Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Improving digital circuit simulation: A knowledge based approach

John A. Benavides
Dana L. Wyatt
Department of Computer Science
University of North Texas
Denton, Texas 76203

ABSTRACT

The simulation of digital logic circuits is a common practice in design automation. Because of advances in integrated circuit technology, currently available digital circuit simulator can not model all circuit behavior. Combining artificial intelligence and simulation techniques, a knowledge based simulator was designed and constructed to model non-standard circuit behavior. Circuit designer expertise on behavioral phenomena is used to diagnose, analyize, and emulate this behavior in the knowledge based simulator. An expert system guides the base simulator by manipulating its events in order to achieve the desired behavior. This paper focuses on the prototype system architecture which integrates features of a an event-driven gate-level simulator and features of the multiple expert system architecture, HEARSAY-II.

INTRODUCTION

Simulation in the electrical engineering domain serves multiple purposes. With a simulator, an engineer can analyze a circuit before investing time and money in manufacturing. Once correct functionality of the design is established, the engineer can also use the simulator to check its performance speeds, as well as the performance of manufacturing tests (Terman 1987 and Trimberger 1987).

Currently available simulators for digital circuit design work with different levels of circuit abstractions. The lowest level of abstraction models the circuit based on the physics of its components (Nagel 1975). While this type of continuous simulation is very accurate, the computation cost prohibits its use to small circuits, containing no more than several thousand circuit devices (Terman 1987). Simulators with higher levels of abstraction model the circuit at a switch level, logic gate level, functional block level, or at a behavioral level and use a discrete simulation methodology. However, as the level of abstraction increases, computation cost decreases but so does simulation accuracy (Miczo 1986).

As the speed of circuit components increases, circuit behavior breaks from clean digital switching characteristics. The problems of simultaneously switching outputs, transmission line reflections, power supply noise, and crosstalk between signals become major design concerns (TI 1987). The ability to simulate these problems requires the modeler have an expertise with these phenomena.

Several methods of modeling these phenomena are possible. The first would be a reduction of the entire circuit to a physics level model-abstraction. Then, a continuous simulator would exhibit the non-standard digital circuit phenomena. However, because of the prohibitive computation costs, this can not be done with large circuits. A second method would be a combination of the abstraction levels, a so-called mixed-mode simulator (Bloom 1987). By simulating portions of a circuit at the physics level and portions at a digital circuit level, a designer might be able to observe these phenomena. However, this method requires the designer be an expert with these phenomena in order to recognize where to partition the circuit. The research reported in this paper is aimed at pursuing a third method using new simulator technology that combines artificial intelligence techniques with simulation. This method encapsulates expert knowledge of known phenomena in a knowledge based simulator in order to model non-standard behavior and avoid difficulties of the previous methods.

The prototype knowledge based simulator reported herein combines an event driven digital circuit simulator with a knowledge based expert system. The digital circuit simulator brings with it efficient analysis of coupled components behavior. This is standard behavior and results from formal connections. The expert system recognizes, analyizes, and initiates the simulation of behavior resulting from unformal connections. This is possible because such non-standard behavior may be linked to physical, structural, or topographical information not ordinarily used by the digital circuit simulator. The expert system steers the simulator into non-standard behavior by modifying its events during operation. This results in future behavior which reflects a circuit's reaction to the initial non-standard behavior.

This knowledge based simulator is envisioned as a design tool which can be used early in the design process. By discovering and understanding non-standard behavior, a designer may incorporate appropriate design in order to avoid potential problems and prevent the expense of manufacturing a non-functional circuit. The following section explores previous research efforts of combining knowledge bases and simulation.

Knowledge Based Simulators

Murray and Sheppard (1988) note that knowledge based simulation systems employing artificial intelligence techniques have been designed for several purposes. These include:

1) Computer-assisted model construction.
2) Decision-support functions.
3) Intelligent front-ends for assisting in simulation experimentation and analysising simulation results.

4) Complete simulation environments for supporting the simulation model life cycle.

