Described is an experimental system for verifying logic designs in the development of a computer before a commitment to produce the computer is made.

The system simulates logic activity with both known (0, 1) and unknown (X) values. The use of the third value facilitates the generation of tests and the detection of circuit hazards.

A three-value computer design verification system

by J. S. Jephson, R. P. McQuarrie, and R. E. Vogelsberg

The development of new technologies for computer construction has led to new ways of circuit design. These computer technologies —methods for the control of the electronic signal flow using devices such as transistors and integrated circuits—have accentuated the usefulness of logic simulators during a machine development cycle. The long procurement cycles and hardware invariance associated with these technologies make design errors much more costly than in past technologies. Moreover, increasing circuit densities and speeds make hardware troubleshooting more difficult, and variations in circuit parameters between machines make hardware debugging[1] accurate only for the particular machine studied. The use of a simulator alleviates many of these problems. The simulator machine description is easily modified; thus a simulator is not affected by hardware procurement cycles and invariance. Moreover, the simulator operates independently of circuit variations and can perform worst-case analysis of timing hazards. Since the techniques used for troubleshooting an actual machine are similar, if not identical, to troubleshooting a simulated machine, simulation provides an opportunity to develop these techniques and to learn the computer system prior to the availability of hardware. This ability improves the efficiency of diagnosing hardware problems later in the development cycle.

In this paper, an experimental three-value simulation system is described. It is designed for use as an engineering tool to verify large (5,000 block) logic designs prior to the actual construction of hardware. The use of this logic simulator also ensures microprogram and logic compatibility in a microprogram-controlled machine. In addition to the advantages of other logic simulators, the three-value simulator possesses features that further aid the design verification task. The main difference in the simulator under discussion is that it operates with three values instead of the usual two. Besides the binary 1 and 0 values, the simulator uses an X value. This value represents an unknown condition and is considered to occur during any transition from one binary value to another. It acts as a transitory state in going between 0 and 1. The X value may also be used to represent an initially unspecified condition, an unpredictable oscillation, or a "don't-care" condition.

the third value

The use of this third value to represent the transition allows the simulator to detect combinational hazards, critical races, and feedback oscillations in the machine design.[2] Such hazards may occur as a result of single or multiple input changes to a logic block and indicate that either a timing or design error may exist in the simulated logic. In addition to detecting such errors, the simulator uses the X value to propagate the effects of the error through the circuitry. In most synchronous sequential machines, there are a number of potential circuit hazards that do not affect the actual operation of the machine. The three-value simulator's ability to propagate an unknown condition allows the designer to concentrate on only those hazards that may affect the machine operation.

In order to test a piece of hardware, the test procedure (stimuli) to be applied to that hardware must be defined. The problem of developing a complete specification test for a general-purpose computer has not been solved. The problem is essentially one of selecting a set of sequences of input patterns that will completely test the function of the computer from an extremely large number of possible combinations (a small processor recently simulated had a possibility of $10^{10^{14.3}}$ input tests).

The three-value simulator system addresses the problem of test specification at several levels. A functional microprogram simulator is used to prepare most of the simulator input patterns. This limits the input sequences to the set of input operations for which the hardware action has been designed and for which expected responses exist.

The use of the X value as an unknown input condition can further reduce the amount of test data that must be specified and can greatly increase the effectiveness of a given test. For example, suppose the user wishes to simulate the reset of a machine. The machine should reset regardless of the contents of the control register, the status of the console switches, or the states of the memory elements internal to the machine. In the three-value

simulation, this test can be made by setting the control register, all console switches except reset, and all memory elements to X, then simulating the reset. Any networks remaining at X represent networks that have not been reset to a known value. Without the X value, the reset would have to be tested for every possible combination of switch, control register, and memory element bits. For example, in a machine with 20 console switches, a 60-bit control register, and 500 other memory elements, this would mean repeating the test routine 2^{580} times (about 10^{175} separate resets) to gain the same amount of information.

