Using a hardware simulation engine for custom MOS structured designs

by Z. Barzilai
D. K. Beece
L. M. Huisman
G. M. Silberman

Mixed-level simulation techniques are widely used in VLSI designs for verification and test evaluation. In this paper we indicate how to perform mixed-level simulation on structured MOS designs using the Yorktown Simulation Engine (YSE), a hardware simulator developed at IBM. On the YSE, simulation can be done at the functional, gate, and transistor levels. The design specification used by the YSE is well suited for mixed-level simulation, particularly with regard to interfacing the different levels. We apply our techniques to an nMOS design to show the important features of our approach.

Introduction

Advances in silicon technology have led to chip designs consisting of hundreds of thousands of devices on a single chip. Typically, these chips have relatively few probe points compared to the number of devices they contain, thereby complicating debugging and testing. Moreover, the manufacturing costs of such complex designs are high. These factors have led to extensive software modeling prior to hardware manufacture. Software models are used to analyze the design in ways not possible or practical with actual chips or hardware (e.g., TTL) models.

Many design techniques use simulation to exercise the circuits for some specific purpose. The two most common applications are in design verification and fault simulation. In the former, the design is simulated to verify that it performs a correct function, while in the latter manufacturing test cases are evaluated for effectiveness in detecting assumed physical defects.

The design methodology strongly influences the kind of modeling and simulation employed. Most VLSI chips are designed using a *structured*, top-down approach. The chip is partitioned into a set of macros, each of which can be modeled independently and at different levels of complexity. Three levels of complexity are commonly used: the functional or behavioral, Boolean or logical, and transistor or device level. If the simulation uses one modeling level for all of a design it is said to be flat, or single-level. In a mixed-level approach, different parts of the design are specified at different levels.

Depending on the size of the design and the modeling techniques employed, the computer resources used during simulation can be astronomical. In order to reduce the large expense of simulation, specially architected computers have been developed for high-speed simulation [1–5]. These hardware simulators have the capability to simulate large designs at very high speeds. For example, a fully configured Yorktown Simulation Engine (YSE) can simulate more than two million logic gates at more than three billion gates per second [2].

We indicate in this paper how a hardware simulation engine can be applied to custom MOS designs. As a specific example, we consider the use of the YSE. First, we describe the structured design approach and how simulation is used for verification and test coverage evaluation. Next we outline how the YSE models are created for different design levels and how the different levels are interfaced. Finally, we describe a sample nMOS design which illustrates these techniques.

Simulation and custom MOS structured design

In the early phases of a design, each macro might be modeled using a high-level behavioral description of relatively low complexity and detail but high functional content. As the design progresses, different pieces are designed at the Boolean level, using logic gates which are readily and efficiently implemented with a given technology, such as NAND and NOR (see **Figure 1**). Ultimately, the circuits in each logic gate are described at the transistor level, sized, and implemented. This top-down method allows attention to be focused on local design considerations, but always in the context of the global environment.

The functional level describes the macro in terms of objects which are close to the chip's architecture. A common representation expresses the function of a macro as interconnected storage elements, called registers (or latches). Information is transferred between different registers on specific control conditions. The structure of this level is often influenced by specific technology considerations, but the specification itself is usually independent of technology.

A Boolean representation expands the functional objects into the logical expressions which implement the latches and the transfers defined in the functional level. Technology-dependent parameters, such as propagation delays, are often included.

The transistor level gives the implementation of the macro's function in terms of the actual devices used on chip. The transistor level is closest to the actual artwork used for chip fabrication and as a consequence requires specific technology-dependent information.

Certain MOS circuits require careful modeling for a correct Boolean specification because of the special characteristics of MOS technology, i.e., the bidirectionality of the MOS transistor and the sharing of charge between