Two approaches have been used to achieve these capabilities.

One approach has been to use a knowledge base expert system as the base system. Murray and Sheppard (1988) built a computer-assisted model constructor system for SIMAN models by combining model construction knowledge and SIMAN knowledge in an expert system. Moser (1986) implemented a decision-support system using an expert system knowledgeable on corporate business to analyize simulation results of a corporate model. Borden and Hugh (1985) incorporated temporal knowledge into an OPS5 expert system in order to simulate electronic warfare threats. System architectures which use expert systems as their base represent knowledge in rule bases and through inference strategies.

The second design approach for knowledge based systems has been to use an object-oriented programming paradigm. Kim and Zeigler (1988) note this approach is popular because of its similarity to the discrete event formalisms. In object-oriented programming, objects represent world entities containing methods and properties. These methods are behavior rules used to respond to messages representing world changes. All communications between objects is accomplished with messages which may represent passage of time, changes in object relations, or requests for object action. If an object does not contain a method for a specific message, it may inherit a method from a object defined to be superior to it in its object hierarchy (Adelsberger et al. 1986). One example of this approach is ROSS which was designed by the RAND corporation (Klahr and Waterman 1986). ROSS supports the simulation model life cycle and has been applied to military battle simulations. KBS is a second example of an object-oriented system (Reddy et al. 1986). KBS is a decision support environment for intelligent management of business operations. Simulation, model verification, and result analysis are all parts of the KBS environment. Systems using object-oriented programming, represent knowledge through methods, object properties, and inheritance.

SWIM Prototype

A prototype simulator called SWIM, Simulate What I Mean, has been constructed on a Texas Instrument's Explorer Workstation in Common LISP. Using this workstation environment has facilitated rapid programming, debugging, and testing of a prototype system.

The approach used in the SWIM prototype differs from the expert system shell and object-oriented language approaches because it integrates the expert system directly into the simulator. The expert system becomes a controlling component of the base simulator in order to increase simulation capability. In the following section, an overview of the system architecture is presented. Following the architecture description, an example using a knowledge source for simultaneously switching pads cells in a integrated circuit is described. The paper concludes with implementation notes and ideas on this research.

SWIM SYSTEM ARCHITECTURE

The prototype system developed at the University of North Texas integrates features of an event-driven gate-level simulator (Lightner 1987 and Dillinger 1988) and features of the multiple expert system architecture, HEARSAY-II (Hayes-Roth 1983). Using this hybrid architecture, simulation of a digital circuit is controlled through a menu-driven interactive window. From the simulation window, a designer can:

1) load a circuit description from a file into the simulator.
2) construct simulation stimulus for a circuit.
3) initialize a circuit's state by setting nets and memory element values.
4) collect simulation results in a file.
5) manipulate the expert system's knowledge base.
6) start, stop, or step through a simulation.
7) switch into a ZMACS text editor window to create or modify a circuit description.
8) save a simulation session for later reuse.

The major components of the system architecture are the world data base, the model base, the simulator, the expert system, and the evaluator (see figure 1). The world data base contains all information on the circuit being simulated. Both the evaluator and the expert system interrogate the world data base. The model base contains component simulation models. The simulator coordinates events, manipulates system time, tracks system operation, and schedules behavioral evaluation with the evaluator. The evaluator determines component behavior through model analysis and expert system consultation; when expert knowledge may be available the evaluator modifies its behavioral interpretation based on model analysis.

World Data Base

The world data base describes the circuit being designed to the simulator as well as the expert system. When a circuit is loaded into the simulator its corresponding knowledge is added into the world data base. There are four types of information in this data base: structural, functional, physical and dynamic. Structural information describes component connections (coupling). Components connected together will share a common net. Functional information maps a component to its simulation model. A simulation model describes the behavior of a component's output pins when its input pins are stimulated. Physical information describes the planned geometric properties of the final circuit. For example, two types of physical information are net lengths and component coordinates in the physical design space. Dynamic information describes the states of all nets connecting components in the circuit, as well as any memory devices that may exist in a component. During a simulation, nets and memory devices change their values to one of three discrete values: high, low, and unknown. Both the simulator and the expert system can access all four types of information.

