The role of simulation in LSI design

by J. J. TEETS

IBM Systems Development Division
Endicott, New York

INTRODUCTION

Forecasts for Large Scale Integration (LSI) indicate that logic gate densities of 1000 gates per chip could possibly be obtained during the next decade. Supposedly, device and circuit innovations could be used to achieve even higher densities. There are, of course, a number of reasons why the industry is not yet manufacturing major LSI hardware on a large scale. One of the reasons is the lack of superior software support for computer-aided design, particularly simulation aids. For LSI, a major increase in logic design verification simulation will be required at the chip level to ensure correctness of design and to minimize the need for chip redesign.

The purpose of this paper is to review what has happened in the last five years, particularly in design verification, and to project what must be accomplished in the next five to ten years in the area of simulation in order to implement the promise of LSI.

REVIEW OF THE LAST FIVE YEARS

This section briefly describes high-level (architectural) and low-level (circuit component) simulators, and then closely examines recent logic level simulators. Logic simulation employs much more detail than high-level simulators and much less detail than circuit simulators. The purpose here is not to demonstrate that these simulators are inadequate for LSI technologies, but to show that each still has a place. The intent is to display a need for multi-mode simulation at the logic level in one user interface.

High-level and low-level simulation

Predictably, there has been and will be high-level simulation studies on various approaches to implementing LSI. Some of the major aspects and tradeoffs evaluated include: (1) replacing third generation systems with LSI, (2) enhancing architectural approaches with LSI scratch pads and main store, (3) microprogramming techniques (both read-only and writable control stores), and (4) considering hardware-software tradeoffs.

An example of an available capability for such high-level studies is the General Purpose Systems Simulator (GPSS). GPSS provides for modelling systems and can collect various statistical measurement quantities such as throughput, resource utilization, queue lengths, busy times, system bottlenecks, etc. The collection of such data is very valuable in evaluating high-level architectures, both hardware and software, in all phases of design.

On the other end of the simulation spectrum is the very detailed low-level (i.e., circuit analysis) simulator. This analysis is at the component junction/resistor level; here the differential equations are solved by numerical methods.

Design verification at the logic level

The requirements of pre-LSI technologies such as RTL, DTL, TTL and recently ECL have led to the development and enhancement of computer-aided design support for simulation, wiring, testing, and packaging of designs. The concept of a central design file, used by all these applications and massaged to satisfaction, is well-known. In this section, the simulation techniques used for technologies leading to LSI are reviewed in some detail to provide a basis for discussing later improvements and extensions required for today and tomorrow.

The techniques used by early logic design verification simulators were of relatively narrow purpose, i.e., they attacked a small set of design problems by using techniques specifically designed for these problems. There were basically three early types or modes of
logic simulation that became popular and were implemented (2VND, 2VUD, and 3VZD). All were similar (table driven, handling both sequential and combinational logic) but each was valuable to the designer in a different way.

1. 2-Valued, Nominal Delay Simulation (2VND)\textsuperscript{3,9}

This mode of simulation assumed that the state at any point in the logic at any given time was either “0” or “1”. The nominal delay was a gross approximation of the circuit delay (including loading and line delay where appropriate) in the form of a “rise” or “fall” delay, which corresponded to the amount of time taken for the circuit to respond to an input change and change the output state from 0 to 1, or 1 to 0, respectively (see Figure A). When primary inputs to the logic are stimulated, the simulator performs the Boolean function of the blocks driven by the primary inputs. The basic functions performed are shown in Figure D (ignore the “X” and “U” values). For each block whose value is to change, the proper delay is selected and the new block value is propagated after that delay. Propagation continues through the logic until a steady state is reached. When a block’s value does not change as a result of an input change, propagation does not occur. This is known as “significant event” simulation and is common to all three simulation techniques being discussed. This mode of simulation, although somewhat gross in timing analysis detail, was very useful in gaining some engineering confidence in the behavior of the early design. This mode was more economical than the “three-valued zero delay simulation (3VZD).”