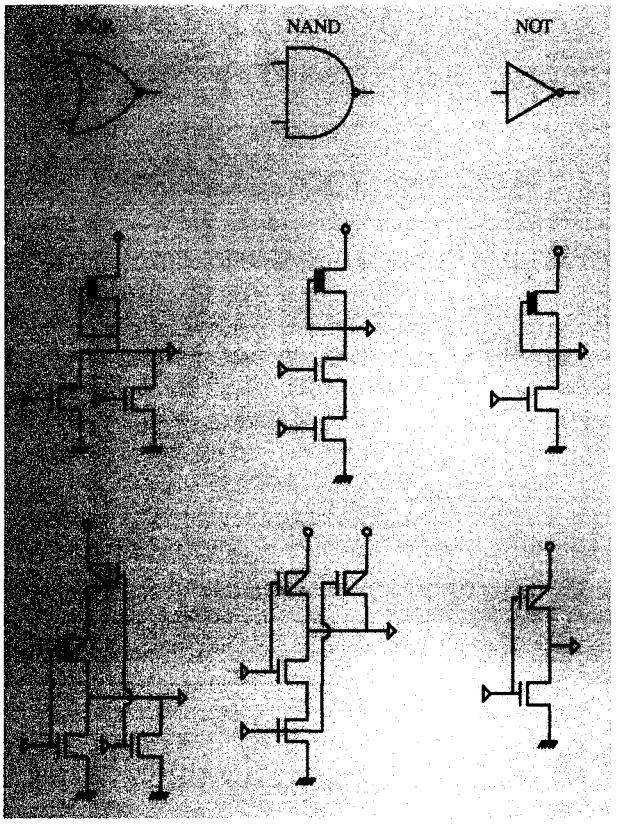

Figure 1

Logic gates for nMOS and CMOS: The NOR, NAND, and NOT logic gates (top) are three common circuits used in both nMOS (middle) and CMOS (bottom).

nodes that are effectively isolated from external current sources.

As a simple example, consider the nMOS multiplexer circuit shown in **Figure 2**. The logical expression $(X \wedge A) \vee (Y \wedge B)$ is not accurate, since conditions where $X = Y$ can result in potential ambiguities:

- The output when $X = Y = 1$ depends on the relative conductances of the transistor (the less resistive path dominates).
- The case $X = Y = 0$ decouples the output from the external sources, resulting in a memory state which depends on the previous history of the circuit.

Problems with such circuits often result in restricting their use to the interior of macros and/or strict design rules to ensure that ambiguities do not occur (i.e., to ensure that $X \neg= Y$ in the above example).

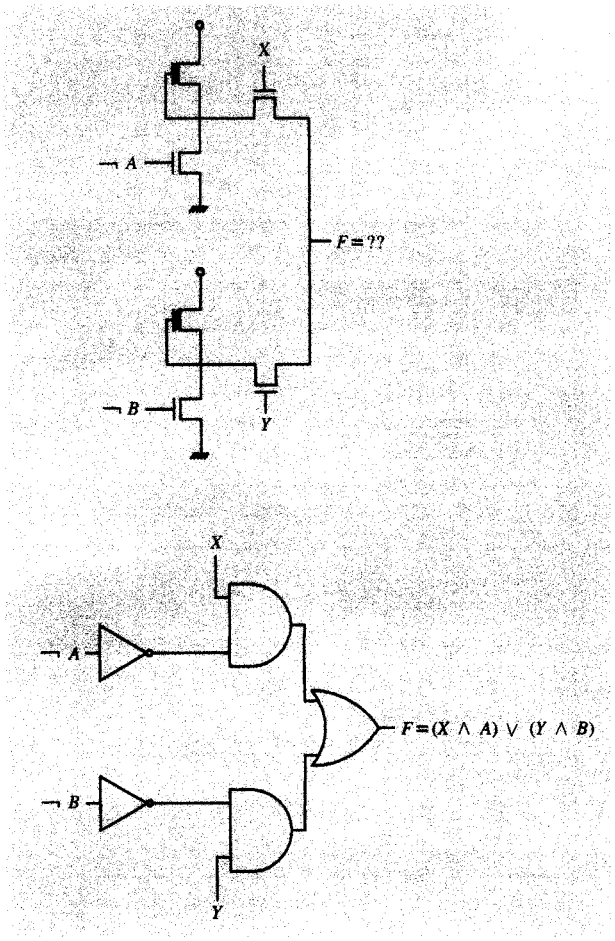

Figure 2

nMOS multiplexer circuit: The nMOS multiplexer circuit shown in (a) does not have the logical representation shown in (b) unless it can be guaranteed that $X \neg = Y$.

much more efficient. Functional-level simulation is even more efficient, as many logical gates can be combined into latches and control lines. Because of this efficiency/accuracy tradeoff, simulation at the functional and Boolean level is not often used to determine how fast a particular transistor or logic gate switches. Instead, a *unit delay* discrete simulation is usually employed to ensure that the design performs correctly under a wide variety of input patterns and control sequences.