timing problems

Since the three-value simulator is one of zero-delay, it does not take into account the signal propagation delays that are a function of the circuit technology, loading, and packaging. In the actual hardware, timing problems manifest themselves in two ways:

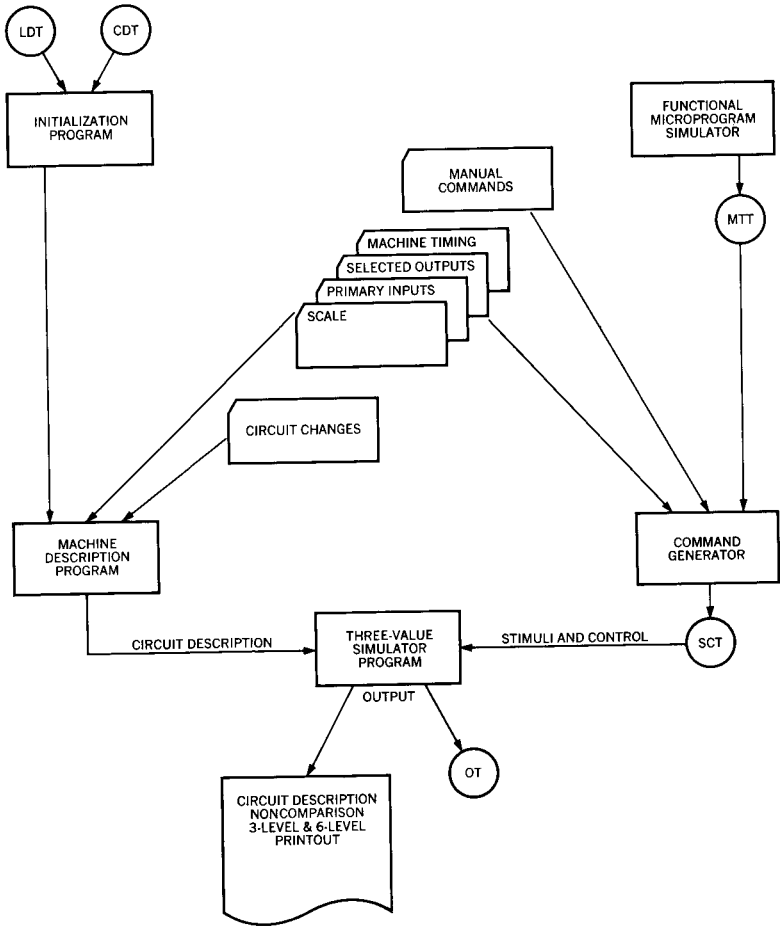
- The circuit behavior may be dependent on the relative delay through two paths in the logic circuitry.
- The cumulative delay of several layers of logic circuitry may cause it to respond too slowly to meet the specified machine cycle.

Due to running-time considerations, the simulator addresses only the first of these timing problems. Because the simulator is a zero-delay simulator, it is extremely pessimistic with respect to relative delay. All logic paths are assumed to have the same (zero) delay and thus a worst-case analysis of relative delay timing problems is performed. In situations where it is determined that a detected hazard does not exist in the actual hardware, the simulator operation may be modified by the definition of appropriate delay blocks.

The simulator is specifically a significant event simulator. When logic designs containing several thousand blocks are simulated, the running time of the simulation can become a problem. This problem results from the slow speed of simulation as compared with the actual hardware speed. The simulator must execute a serial sequence of program instructions at microsecond speeds to simulate a parallel set of hardware operations that occur in nanoseconds. The simulator addresses this problem by simulating only significant events. The simulation model of a logic design consists of a number of interrelated table structures that contain pointers representing the circuit interconnections. For any given input change to the logic, only a small number of blocks will change state (i.e., there will be a limited number of significant events). The simulator pointer structure allows the effects of an input change to be traced through only those blocks that change value and thus limits the amount of computation required during a simulation to a minimum.