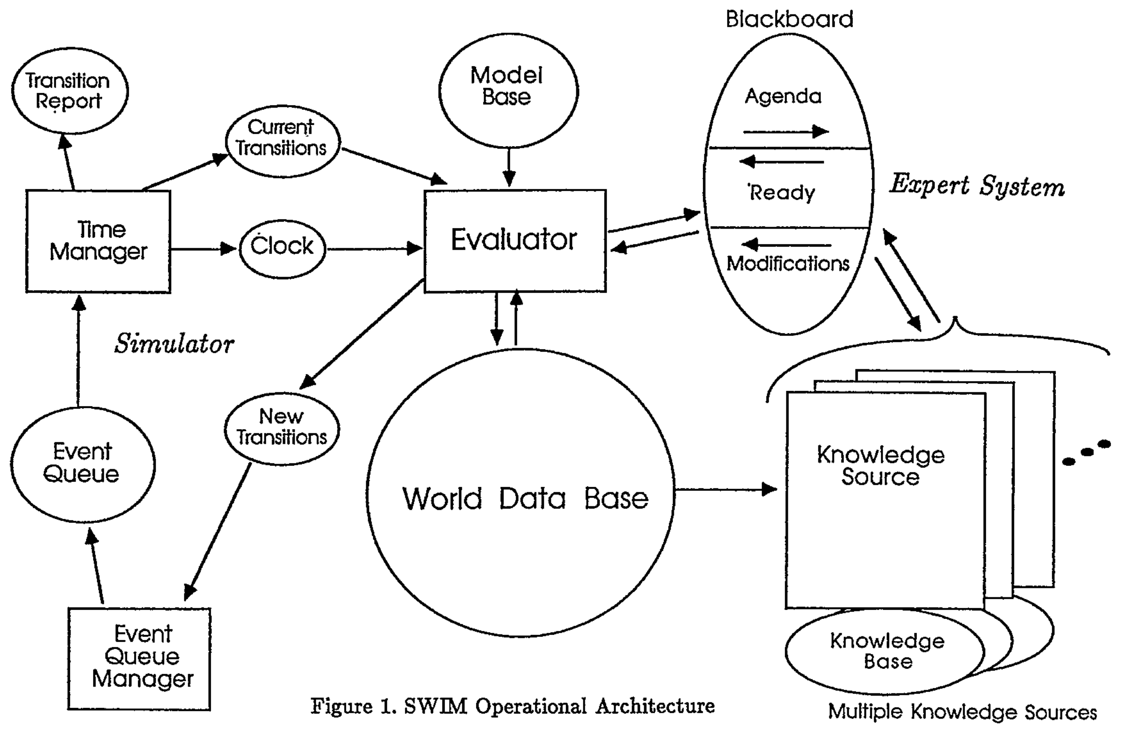


Figure 1. SWIM Operational Architecture

Because engineering design is an iterative process, not all information will initially be in the world data base. Using multiple simulations, a designer successively refines a design until it fulfills all its specifications. Consequently, design information not initially available in the world data base is added as a result of the design process. The physical information does not become available until the structural view of the circuit is converted to a physical representation. With the physical information available, a more accurate simulation model for the entire circuit is derivable. Simulation with the resulting circuit model verifies that preceding design decisions continue to support design specifications (Dillinger 1988).

The world data base fragments the four types of information into PIN, NET, and PART records. Figure 2 illustrates the Common LISP structures for these records. Pins represent the port through which a part (component) is connected to a net (signal). A pin can have one of three directions from the net to the part: in, out, or inout. All pins connected to a net are referenced in the pin list of a NET record. Similarly, all pins connected to a part are referenced in the pin list of a PART record.