• *Mixed-level simulation*

Mixed-level simulation combining the functional and Boolean levels is commonly used in chip designs, while it is less common to see a mixture of these with the transistor-level simulation. The reason for this is the similarity between internal representations used by the simulator for both the functional and Boolean levels, which allows for a straightforward interface between these two levels. On the other hand, the representations and algorithms used for circuit simulation are substantially more sophisticated. The key to mixing transistor-level simulation with the Boolean/functional simulation is to either make the functional/Boolean simulation continuous or make the circuit simulation discrete. We use the second alternative, since the discrete simulation is inherently faster and more suited to the YSE architecture.

A discrete simulation for MOS transistor networks can be achieved by replacing the transistors by switches [8, 9]. While not applicable to all transistor circuits (such as operational amplifiers), switch-level simulation (SLS) correctly predicts the behavior of digital circuits commonly used for logic chips. Because SLS is a discrete simulation, its interface with the functional/Boolean level is straightforward, yet it maintains a close correspondence to the actual circuits.

For design methodologies that do not use a technique like SLS, the only way to connect the functional simulation with the transistor-level simulation is by mapping the transistor network into a set of Boolean equations. This is straightforward when the transistor network was obtained from a corresponding Boolean level, but is often artificial when the Boolean equations are derived from the transistor network itself.

• *Simulation for design verification*

Simulation is used for design verification to ensure that the design has the correct, i.e., intended, function. First, the design is described at the functional level for each macro and simulated. Next, macros are designed physically and mixed-level simulation is run using a lower-level description for each macro.

With SLS it is in fact possible to avoid building the Boolean specification and work directly with the transistor-level description for many macros. In addition, since the

• *Continuous and discrete simulation*

The accuracy required of simulation has to be balanced by its cost, which is usually measured in terms of CPU time and memory requirements. For example, detailed electrical circuit simulation [6, 7], using parameters estimated from actual wafer dimensions, is required to accurately predict transistor characteristics, such as switching speed and power consumption. While highly accurate, the cost of circuit simulation is so high that it becomes impractical for more than a few hundred transistors at one time.

Circuit simulation provides a continuous prediction, e.g., a waveform, of the response of the design. Gate-level simulation, on the other hand, approximates the response of the design by discrete quantities. By definition, gate simulation is less accurate than circuit simulation, but is

transistor description used in SLS either can come from the designer or can be extracted directly from the mask artwork using standard device recognition programs, SLS can also be used to verify (by simulation) the physical design of complete macros.

• *Fault simulation*

The goal of fault simulation is to evaluate a set of test cases to be used during manufacturing. In fault simulation, a fault model is used to approximate the effect of certain manufacturing defects. These defects, called faults, are introduced into the software model of the design; the faulted model is then simulated and the results compared to the good machine simulation results. If the two models produce different outputs, the fault is said to be covered by the test patterns used during the simulation.

Fault simulation requires a description of the circuit which can be correlated with real devices. For gate-level descriptions which are good structural representations of the actual circuits, such as NAND gates, the determination of which nodes in the model to fault and how to fault them is straightforward.

For those MOS circuits which are difficult to model using Boolean expressions, the identification of faults is also difficult. In SLS, however, the description is close to the actual structure of the design, and thus fault identification is easier. Therefore, SLS allows accurate coverage estimates of test cases for these difficult MOS circuits.

Model building for the YSE

We begin by outlining some properties of the YSE architecture [1, 2, 10]. The basic design of the YSE is oriented toward high-speed 4-valued logic simulation at the gate level, e.g., 0, 1, unknown, and tri-state. The YSE itself (see **Figure 3**) consists of a collection of identical logic processors (LPs) along with an interprocessor switch. Each LP executes its own program, but in lock-step fashion, using the switch for interprocessor data communication.

A YSE instruction can accommodate four 2-bit-wide inputs and produces one 2-bit output, according to a user-programmable truth table. In addition, all inputs and the output may be further modified by using de Morgan fields, which indicate a user-programmed transformation to be applied to the appropriate value before its usage (when input) or storage (when output). For example, inputs and/or the output can be inverted, set to a constant, etc.

Using the YSE as the target machine for algorithm execution makes it necessary to serialize the simulation process, i.e., avoid the use of conditionals. Also, iterations which have to be carried out a variable number of times (depending on current external input) have to be avoided as much as possible. The reason for this is the high overhead incurred by these operations when running on the YSE (for more details see [1, 2, 10]).

