be true if the t value of the current last event-time pair (e_k, t_k) in set E is such that $t_k > t_i$ is true. To state that this occurs with a nonzero probability, we only need show how it can occur and then note that condition A.3 and condition A.4 assign nonzero probabilities to the contributing factors.

From proof and terminology of Theorem 1 we note that $t_k = X + d_k$ and $t_i = T + d_i$, where X is the simulation time at which event e_k was scheduled. We only need know that $t_k = X + d_k > T + d_i = t_i$ may occur. We note that if $X \leq T$ then d_k must be greater than d_i for this relation to occur. Since the probability that d_i is not equal to d_k is nonzero and the probability that two events e_i and e_k having $d_i \neq d_k$ will be in set E simultaneously is nonzero thus the probability that $t_k > t_i$ is nonzero. Q.E.D.

NOTE: For a search from the first member of set E, the same theorem can be stated and proved in a similar manner. It may be possible that for certain next event simulations given specific information about event occurrence and scheduling one might discover a means to predict search length and thus determine from which direction to search the set E. This might be a good problem to attack when the d_i's are known psuedo-random functions.

In the case of a logic net simulation, it can be seen intuitively that the search in set E will often be greater than one in length. This follows from the large number of different d_i's and events being scheduled as the logic net operates. Hence, the distribution of the type of events in the set E will be varied and accordingly due to the different d_i's the t_i of each event-time pair (e_i, t_i) will vary over a sufficient range such that a search of greater than length one will often be necessary.

APPENDIX B

TEGAS2 Execution Times
Synchronous Sequential Networks

	33 GATES	85 GATES	497 GATES
Time Units Simulated	2000	2000	620
Elapsed Time-Seconds	7	5	7
Seconds/Step	.0035	.0025	.011
Max Events At One Step	2	30	197
Total Events	229	1459	6595
Average Events/Step	.1144	.7291	10.619

References

1. Pritsker, A. A. B. and Kiviat, P. J., Simulation with GASP II, Prentice-Hall, Englewood Cliffs, New Jersey, 1969.

2. Kohavi, Zvi, Switching and Finite Automata Theory, McGraw-Hill, New York, 1970.

3. Kiviat, P. J., Villanueva, R., and Markowitz, H. M., The SIMSCRIPT II Programming Language, RAND Report R-460-PR, October, 1968.

4. "General Purpose Simulation System/360--Introductory User's Manual," IBM, H20-0304-1.

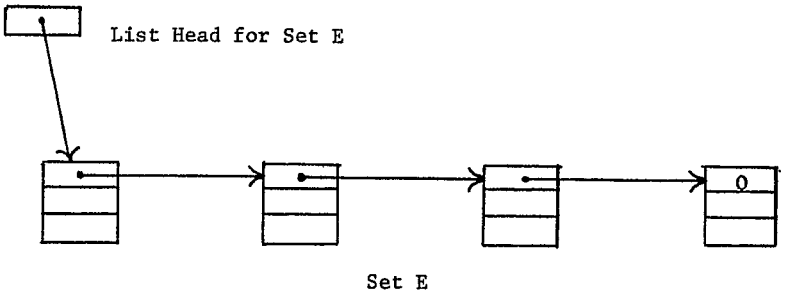
5. Naylor, T. H., Balintfy, J. L., Burdick, D. S., and Chu, K., Computer Simulation Techniques, Prentice-Hall, Englewood Cliffs, New Jersey, 1969.

6. Szygenda, S. A., "A Software Diagnostic System for Test Generation and Simulation of Large Digital Systems," Proceedings of the National Electronics Annual Conference, December, 1969.

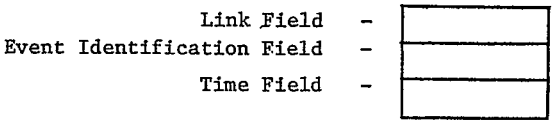
7. Szygenda, S. A., "TEGAS--A Diagnostic Test Generation and Simulation System for Digital Computers," Proceedings of the Third Hawaii International Conference on System Sciences, January, 1970

8. Szygenda, S. A., Rouse, D. M., and Thompson, E. W., "A Model and Implementation of a Universal Time Delay Simulator for Large Digital Nets," AFIPS Proceedings of the Spring Joint Computer Conference, May, 1970.

9. Nance, R. E., "On Time Flow Mechanisms for Discrete System Simulation," Technical Report CP 69006, Computer Science/Operations Research Center, Southern Methodist University, July, 1969.