During simulation, the records in the world data base are accessed in a NET, PIN, PART, PIN, NET order. When a net changes value, the parts connected to the net through an input pin are examined for behavior changes. A model defining the part's behavior is referenced in the PART record. Using this model, values for this part's output pins are determined. Consequently, these output pins may change other nets, and the access loop (NET, PIN, PART, PIN, NET) is repeated.

Model Base

The model base is a data base containing all component behavioral models. The SWIM prototype system is currently constructed with gate level behavioral models. Available component models represent logic functions AND, OR, NOT, and NAND, as well as memory elements D, SR, and JK flip flops. Each component model is a LISP function which returns a list of resulting event(s) when called with a list containing the current value of input pins. Increasing the level of component model complexity is planned as a future enhancement.

Simulator

The simulator of the system is a discrete-event simulator containing event and time handling utilities and a simulation clock. One utility, the event queue manager, maintains the event queue. Another utility, the time manager/sequencer, tracks the simulation time, generates trace information, and schedules event handling.

Simulation activity proceeds according to the following basic control algorithm.

1) The event queue manager passes all event(s) scheduled to occur next to the time manager.
2) The time manager updates the system clock, records these changes on the trace output, and schedules evaluation of all events with the evaluator.
3) The evaluator determines behavior of models being excited, consults the expert system for any simulation advice, and passes any resulting events to the event queue manager.

```
;;
(defstruct net
  "Signal net"
  (name 'unknown)
  (type 'core)          ;CORE or
                        ;peripheral net (IN, OUT, INOUT)
  (value 'x)            ;Current net value
  (change-time 0)       ;Time of last change (Absolute time)
  (pin-list nil)        ;All pins connected to this signal net
  (physical-list nil)   ;Physical information
  )
;;
(defstruct pin
  "Pin instance"
  (name 'unknown)
  (number 0)            ;Pin number
  (type 'inout)         ;Legal types are: in, out, or inout
  (part nil)            ;Points to owning part
  (net nil)             ;Points to owning signal net
  (physical-list nil)   ;Physical information
  )
;;
(defstruct part
  "Part instance"
  (name 'unknown)
  (type 'unknown)       ;Points to functional model
  (eval-bit nil)        ;A flag showing pending evaluation
  (pin-list nil)        ;All its own pins; ordered
  (physical-list nil)   ;Physical information
  )
```

Figure 2. Lisp structures for world data base.

Event Queue Manager. The event queue manager maintains the event queue. Events may represent net transitions scheduled at a specific time or re-ignition of a knowledge source at a specific time. During initial system set-up, the event queue manager constructs an event queue from simulation input waveforms. During system operation, the event queue manager inserts all new events in the event queue when delivered by the evaluator.

The event queue is an array of 500 lists treated as a circular queue, called a time wheel. Each list represents a time step, contains event(s) occurring at the same time, and will be accessed through its array index. One index of the array will represent the current time. The array index following the current-time array index will represent one time step into the future. Because the array is circular, the array index proceeding the current-time array index will represent 499 time steps into the future. Anything beyond 500 time steps is stored in an unordered list referred to as the "far list". The time wheel data structure was first designed by Ulrich and Suetsugu (1986).

During simulation, the event queue manager manipulates the time wheel. When all the events of the current time step are processed, time is advanced by incrementing the current-time array index until an non-empty event list is found. Each time the starting array index is reached, the far list is scanned once and any events occurring in the next 500 time steps are moved into the time wheel. When receiving new events, the event queue manager places any events scheduled for the next 500 time steps directly into the time wheel; remaining events go into the far list.

Time Manager. The time manager updates the simulation clock, generates trace information, and schedules event evaluation with the evaluator. The event queue manager sends the time manager a list containing event(s) scheduled to occur next in the simulation. The time manager then updates the clock to the time of these scheduled events. Next, any events occurring on nets designated to be traced by the simulation window interface are recorded in a trace file. The trace information includes the net name, the transition type, and the transition time. In addition, if the event was influenced by the expert system, this is noted in the trace file. Lastly, the time manager passes the list of events to the evaluator for behavioral analysis.