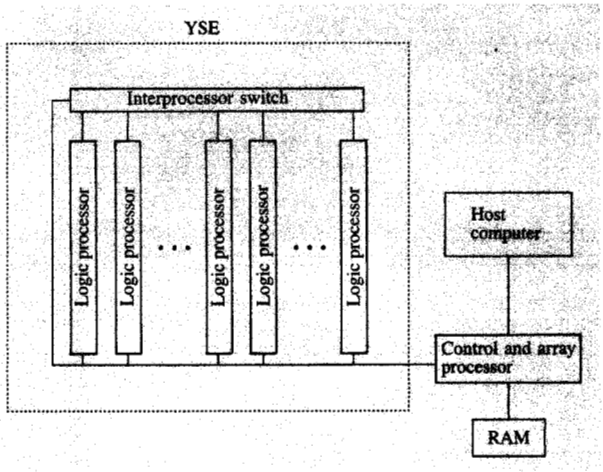

Figure 3

YSE architecture: The YSE consists of a set of logic processors, connected through an interprocessor switch, and linked to a host machine.

Simulation on the YSE is thus conceptually similar to software-compiled code simulation. In order to simulate on the YSE, a design must be translated into a special YSE assembly language, which is then compiled into the appropriate machine code, linked, loaded, and then simulated on the YSE (see **Figure 4**).

• *Functional and logical model building*

Functional and Boolean model building on the YSE are very similar. YSE functional model building uses logic synthesis techniques [10] to obtain a Boolean equivalent description from a suitable register transfer level (RTL) language. YSE Boolean model building uses the macro facility of the YSE compiler, whereby multiple input-output logic gates can be expanded into equivalent sets of four-input, one-output logic gates. These reduced logic gates are then mapped on YSE instructions, as the YSE functions (i.e., operation codes) are fully programmable.

• *Arrays*

While it is possible to simulate arrays (ROM and RAM) at the device or gate levels, for many applications this amount of detail is not necessary. The YSE has special assist hardware (see Fig. 3) which allows it to simulate large arrays. Special software allows us to specify arrays at the functional level; the declarations are then converted into the appropriate instructions for the hardware.

• *Transistor model building*

We model transistors on the YSE using a switch-level model. Details of our approach have been described elsewhere [9], so only a summary is presented here.

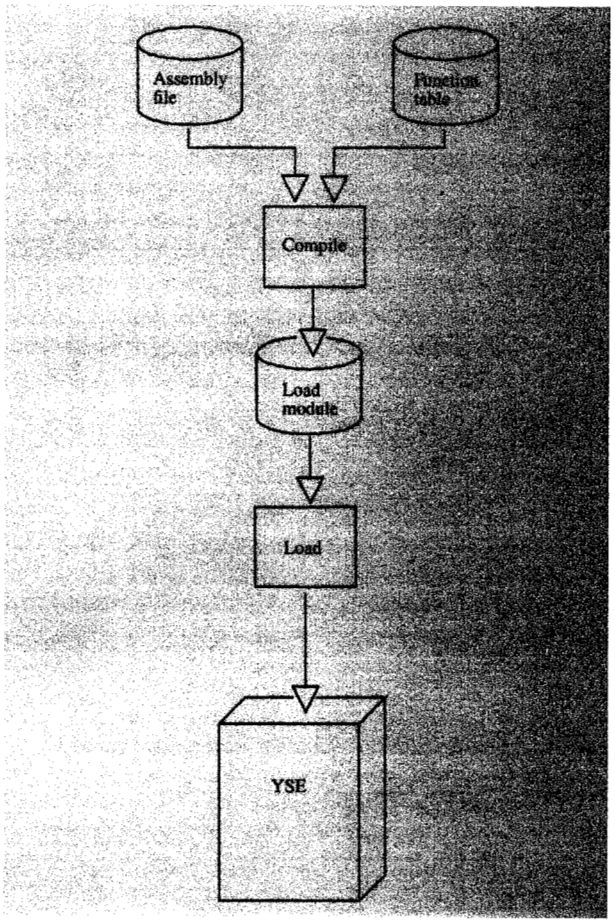

Figure 4

YSE model generation: Designs to be run on the YSE are described in a YSE assembly language file, which is then compiled, linked, loaded, and simulated.

```
DO FOR (all inputs)
  apply inputs to YSE
  DO WHILE (not consistent)
    DO FOR (all transistors)
      update transistor state
    END
  DO WHILE (not converged)
    DO FOR (all transistors)
      update node states
    END
  END
collect outputs from YSE
END
```

Figure 5

YSE SLS algorithm: The SLS algorithm for the YSE consists of a node update phase and a transistor update phase.