In summary, we can say that the system described here detects logic design errors and timing hazards in large logic designs using a reduced set of test patterns and a minimal amount of computer time.

Figure 1 The three-value simulator system



Simulator system

The simulator system is composed of a series of computer programs that together provide a sophisticated analytic aid for logic design. The simulator can be used in development work as soon as the logic of the proposed machine is described on tape (here called the logic description tape, LDT) and the circuitry of the machine is described on tape (the circuit description tape, CDT). Either the entire machine logic or selected sections of the logic can be simulated.

preparation stages

A simulation run can be made only after the logic to be simulated, the input stimuli to be applied to that logic, and the output information to be produced by the simulator are defined. The preparation of a run for simulation must be done in stages with the designers providing the control data and instructions for each stage of the operation. Figure 1 illustrates the major stages of the simulation setup and the general simulator data flow. The three major stages are: (1) initialization, (2) machine description, and (3) simulation control.

During the initialization stage, the logic to be simulated must be selected and an initialization program must be run to prepare a rough circuit description for the simulator. Upon completion of this stage, the machine description stage is initiated. During machine description, the circuit description is updated to correct any errors detected during the initialization, and primary inputs, outputs, and machine timings are defined.

Finally, in the simulation control stage, the circuit input stimuli and the simulator controls are defined. In general, a functional simulator is utilized to provide a microprogram trace tape (MTT) containing input and output information, and the simulator controls are manually specified. At the completion of this stage, the simulator is run and the desired output produced.

The three-value simulator

The simulator program carries out the simulation of the circuit image that has been defined during the machine description stage. The simulation control tape (SCT), which is prepared during the simulation control stage by a program called the command generator, controls the simulator program action.

When the command generator has completed the preparation of the SCT, the simulator program is called. The simulator reads the circuit data into storage, and an initialize routine is then executed to establish various tables and initial controls. Initially all block values are set to X (the unknown condition). The simulator then reads the first information on the SCT and begins the simulation.

major commands

The information on the SCT consists of a string of commands to be executed by the simulator. There are three major commands used to control the results of the simulation and a number of commands used to control the output of the simulation. The three major simulation control commands are:

- Clock update
- Apply stimulus
- Compare response

The *clock update* command causes the simulator to set an internal clock to the indicated time. This time is used by the simulator to maintain correct machine timing relationships and to identify simulation output.

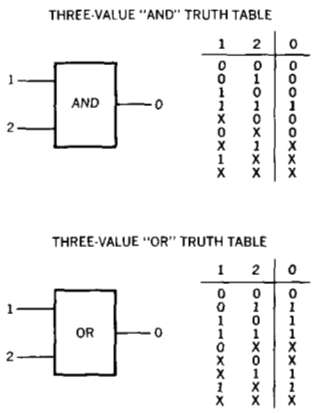
The *apply stimulus* command causes the simulator to change the value on an indicated primary input to a new value. As mentioned previously, the simulation is of "significant events" only. Part of the task of the command generator is to identify those primary inputs that have changed value during a given clock time and to pass only those primary network input numbers and new values to the simulator. This implies that after the first few sets of inputs have been applied, only a small portion of the machine will have to be simulated for any given clock time. A significant amount of simulation running time is thus saved.

The *compare response* command causes the simulator to compare the value calculated for an indicated primary output with the value provided. If a value is found that does not compare, the simulator initiates a printout of the machine status at the time the comparison is made. This type of output is usually the prime source of debugging information obtained during a machine simulation.

The remaining simulator commands correspond to the manual commands and control the types of simulator output and the times during which this output should be produced. There is also a stop simulation command that causes the simulation to terminate at an indicated time (or after a specified number of values are found that do not compare).