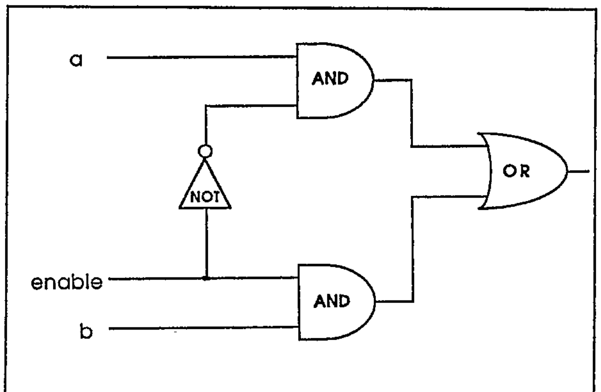
Clock. The clock is a globally accessible object which is used by the evaluator and expert system. The evaluator uses the clock time to calculate the time for future events. Expert system rules can also read the clock time. The clock is updated by the time manager based on the event queue and is the same as the most imminent event(s).

Expert System

The expert system is actually comprised of multiple expert systems, referred to as a knowledge source. Each knowledge source contains a rule set, private working memory, and a suggestion buffer for communications to the evaluator. Working memory is used by a knowledge source to hold facts derived by its rules during operation. In addition, there is a public memory where knowledge sources can store and modify facts. This public memory may serve as a channel for coordinating communications between different knowledge sources.

Knowledge Sources. Knowledge sources are expert systems defined to perform over a sub-circuit or reduced domain. A knowledge source is defined using three parts. The first part describes a subcircuit on which the knowledge source contains knowledge. The second part is a rule base or rule set applying to the subcircuit pattern. The third part, the ignition statement, defines events or starting conditions which will activate a knowledge source and began processing of its rule base using the inference engine.

The knowledge source description is constructed using the simulation window interface and the parts are applied at different times during a simulation. The subcircuit description is described using a domain definition statement. Two types of domain definitions can be made, TREE and SET. TREE type domain definitions define a subcircuit with connections between the components. SET type domain definitions group a number of components which may or may not be connected into a set. Figure 3 illustrates a TREE type domain definition and its corresponding circuit. A later section will use the SET type domain definition in an example based on simultaneously switching bond pads. While the rule base and ignition statement information will be used during simulation, the subcircuit description will be used when a circuit is entered into the simulator to search for instances of this knowledge source. This searching is called knowledge source instantiation time.



```
;
(def-domain '(TREE (OR:ref_or1
                    (AND:ref_an1
                        a
                        (NOT:ref_n1 enable))
                    (AND:ref_an2
                        enable
                        b))))
```

Figure 3. Example of a TREE type domain definition

During knowledge source instantiation time, when an instance of a knowledge source is recognized, it is cataloged in the blackboard by creating entries for (1) a relative-to-absolute mapping, (2) a trigger, (3) a working memory area, (4) a suggestion area, and (5) a status word. The relative-to-absolute mapping is used during inference time to map all rule base references to parts, pins, and nets to their absolute items in the world data base. The trigger is a translation of the ignition statement to an executable LISP predicate which has had all relative references mapped to the world data base absolute entities. The working memory area, suggestion area, and status word are used during the inference operation on a knowledge source.

The ignition state is a boolean statement which references parts and nets in its domain to define the conditions which might require expert help from a knowledge source. For instance, if a knowledge source were defined to be activated every time a net *s* became high, the statement would look like "(becomes (net '*s*) 'high)".

Rule Base. The knowledge source rule base is constructed with if-then rules modeled after the CLIPS language (Culbert 1987). CLIPS is an expert system shell constructed by NASA and is written in the C language. Additional utilities were added to the rule language for accessing the event queue, the world data base, and the LISP language. These rules are used in a forward chaining inference engine. Suggestions to the evaluator result in event manipulation. Events associated with nets or knowledge sources can be changed, created, or destroyed.