One area where this mode fell short, even to the point of misleading the user, can be demonstrated by the following example: the 2VND technique (as used in early simulators) predicted that the response of a TTL inverter whose rise and fall delays were 15 nanoseconds (ns) to an input pulse of width 1 ns, is an output pulse 1 ns wide and 15 ns later than the input (Figure B). In fact, however, a TTL gate requires a minimum input width of about 12 ns to achieve any output response and would, therefore, completely ignore an input pulse of 1 ns or even

10 ns wide (Figure C). Moreover, the technique is not pessimistic as a simulator should be. It allows the user to think that he can count on a narrow pulse to get through a logic gate that is sure, in fact, to ignore it. The point here is that simulators should not tell lies. Pessimism may produce conclusions from simulators which may never exist in real hardware and hence cloud the issue; however, the designers must be alerted to potential problem areas, and then decide for themselves when a real problem exists.

2. 2-Valued, Unit Delay Simulation (2VUD)\textsuperscript{6}

This mode is very similar to the 2VND mode, except that the delay of each circuit is the same (one unit), independent of the type of circuit and the direction of change. The contention of this mode is that if one unit of delay is assigned to each logic block (with some exceptions which are assigned a delay of zero), the results of simulation will be sufficiently accurate to serve
a very useful purpose. Furthermore, if a convenient facility is available to add or reduce delays by units, the simulator will be able to closely represent the operation of the actual machine. The applicability of this approximation is dependent on the manner in which the machine was designed. Due to the variation of delays in logic circuits from manufacturing, the designer must allow sufficient tolerance so that all of the manufactured machines can operate identically to the prototype within certain limits of delay variations. Thus, one can consider the unit delay machine merely as one of those variations.

The most important advantage for this simplified timing simulation mode is its speed. It is faster than the other modes and does not require knowledge of nominal delays nor does it require storage for those delays in the simulator. It is subject to shortcomings similar to 2VND in that both modes fail to detect most hazards (subtle timing problems).

3. 3-Valued, Zero Delay Simulation (3VZD)

This mode, the most pessimistic of the three presented, was implemented primarily to detect hazards and races, as well as to investigate Boolean behavior. All possible timing problems of a design are identified to the designer. This was realized by the introduction of a third value, “X”. Any time an input (beginning of course with the primary inputs) changes state, it is forced to pass through X, signifying that the signal is “in transition” or “unknown”. When this technique is used, first every primary input that is changing state goes to X, the driven blocks are simulated, and propagation of all blocks to a steady state occurs. The rules for simulating X in a Boolean function are a subset of Figure D, ignoring the “U” for now. After a steady state is reached, the changing inputs go to their final values and again propagation takes place to a steady state (if possible). If a timing problem is a possibility, the value remains X on a block. This indicates a potential hazard involving that block. The possible steady state effects of propagating the hazard are also shown by X values. A facility exists to allow the designer to eliminate some of the simulator’s pessimism where races detected by the simulator are known to be noncritical. When used properly, this mode will allow the designer to be confident that all potential timing problems have been identified. However, eliminating pessimism in one part of a design often has unexpected effects on other parts.

Because of the two-pass operation described above, this mode of simulation is a little more expensive to use because of the additional computer running time; however, this must be evaluated as a tradeoff against extensive hazard detection capability not available with the other modes.

4. Extensions

Combinations and other variations of the previous three techniques have been used; one of the more useful perhaps was the introduction of a fourth value, “U”, which signifies “uninitialized”. When a U appears on the input of a block

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>X</td>
</tr>
<tr>
<td>U</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>U</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>U</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure D—Example of simulator rules
(for all modes, 4-valued)
being simulated, it is propagated according to the rules of the simulator (Figure D). At any given steady state condition, the appearance of U’s in the logic indicates the absence of initialization data, and therefore, identifies the importance of initial values. This can be very useful when evaluating system reset operations or the effect of incompletely specified input patterns. Two forms of the output from these simulators include: (1) timing charts, which show the reaction (in simulated time) of inputs and outputs of selected blocks, and (2) "snapshots", which display the value of all points in the model at selected times.