During the simulation, all logic functions have a three-value truth table. These tables for an AND block or an OR block are shown in Figure 2. The X or unknown value appears at the output of a block if the two-value truth table of the block indicates that the changing of the value on the input may cause the block output to change (i.e., the input change may propagate to the output). This use of the third value allows circuit hazards that would go undetected in a two-value simulation to be detected. For example, changing the inputs of an AND or an OR gate from 0 to 1 and from 1 to 0 in the same cycle may cause a temporarily erroneous output. This condition is detected and propagated in the three-value simulation but not in a two-value simulation as shown in Figure 3.

Figure 2 Truth tables



During a simulation, the simulator reads a simulation control tape record and then executes the commands in that record. The usual ordering of commands for one machine time is:

command ordering

1. Clock update
2. Control commands
3. List of apply stimulus commands
4. List of compare response commands

For such a sequence, the simulator first updates the internal clock, then sets a series of controls based upon the control commands, and finally processes the list of apply stimulus commands. The processing of this list is the actual logic simulation stage of the operation.

For each primary input network specified in an apply stimulus command, the simulator applies an X to the network and propagates that value through the logic until no further block output changes can occur. For example, if an X being propagated on a network reaches an AND block that has some other input set at 0, the propagation of the X value is terminated for that block. When no further block changes can occur for the given primary input, the next primary input specified in an apply stimulus command is set to X, and the operation is repeated. This sequence continues until all the primary inputs mentioned in the list have been set to X. The propagation of X values on these networks

Figure 3 Combinational hazard detection

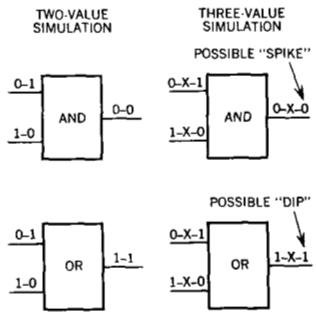
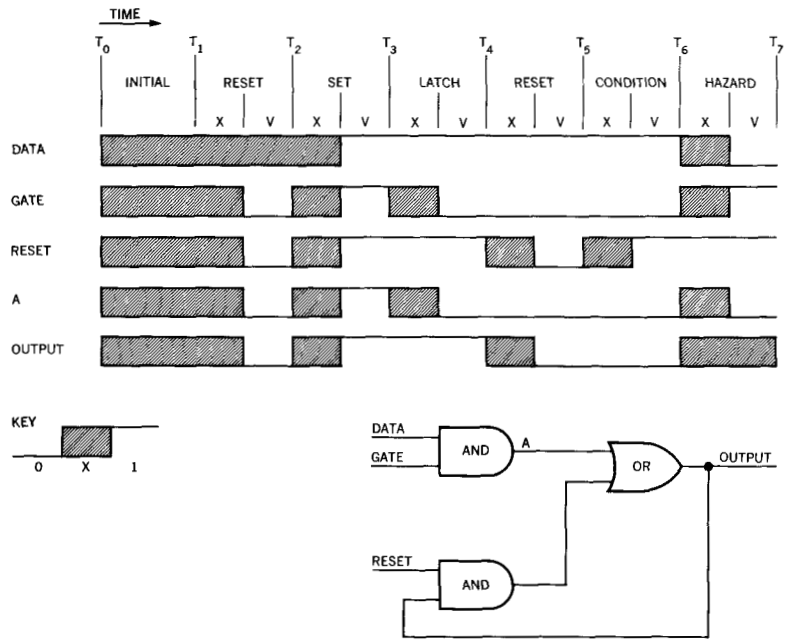


Figure 4 Latch with hazard



is the first of two passes that the simulator must execute in changing a set of input values and is called an X-PASS. During the second pass, the values given for the primary inputs in the apply stimulus commands are applied to the primary inputs and propagated one at a time in a similar manner to that used in the X-PASS. This pass is called the VALUE-PASS and (for circuits containing no unit delays) completes the simulator action for the set of stimuli provided.