Blackboard. The blackboard is a data base containing knowledge source information and information required for communications between the simulator and the expert system. Working memory for each knowledge source, knowledge source status flags, ignition state information, and resulting expert system suggestions are the types of information held in the blackboard.

Evaluator

Using current events specified by the time manager, the evaluator modifies the world data base and determines circuit behavior. Events reaching their scheduled time will either represent net transitions or re-ignition of knowledge sources. If the event is a net transition, and the transition will change the world data base, then the evaluator changes the net's value. Otherwise, net transitions which will not cause a change in the world data base are ignored. During a simulation, the evaluator is the only module which modifies the world data base. Events calling for re-ignition of a knowledge source do not require that the knowledge source's trigger be met. These types of events can be created by a knowledge source to continue monitoring circuit behavior after an initial ignition.

When a net transition changes the world data base, the evaluator determines which components and knowledge sources are affected. Components with changing input pin values have their behavior analyzed first. Next, knowledge sources becoming ignited, according to their triggers, are started. The evaluator determines that all knowledge sources are completed by examining the "Ready" section of the blackboard. If a knowledge source surmises a behavior different from the simulator, it stores suggested changes in the modification section of the blackboard. The evaluator uses these modifications to edit previously created events. When all events for a time step have been processed the evaluator passes behavioral results to the event queue manager as future events.

EXAMPLE KNOWLEDGE SOURCE

In this section an example of a knowledge source designed to detect simultaneously switching outputs of an integrated circuit is described. In advanced CMOS integrated circuits, switching multiple outputs of a circuit simultaneously can cause noise in the resulting output waveforms (TI 1987). For a short time the outputs should be considered unknown and thus unusable. The knowledge source in figure 4 and figure 5 is designed to detect simultaneous switching of three or more output pins in the fictitious device of figure 6.

Figure 4 describes the pattern domain and ignition statement for this knowledge source, called *switching-pads*. The domain definition fulfills two purposes; it defines a sub-circuit upon which the knowledge source has knowledge and assigns relative references to the sub-circuit. The domain definition uses the SET keyword which specifies that the set of objects following may or may not be connected. In this case, the set of objects contains four parts of type *outpad*. The ignition statement and rule base will reference these parts by the names *outpad1*, *outpad2*, *outpad3*, and *outpad4*. The knowledge source is defined to be triggered when any of the four parts *outpad1*, *outpad2*, *outpad3*, or *outpad4* have their second pin's signal go high.

```
; Define a knowledge source;
; pattern domain and ignition state.
(defks switching-pads
  (def-domain '(SET (part outpad outpad1)
                    (part outpad outpad2)
                    (part outpad outpad3)
                    (part outpad outpad4)))
  (def-ignition '(or (becomes (part-pin outpad1 2) 'high)
                     (becomes (part-pin outpad2 2) 'high)
                     (becomes (part-pin outpad3 2) 'high)
                     (becomes (part-pin outpad4 2) 'high)))
```

Figure 4. Start of a Knowledge Source description.

Figure 5 illustrates the rule base for the knowledge source *switching-pads*. There are six rules in the set. The first rule, called *start*, initializes our knowledge source. When any knowledge source is started, an initial fact, *initial-fact*, is placed in its working memory. The initial fact activates the first rule in this rule base which counts the number of simultaneously switching outputs using the "dolist" loop. A dolist takes each item in a list and binds it to a loop variable for one pass through the body of the loop. In this case, the loop variable is *?sig* and the values in the list are pin 2 of part *outpad1*, pin 2 of part *outpad2*, etc. The "dolist" also checks to see if any of the outputs are going high or will go high in the near future by using the "went" construct. As the first rule is counting, facts are created specifying which outpad parts are switching to high and an additional fact *counted* is created to fire future rules.

All facts generated by a rule are placed in working memory concurrently when the rule is completed firing. When the fact *counted* is place in working memory after rule one is completed, it will cause the second rule to fire. Rule two, called *determine-delays*, will determine if a simultaneous switching state exists and, if so, an appropriate delay factor is computed. Rules three through six are all similar; each of them determines if one of the four different outpads is switching and the appropriate actions to take. For rule three, if *outpad1* is switching, then an event is created to model noise generated by the switching. Following the period of noise the signal should stablize and change as planned. A second event is created to show the eventual transition of *outpad1's* second pin to high. A final action carried out by the rule is to note the behavior discovered by printing a message.