Switch-level model

In the switch-level model [8], a circuit is a network of nodes interconnected by transistors (switches). Nodes are either external (i.e., inputs, V_{dd}, Gnd) or internal (all others). External nodes are not affected by the circuits they drive; internal nodes have an associated capacitance, reflecting the charge-sharing property of MOS circuits.

The transistors in this model are considered to be bidirectional switches in one of three possible states: open, closed or unknown, depending on the transistor's gate value. The source and drain terminals of a transistor interact in a manner which depends on the transistor's state and conductance. This interaction also depends on the *strength* of the signals present on each node. For example, driven nodes dominate those with stored charge, and signals through enhancement devices are usually stronger than those through depletions.

After modifying external inputs, the circuit undergoes changes until it finally reaches a steady state (if it does not oscillate). In order to simulate this process (see **Figure 5**), we perform a series of two-phased iterations [9]. The first phase of an iteration is the so-called transistor update process (or *T-phase*), and the second phase consists of the node-updating procedure (or *N-phase*).

During the T-phase, the current state (0, 1, X) of each node connected to a transistor gate is used to determine the transistor's state. For n-type enhancement devices, a gate value of 0 opens the switch (i.e., the transistor is not conducting), while a 1 value closes the switch (for a p-type device, a gate value of 1 opens the switch and a value of 0 closes it). For the case where the gate node is in an X state, we set both source and drain nodes of the transistor to an X state. This approach is pessimistic since Xs propagate through the simulated network further than they would in the actual circuit. Nevertheless, this has proven very effective in uncovering mistakes during design verification. More sophisticated approaches to the X state are also possible.

Once determined during the T-phase, the transistor states remain constant for the complete iteration, even if the appropriate nodes do change states.

After the T-phase is completed, the N-phase is executed, according to the following principles:

- Changes in external inputs induce a dynamic partitioning of the network into sets of nodes which are connected through conducting transistors. These sets should attain common states in the process of bringing the entire network to a steady state (if the circuit does not oscillate).
- Our approach is to reach common states for every set of nodes in a manner which is independent of the current transistor settings. To do so, we perform (fixed) pairwise interactions between nodes which share a transistor.
- The convergence of all nodes to common set states may require several passes through the network (i.e., repeated

interactions between adjacent nodes), depending on the network topology and the input pattern.
- The above iterations must be repeated for circuits containing feedbacks and/or transistors with gates controlled by internal nodes, until a steady state is obtained.

From physical design to YSE transistor simulation

Figure 6 shows the various steps that are necessary to generate the YSE instructions for SLS. We require a flat nodal description of the transistors and a list of the primary inputs. Some physical information about the transistors is also used, such as their types (enhancement or depletion) and the type of their carriers (p or n). The description can come either from physical layout or a hierarchical network definition [6]. In the first case, the transistor network is extracted from the shapes, while in the second case it is obtained by denesting.

SLS algorithms for computing the node states require certain structure analysis to be performed on the network. The analysis is done to enhance the efficiency of code generation and simulation itself. There are essentially three phases:

1. The first phase partitions the network into components called groups. Groups are directed in the sense of a logic gate, such as an AND gate, in that it has well-defined inputs and outputs. A switch, by definition, does not cross a group boundary because of its bidirectional nature.
2. The next phase orders the groups. Optimizing this order with respect to many objective functions, such as reducing the number of global feedbacks, is NP-hard. We employ a heuristic based on the use of an immediate dominator list for each group to reduce, in a game-like fashion, the number of global feedbacks.
3. The last phase imposes an orientation on each transistor, i.e., a direction for propagating information in each pass, and assigns an ordering for the transistors in each group. As was the case for global ordering, optimally ordering and orienting transistors to minimize the number of iterations are NP-hard problems. Our ordering and orientation heuristic is described in detail in [11].

- *Level interfacing*

Up to now we have discussed only the model building for single-level models. As discussed in the section on mixed-level simulation, one often wants to simulate parts of a design on different levels. This introduces the problem of how to interface these different parts. For the three different levels that we have discussed so far, the interfacing requires some care.