Other improvements include extension of 2VND and 2VUD to include both the X and U concepts. Thus, the nominal delay and unit delay modes can be made more useful in the area of hazard detection.

5. The Unfortunate User

Irrespective of the merits of these simulators individually, they had the collective disadvantage of not being mutually compatible. To simulate for more than one purpose (using more than one mode) required considerable, if not total duplication of effort. The problem was compounded because each program had a unique user interface. Thus, the user had to learn several incompatible systems before he could simulate at any desired point in his design cycle. The cost of this additional training and duplication of effort is no longer tolerable, particularly for LSI. It is now essential for different simulation efforts to have maximum commonality in all respects, particularly the user interface. This resulted in the requirement for multi-mode simulation and leads us into present simulator technology.

TODAY AND TOMORROW

Current simulation techniques in the design process

Whatever the size of a design project, there is a general design procedure to be followed. Several stages in that procedure involve simulation and these and other stages may be iterated to feed back information to improve the design. For purposes of discussion, these key stages in design are:

1. Establish system objectives and specifications.
2. Model the system and evaluate it by high-level simulation.
3. Partition the system into units (subsystems) and establish unit specifications.
4. Model the units and compare to specification by simulation.
5. Design units at the implementation level, modeling them for design verification simulation.
6. Compare models from different stages by simulation to ensure the detailed design meets unit and system specifications.
7. Build the system.
8. Check the system against simulation results.

At stages 2, 4, and 6, simulation is employed; however, it is obvious that the level of system being modeled is different at each stage. Moreover, stage 6 simulates several levels together. Hence, multi-level simulation is required for a fully integrated simulation capability.

Another major simulation problem that can and must be solved is that of getting very large models into relatively small computer storages. Previously, users had no option (other than not simulating) except to eliminate bits of their model until it was small enough to fit. This usually caused distress by ending up with functionally awkward pieces and unchecked interfaces between pieces. The solution for this is to partition a model, from the start, into a set of coherent regions (units), each having a well-defined interface with its neighbors (Figure E). These regions may then be simulated together as a system with the simulator automatically handling inter-region transactions by connecting together signals with identical names. An incomplete (or not yet designed) region can be simulated by using manually generated data or higher level models...
in place of missing parts which influence the regions modelled at the logic level.

This multi-region concept readily lends itself to techniques for solving the storage problem previously mentioned. One solution is to use a roll-in, roll-out technique to make storage available for active regions. Another solution is to compress the storage requirement (and execution time) of those regions that are being used only to generate interface activity for the benefit of simulating other regions in detail. This compression takes the form of higher level models for the other regions so that the functions can be reproduced at the interface. With this technique, a "multi-level simulation" capability is defined. Merged with a multi-mode capability for those regions modelled at the logic level, an overall capability for system simulation can be provided where the detailed simulation takes place in the area of user interest.

These concepts lead toward the idea of "top-down" design using simulation. In stage 4 of the design, high-level models for the different units may be used to compare specifications of the units against system level specifications by simulation. As stage 5's logic level model becomes available for a given unit, the simulation at stage 6 may take advantage of the higher level models of other units not yet designed in detail. The stimulus (i.e., the input patterns) for the detailed model can be provided by the high-level models surrounding it and can operate concurrently with the high-level models. The high-level model for the region being simulated in detail may be simulated at the same time, and the outputs of both can be fed into a comparator to ensure that the implementation design meets unit and system specifications. This use of the "high-level environment" provides the user a pseudo system simulation at all times, regardless of which unit is being investigated and designed in detail. It also eliminates much of the problem of providing adequate stimulus to the low-level design, which is otherwise a very tedious task.

Thus, the initial and unavoidable need to partition a model can be turned to an advantage. It leads to orderly design, region by region, with thorough checking and evaluation of each unit. In addition, very important advantages, particularly in the LSI design environments, can be addressed in the area of design verification; these include:

1. verify the correctness of design (including comparing high-level to low-level models and comparing different versions of a unit).  
2. detect hazards and races which would not be detected by building a hardware model.  
3. speed up the design process by using top-down design techniques (which also provide good communication between unit design groups).  
4. reduce the overall cost of the design process.