propagation example

The simulator then processes the list of compare response commands and initiates output operations if necessary. For example, consider the latch circuit of Figure 4. At T_0 all the input networks are at X; the OUTPUT and the line labeled A will also be at X. If the latch is reset as shown at T_1 by setting the GATE and RESET input lines to zero, the OUTPUT and A will eventually go to zero. The simulator might propagate this change as follows:

1. Apply X on the GATE line. Note that during an X-PASS a block output can change only to an X from a value. Since the block value is already X, no further propagation is necessary.
2. Apply X's to the RESET line with similar results. This completes the X-PASS.
3. Apply a zero to the GATE line. Note that during a VALUE-PASS a block output can change only to a value from an X. Because the AND block's present value is X, it can change. Thus

calculate the value of the AND block's new output. The value is zero, which is different from X, so propagate this value to the OR block. The OR block's previous value is X, so it can change. Thus calculate the value of the OR block. The new OR block value is X (because the lower input is still at X). Because this is the same as the previous output, stop propagation.

4. Apply a zero to the RESET line. This change causes the lower AND to go to zero followed by the OR block going to zero; then propagation will stop at the lower AND block's input (because the block is now set to a value).

The remaining circuit changes ranging from T_2 to T_5 can be verified in a similar manner.

hazard indication

Note that the circuit shown contains a hazard. At the time labeled T_6, changing the levels on both the DATA and GATE lines causes a "spike" to be produced. The simulator indicates this condition by allowing the latch to be set to X (where it will stay until another set or reset is applied).

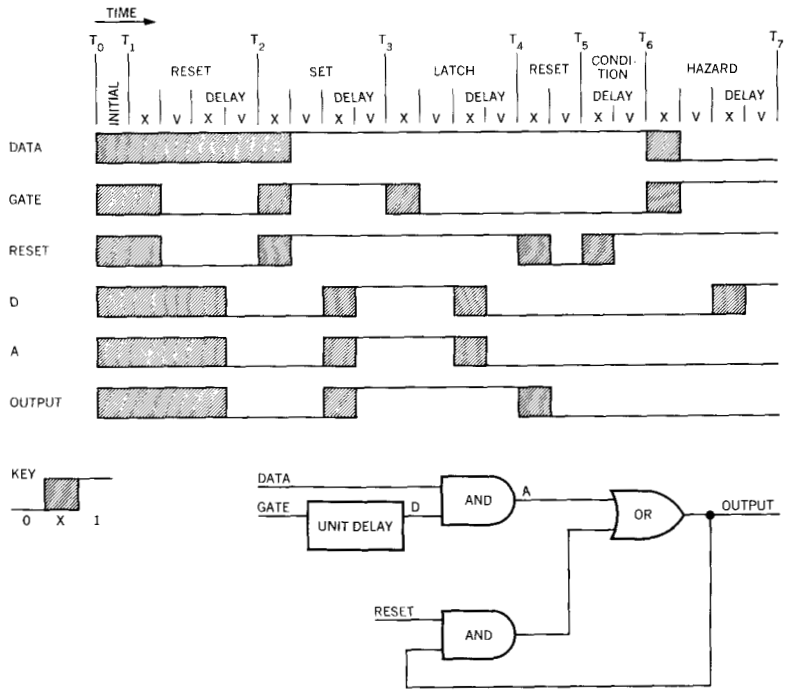
The hazard thus indicated may or may not exist in the actual hardware. The duration of the spike may not be sufficiently long to cause the actual latch to be set, or there may be timing constraints due to a delay in the logic paths that generate the DATA and GATE lines that ensure that no spike will be produced. When a circuit depends on such logic delay to operate correctly, a unit delay block must be inserted in the model of the circuit to correct the operation of the simulator.