We interface the different levels by using correspondence of net names, which ensures the equivalence of values across the level boundaries. This causes no problems when interfacing the functional and logical levels, because both use the same bit-encoding scheme for logical values. On the YSE, the same approach works for the interface between a switch-level block and a Boolean or functional-level block, because the SLS encoding uses one YSE data word for the logical value and a different data word for the strength of the signal.

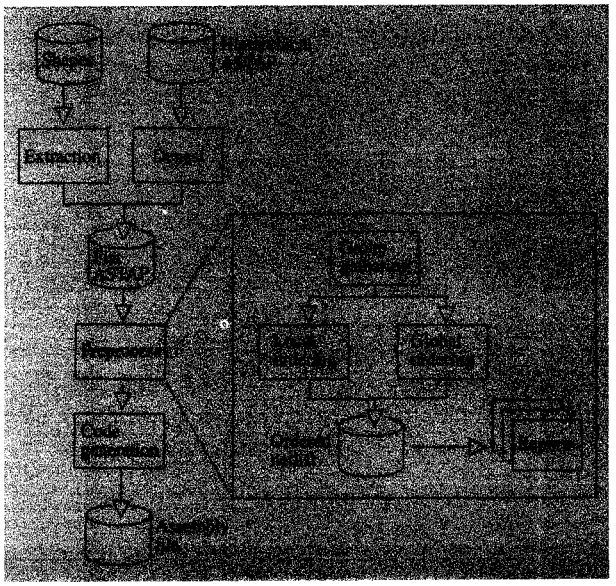

Figure 6

SLS processing steps: The NODES file is generated from the nodal description of the network by a participating/scheduling algorithm.

Propagation of signal strengths between the functional/Boolean level and the transistor level does require some special consideration. Even the propagation of the signal value itself across such an interface depends on the strength of the sending and receiving signals. Consequently, we have to include some information about the strength of those nets that are outputs of a Boolean or functional block and are connected to the source or a drain of a transistor.

Example

The methodology presented above has been exercised on a small example: a 4-bit slice of a Manchester carry chain ALU (adapted from [12]). The circuits are shown in **Figure 7**, for a bit slice and part of the carry chain. Each bit slice has two data inputs, which are controlled by select lines, and contains three universal logic function blocks. Two of these blocks determine the *kill* and *propagate* bits for the carry of each bit, while the third block is used to obtain the result. The function to be performed by the ALU is selected by supplying the appropriate controls to the *kill*, *propagate*, and *result* logic function blocks.

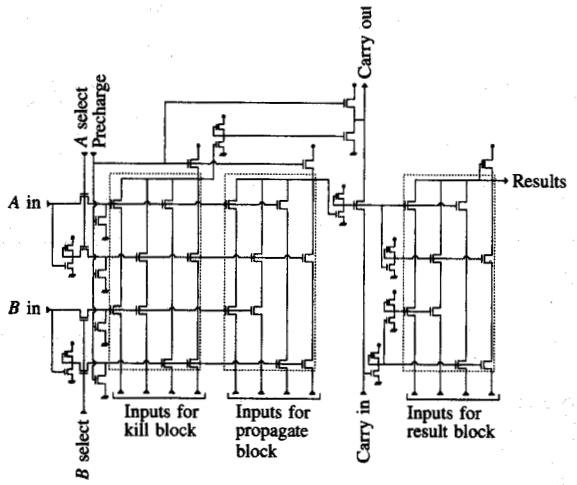

Figure 7

ALU circuit: A single bit slice of the ALU (adapted from [11]).

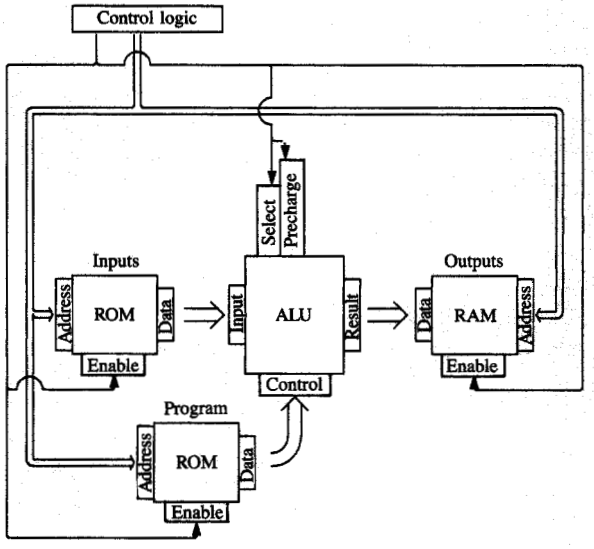

Figure 8

Block diagram of the embedded ALU.