FORMAT OF ENTRIES
IN SET E



Link Field - Points to Next Entry in Link List Field and is zero in last entry

Event Identification Field - Contains Information Describing the event to occur

Time Field - The time at which the Event is to occur

FIGURE 1 - THE NEXT EVENT ALGORITHM-A SIMPLE IMPLEMENTATION

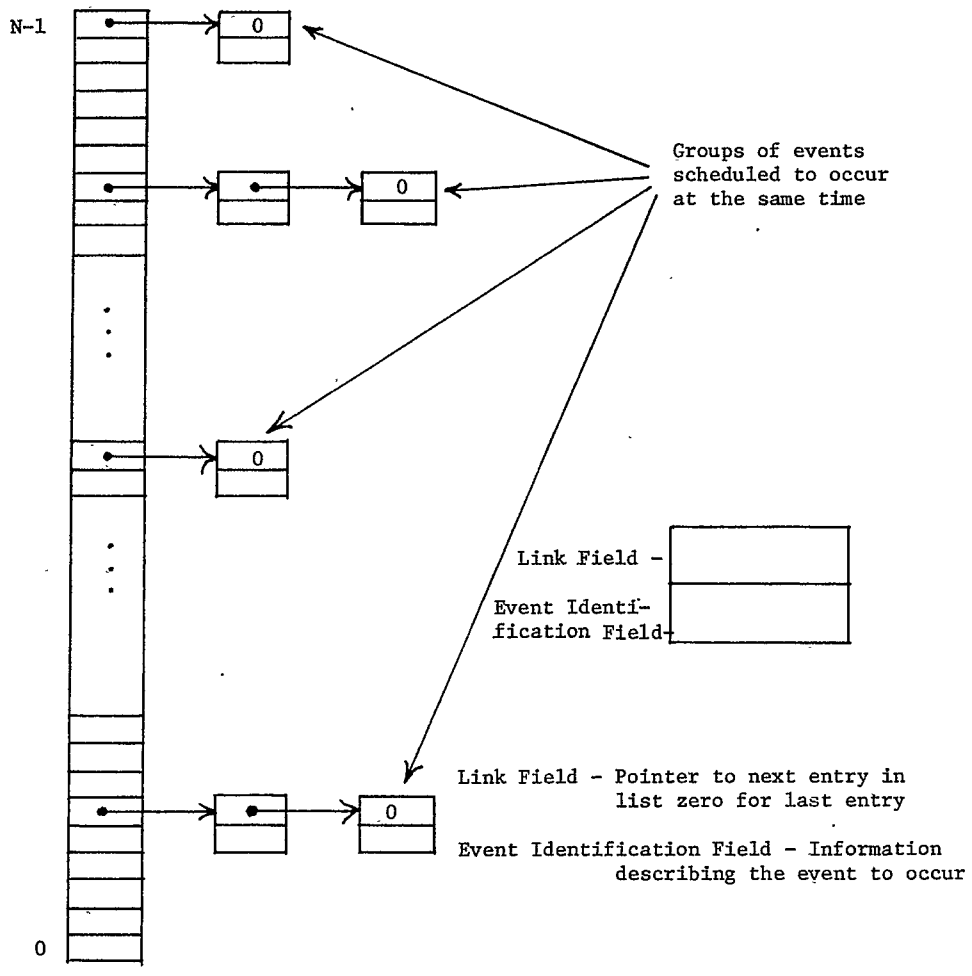
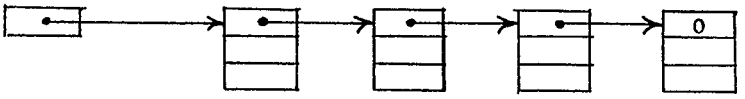
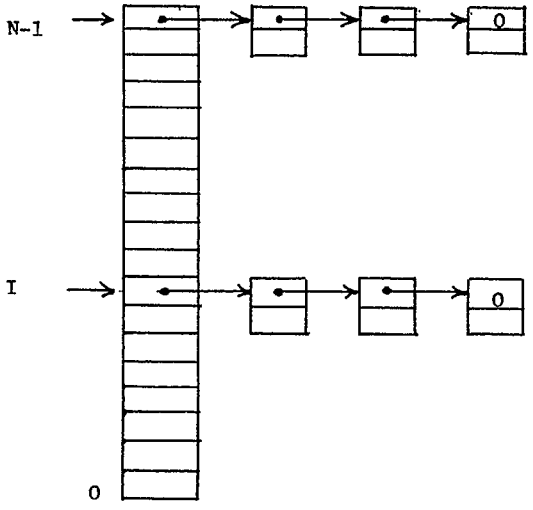


FIGURE 2 - THE FIXED TIME INCREMENT ALGORITHM - A SIMPLE IMPLEMENTATION



Event List

Figures 1 and 2 contain Descriptions of these structures.



ARRAY Z

THE EVENT LIST UPDATE EVENT is always found in the group of events pointed to by the pointer at Z(N-1). This event increments S by one and moves any events from the event list that are to occur in the next N-1 time intervals and places them in event groups.

If I is the Index to a Group of Events, the events are to occur at simulation time equal to (S*N)+I where S is the number of cycles that have been made through array Z and N is the length of array Z.

FIGURE 3 - TEGAS2 IMPLEMENTATION