delay block

When the delay block is encountered during the propagation of a signal, the propagation is halted at the delay block, and an entry is made in the delay table. The propagation of the next signal change is then initiated. This operation is continued for both the X-PASS and the VALUE-PASS. Thus, starting with an X-PASS, the X's on networks that are changing are propagated as far as possible (i.e., until a block that doesn't change state or a delay block is reached). Then a VALUE-PASS is made during which the values are propagated as far as possible (in the same way as for an X-PASS). When these two passes are complete, the simulator has entries in the delay table corresponding to the X's and the values that have propagated to delay blocks. The X's are then placed on the outputs of the delay blocks and propagated as far as possible in another X-PASS. When this X-PASS is completed, the values in the delay table are placed on the outputs of the delay blocks and propagated in another VALUE-PASS. These are called the DELAYED-X-PASS and DELAYED-VALUE-PASS, respectively.

If a delay block is encountered during a DELAYED-X-PASS and DELAYED-VALUE-PASS, the above process is repeated, and a third set of X and VALUE passes is established. Thus three delays in series cause three DELAYED-X passes and three DELAYED-VALUE passes to be taken.

Figure 5 Latch without hazard



Returning to the latch of the example, consider the circuit as shown in Figure 5. A unit delay has been added to the circuit in the GATE line to correct conditions resulting from the previously detected hazard. The simulator operation shown then results. Notice the sequence of events at time T_6. The change in the DATA line is propagated immediately as before; however, the change in the GATE line is delayed until the second set of X and VALUE passes by the unit delay. Because these transitions have been separated in time as seen by the simulator, the spike is no longer produced, and the new circuit contains no timing hazard.

The addition of the delay block has doubled the number of passes that the simulator must make to simulate this circuit. The addition of a delay to a larger circuit will not necessarily double the number of simulator passes, but it will increase the number. Care should be exercised in the placement of unit delays to avoid excessively increasing the length of the simulator running time.

It is also possible to construct a circuit, using logic blocks and unit delays, that oscillates indefinitely. To guard against such a situation, the simulator, after a clock update, counts the number of times it passes through a delay level (a delay level is defined as a DELAYED-X and a DELAYED-VALUE pass). If this count exceeds a specified number, the simulation is terminated, and appropriate output is produced.

printouts

Four types of printouts are available to aid in analyzing the logic that has been simulated:

- Printout when values do not compare
- Three-level printout
- Six-level printout
- Circuit description printout

The printout for values that do not compare is initiated by the simulator whenever a difference between an expected result (as defined on the MTT) and the result determined by the simulator occurs. This printout lists the values of the response facilities, the values of the stimulus facilities, and the block values of the remaining networks in the logic at the time that the values are found not to compare. Knowing the machine time, the operation to be performed, and the status of all the inputs to the circuit, the user can look at the outputs and determine whether or not the functions were performed correctly. To isolate the error, the block values can be analyzed to determine the status of every network in the machine. With these values, he can trace the circuit back from the failing output to the point where the trouble can be found.

The three-level and six-level printouts are selected by the user when he desires detailed knowledge of the sequence of the changes in all network values during some phase of the machine operation. These printouts list all the network values of the simulated machine for the times specified. If the three-level printout is selected, the possible network values are 1, 0, and X. The six-level printout is a modification of the three-level printout and is designed to simplify troubleshooting. For this output, the characters $+$, $-$, and $=$ are used to indicate block values of 1, 0, and X that have not changed since the last output. The values 1, 0, and X are used only for those blocks that have been active since the last output.

The circuit description listing provides detailed block data as it is defined in the simulator. This data can be used to check the circuit in the simulator against the machine logic.

During normal operations, the printout showing information about values that do not compare provides the major debugging aid. However, the three-level and six-level printouts allow the user to observe every machine operation and every block change in whatever detail is desired. Additional output can be prepared on tape (an output tape, or OT) in a format that may be processed by other programs to simplify the analytic procedure. Several such programs are presently being considered.

Summary comment

The three-value simulator has been used on both central processing units and input/output controllers with microprogram control and hardware control. Four major systems have been