In our example, the ALU is embedded in a larger piece of logic that contains two ROMs, a RAM, and some control logic (see **Figure 8**). The ROMs hold input vectors and corresponding control vectors for the ALU. The control vectors exercise the ALU by having it calculate several functions with different inputs.

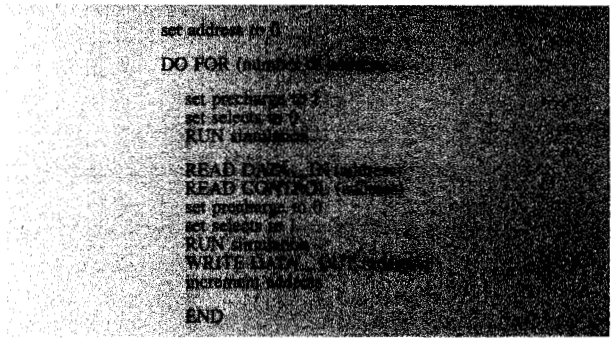

Figure 9

Control logic: The run protocol for the ALU model.

The arrays and control logic are described first at the RTL level. From this RTL description, a set of YSE instructions is obtained using a logic synthesis program. This set of YSE instructions is then merged with those that model the ALU itself at the switch level. The strength of the input vectors and control vectors is assumed to be higher than the strength of any signal in the ALU, while the strength of the output signals from the ALU is assumed to be higher than that in the output RAM. In this way, there is a clear flow of information from the input ROMs, through the ALU to the output RAM.

The simulation protocol proceeds as indicated in **Figure 9**. First the precharge signal is set high and the select lines low, then the YSE instructions are executed to simulate the precharge phase. Next, the inputs and the control bits are read in from the ROM, the precharge signal is set low, and the YSE instructions are again executed to simulate the actual computation phase. Finally, the resulting outputs are stored on the RAM.

The ALU proper contains 192 transistors and uses 900 YSE instructions. The two ROMs and the RAM are each 1024 by 16-bit word arrays. The read-write control logic needs 45 YSE instructions and the array address decode and enable logic takes 131 YSE instructions. Because of the precharge phase, the full set of YSE instructions has to be executed twice for each functional cycle. The resulting simulation time per functional cycle is 0.2 milliseconds.

The model described here was intended for fast simulation of many cycles on the ALU. However, its structure is rather general and is a good example of mixed-mode simulation. In fact, the ALU could not have been simulated on a higher level than the switch level without extensive virtual logic. Also, the control section could not have been as conveniently specified on a level lower than the functional, without a great increase in the number of YSE instructions.

Summary

We have indicated how the Yorktown Simulation Engine (YSE) can be used to perform mixed-level simulation on structured MOS custom designs. On the YSE, we can mix descriptions at the functional, Boolean, and transistor levels in the simulation. A variety of tools are used to support this simulation, including circuit extraction and analysis programs, compilers for the production of YSE code, and monitors to interface with the user during simulation.

The motivation for modeling designs at three different specification levels is quite clear. On the one hand, the size of VLSI designs makes their detailed circuit-level simulation prohibitive; on the other hand, their complexity requires as much detailed simulation as possible, in order to reduce manufacturing costs because of design mistakes. Mixing levels allows simulation of parts of the design in detail, while keeping the environment (i.e., the other parts of the design) realistic but less detailed.

YSE model building at the different levels and their linkage into executable code have been described. Particular emphasis was given to the switch-level model. Finally, an example was shown to illustrate the capabilities of the mixed-level approach.

Acknowledgments

We would like to thank E. Kronstadt and P. Osler, for helpful discussions about the YSE software, and M. Y. Tsai, for assistance in constructing the transistor-level description of the ALU.