CONCLUSION

This paper has described a knowledge based simulator which combines expert system and digital-circuit simulator technologies. Using this simulator, a novice circuit designer can simulate circuit behavior phenomena which differs from the standard digital circuit behavior. These phenomena are not simulatable with standard digital circuit simulators and the conditions leading to the phenomena are only recognized by expert designers. The expert system part of the simulator recognizes conditions leading to phenomenal behavior, determines their consequences, and manipulates the simulator in order to emulate the phenomenal behavior.

```
;
;Count the number of pins switching and
;assert facts for each switching pad.
(defrule start
  (if (initial-fact))
  (then (bind ?count 0)
        (assert (counted))
        (dolist ?sig '((part-pin outpad1 2)
                       (part-pin outpad2 2)
                       (part-pin outpad3 2)
                       (part-pin outpad4 2))
            (if (went 'high ?sig :within 5 'ns)
                (bind ?count (+ ?count 1))
                (assert (?sig switching to high))))))
;
;Determine the delay factor due to simultaneous switching.
; For every pad switching above the count of 2 add
; a 10% delay to signal.
(defrule determine-delays
  (if (counted)
      (test (>= ?count 3)))
  (then (bind ?delay-factor (+ 1 (* (- ?count 2) .10)))
        (assert (modify events))))
;
;Take action:
; First create an event for period of noise, make signal
; unknown (x); then add delay to actual event.
(defrule check-switch-outpad1
  (if ((and (modify event)
            (part-pin outpad1 2) switching to high)))
  (then (create-event (part-pin outpad1 2)
                      :time (get-event-field (part-pin outpad1 2)
                                             :time)
                      :value 'X)
        (modify-event (part-pin outpad1 2)
                      :time  (* ?delay-factor
                                (get-event-field (part-pin outpad1 2)
                                                 :time)))
        (print-msg "Simultaneous switching behavior found..."
                   (part-pin outpad1 2))))
           :
           .
```

Figure 5. Knowledge source rule base.

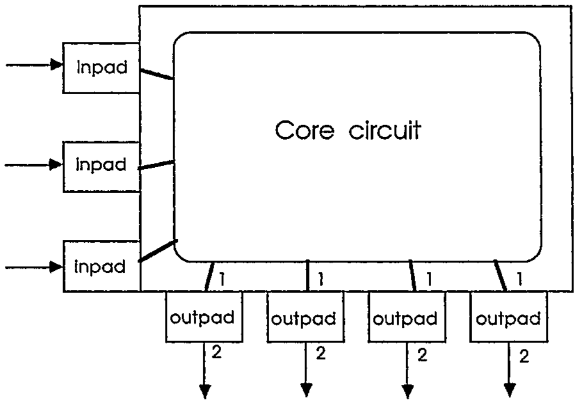


Figure 6. Output pad circuit

The knowledge source characterization of knowledge is attractive because it collects all information for one behavior into a single package. This lends itself to maintainability because rules are not dispersed through out a large unwieldy rule base. Also, the knowledge source package facilitates reusability because knowledge source can be grouped in libraries defined for specific circuit technologies where they are applicable. As a library grows, so does the power of the simulator.

Knowledge sources differ from the objects in an object-oriented system in the way knowledge is applied. Knowledge sources apply knowledge bottom-up, using pattern recognition. Objects apply knowledge top-down, using inheritance. An interesting enhancement to the prototype would be to modify components to be objects.

Currently, only slightly hard phenomenal behavior has been simulated with the SWIM prototype. Knowledge for the knowledge sources constructed was taken from design guides and textbooks aimed at circuit design. Translating this knowledge into rules and verifying its performance has been the most difficult facet of this work.