All of the above, particularly the cost factor, are imperative in the LSI design environment.

It has been stated that for LSI, a major increase in logic simulation at the chip level will be required to ensure correctness of design and to minimize the need for chip redesign. The engineering change problem is particularly perplexing and still lacks good proposals for its solution. If a chip change is required sufficiently early in an LSI machine project, redesign costs will be incurred although slippage may not occur. During machine test, however, where in the past the practice was to install an immediate fix if possible, slippage as well as redesign costs will accrue; a repeated string of such change cycles is obviously intolerable. Thus, LSI ideally requires error-free design which necessitates simulation aids and capability at a level never required or achieved in the past. Such ultimate goals as self-diagnosing and repairing machines in LSI cannot possibly be achieved without significant advances in design verification and fault simulation capability.

Thus, because of the cost factors and the related problems of system prototype "bring up" (engineering change activity), it appears that the LSI technology is insisting upon much more extensive simulation and paper design before manufacture, and at the same time, a reduction in the overall machine development cycle and cost.

Another salient point that must be made is that it is imperative for fault simulation to be integrated more into the design verification process. If a machine design does not lend itself to system fault diagnosis and component fault detection, then many problems arise. For this reason, the requirements of testing the machine must be fed back into the design process as early as possible. One approach is to use the same stimulus used by the design verification simulators as test patterns for fault simulation. If the patterns detect only a small percentage of faults, then an incomplete job of design verification or an untestable design may be indicated. Quite possibly the answers to testability and diagnosis reside in performing fault simulation as soon as possible after a logic level design is firm, or even during the comparison of alternative design approaches.

The change in engineering design philosophy is largely due to the inadequacies of past diagnostic capability. Tests were derived in the past, in some cases, after the
machine had been built. They were, as a consequence, not fully effective in their coverage. The obvious solution is to constrain the initial system and engineering designs by the diagnosis requirements.

An open-ended list of future goals can now be stated for implementation of LSI in the next decade; they include:

1. techniques and software for verifying that system and architectural specifications have in fact been implemented in a machine.

2. highly interactive, terminal oriented, and integrated computer aids for design to reduce user training and turnaround, resulting in usability.

3. advanced and enhanced simulation techniques for verification of designs; particularly multi-level simulation, logic-level simulation, and more effective techniques for timing analysis, at reduced cost.

4. advanced testing capability.

5. a “paper design” capability using software which provides very high engineering confidence and paves the way for the reduction or elimination of engineering change activity, at reduced cost and on a shorter design cycle.

6. Automatic input generation (particularly test pattern generation) and automatic output analysis facilities to relieve the burden of the designer.

REFERENCES

1 M E CONWAY L M SPANDORFER
A computer system designer's view of large-scale integration
Proc AFIPS Vol 33 pp 835-845 1968

2 E B EICHELBERGER
Hazard detection in combinational and sequential switching circuits
IBM Journal of Research and Development Vol 9 No 2 March 1965

3 P W CASE H H GRAFF L E GRIFFITH
A R LECLERCQ W B MURLEY T M SPENCE
Solid logic design automation
IBM Journal of Research and Development Vol 8 No 2 April 1964

4 GPSS V User's Manual
Prog No 5734-X52 (OS)

5 L H TUNG
A unit delay logic timing simulator
IBM Systems Development Division Technical Document TD 01.509

6 M J KELLY
Variable mesh simulator
Digest of the Second Conference on Application of Simulation pp 294-296 December 2-4 1968

7 M A BREUER
General survey of design automation of digital computers
Proceedings IEEE Vol 54 No 12 December 1966

8 K A DUKE H D SCHNURMANN T I WILSON
System validation by three-level modelling synthesis
IBM Journal of Research and Development Vol 15 No 2 March 1971

9 S A SZYGENDA D M ROUSE E W THOMPSON
A model and implementation of a universal time delay simulator for large digital nets
Proc AFIPS Vol 36 pp 207-216 1970