References

1. G. F. Pfister, "The Yorktown Simulation Engine, Introduction," *Proceedings*, 19th Design Automation Conference, 1983, pp. 51–54.
2. M. M. Denneau, "The Yorktown Simulation Engine," *Proceedings*, 19th Design Automation Conference, 1983, pp. 55–59.
3. J. K. Howard, R. L. Malm, and L. M. Warren, "Introduction to the IBM Los Gatos Logic Simulation Machine," *Proceedings*, IEEE International Conference on Computer Design: VLSI in Computers, 1983, pp. 580–583.
4. *ZYCAD Logic Evaluation General Description Manual,* AA-001 Rev A, ZYCAD Corporation, 2825 Fairview Avenue North, Roseville, MN 55113.
5. T. Sasaki, N. Koike, K. Ohmori, and K. Tomita, "HAL: A Block Level Hardware Logic Simulator," *Proceedings*, ACM IEEE 20th Design Automation Conference, 1983, pp. 150–157.
6. *Advanced Statistical Analysis Program, Program Reference Manual,* Order Number SH20-1119-0, available through IBM branch offices.
7. L. Nagel, "SPICE 2—A Computer Program to Simulate Semiconductor Circuits," *Technical Report UCB ERL-M250*, Electronics Research Laboratory, University of California, Berkeley, May 1975.
8. R. E. Bryant, "A Switch Level Model and Simulator for MOS Digital Systems," *IEEE Trans. Computers* **C-33**, No. 2, 160–177 (February 1984).
9. Z. Barzilai, L. Huisman, G. M. Silberman, D. T. Tang, and L. S. Woo, "Simulating Pass Transistor Circuits Using Logic Simulation Machines," *IEEE Design and Test of Computers* **1**, No. 1, 71–81 (February 1984).
10. E. Kronstadt and G. Pfister, "Software Support for the Yorktown Simulation Engine," *Proceedings*, 19th Design Automation Conference, 1983, pp. 60–64.
11. I. Spillinger, "Pass Transistor Simulation on a Sequential Logic Simulation Machine," M.Sc. Thesis (in Hebrew), Electrical Engineering Department, Technion, Haifa, Israel, May 1984.
12. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, Reading, MA, 1980.

Received December 9, 1983; revised March 15, 1984

Zeev Barzilai *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Barzilai received the B.Sc., M.Sc., and D.Sc. degrees in computer science from the Technion, Israel Institute of Technology, Haifa. He was on the faculty of the Computer Science Department at the Technion from 1978 to 1980. Currently, Dr. Barzilai is the manager of the simulation and testing group of the Computer Science Department at the IBM Thomas J. Watson Research Center. His research interests include verification and testing methodologies for VLSI systems.

Daniel K. Beece *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Beece is a Research staff member in the Computer Science Department at the Thomas J. Watson Research Center. His research interests include hardware and software simulators, and verification and testing methodologies using simulation-based techniques. Dr. Beece received the B.S. (with distinction) in engineering physics from Cornell University, Ithaca, New York, and the M.S. and Ph.D. degrees in physics from the University of Illinois, Urbana. He is a member of the American Physical Society and the Biophysical Society.

Leendert M. Huisman *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Huisman received the Ir. degree in technical physics from the Technische Hoge School, Delft, the Netherlands, in 1973. He holds a Ph.D. degree in physics from Harvard University, Cambridge, Massachusetts, obtained in 1979. From 1978 to 1980 he was a postdoctoral research fellow at Brandeis University, Waltham, Massachusetts. Between 1980 and 1982, he was employed by the Stichting voor Fundamenteel Onderzoek der Materie in the Netherlands. Since 1982, he has been employed by IBM at the Thomas J. Watson Research Center, where he is working on VLSI testing and verification. Dr. Huisman is a member of the American Physical Society and the IEEE Computer Society.

Gabriel M. Silberman *Technion–Israel Institute of Technology, Haifa, Israel.* Dr. Silberman received the B.Sc. (*cum laude*) and M.Sc. degrees in computer science from the Technion–Israel Institute of Technology, Haifa, in 1975 and 1976, respectively. He holds a Ph.D. degree in computer science from the State University of New York at Buffalo, obtained in 1980. From 1976 to 1977 he was employed by the Armament Development Authority, Israel Ministry of Defense. Since 1980, he has been with the Departments of Computer Science and Electrical Engineering at the Technion, as an assistant professor. During the summer of 1982 he was with the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, where he is currently a Visiting Scientist. His current research interests include computer architecture, operating systems, VLSI design verification, and fault simulation. Dr. Silberman is a member of the Association for Computing Machinery.