Future work on the prototype will center on behavioral knowledge verification, SWIM's interface for knowledge acquisition/manipulation, physical translation tools, and increasing the level of component model abstraction. The behavioral knowledge is the most important part of this knowledge based system because it directly influences its performance. It is impractical to manufacture a circuit to verify every iota of behavioral knowledge therefore, a more feasible alternative must be investigated.

The knowledge acquisition/manipulation interface of SWIM is tedious because it prompts a user for every portion of its knowledge base. After several sessions with the interface, a user becomes hindered by the interfaces inflexibility. More flexible interfaces will also be investigated. A graphic interface would probably be preferred for specifying a knowledge source's domain description. Presently, there is no automatic way to translate a circuit from its structural view to a physical representation. Entering this information by hand prohibits us from verifying larger circuits. Acquiring access to place and route software for creating a physical representation of a circuit is a solution being pursued. Place and route software facilitates the building of a floor-plan for laying out circuit components and attaching wires between them to realize the physical circuit.

Finally, increasing the level of model abstraction to a functional or behavior level is also being considered because system design is evolving toward these more abstract levels. This occurs because circuit design in now moving toward system design methodologies.

The SWIM prototype was successful in demonstrating the ability to use expert knowledge to simulate. Using SWIM, the knowledge source approach was used to simulate small real world circuits successfully. This demonstrates the utility of an artificial intelligence based approach to digital circuit simulation.

REFERENCES

Adelsberger, H.H., Pooch, U.W., Shannon, R.E., and Williams, G.N.(1986). Rule based object oriented simulation systems. In: Intelligent Simulation Environments (P.A. Luker and H.H. Adelsberger, eds.).*Simulation Series,* 17:1, Society for Computer Simulation, San Diego, California, 107-112.

Bloom, M.(1987). Mixed-mode simulators bridge the gap between analog and digital design. *COMPUTER DESIGN,* January 15, 51-65.

Borden, A. and Hugh, K.(1985). OPS5 as an electronic warfare design tool. In: *Artificial Intelligence and Simulation* (W.M. Holmes ed.). Society for Computer Simulation, San Diego, California, 71-75.

Culbert, C.(1987). *CLIPS Reference Manual.* NASA Johnson Space Flight Center [Published by Computer Software Management and Information Center (COSMIC), The University of Georgia, Athens, Georgia].

Dillinger, T.E.(1988). *VLSI Engineering.* Prentice-Hall, Englewood Cliffs, New Jersy.

Hayes-Roth, F., Waterman D.A., and Lenat D.B.(1983). *Building Expert Systems.* Addison-Wesley, Reading, Massachusetts.

Kim, G.T., and Zeigler B.P.(1988). The Class Kernal-Models in DEVS-Scheme: A Hypercube Architecture Example. *SIMULETER,* 19:2, 20-30.

Klahr, P., and Waterman, D.A.(1986). *Expert Systems: Techniques, Tools, and Applications.* Addison-Wesley, Reading, Massachusetts.

Lightner, M.R.(1987). Modeling and Simulation of VLSI Digital Systems. *Proceedings of the IEEE,* 75:6, 786-796.

Miczo, A.(1986). *Digital Logic Testing and Simulation.* John Wiley and Sons, Inc., New York.

Moser, J.G.(1986). Integration of artificial intelligence and simulation in a comprehensive decision-support system. *SIMULATION,* 47:6, 223-229.

Murray, K.J., and Sheppard S.V.(1988). Knowledge-based simulation model specification. *SIMULATION,* 50:3, 112-119.

Nagel, L.(1975). SPICE2: A Computer Program to Simulate Semiconductor Circuits. ERL Memo No. ERL-M520, University of California, Berkeley.

Reddy, Y.V., Fox, M.S., Husain, N., and McRoberts M.(1986). The Knowledge-Based Simulation System. *IEEE SOFTWARE,* 3:2, 26-37.

Terman C.J.(1987). Simulation Tools for VLSI. In: *VLSI CAD Tools and Applications* (W. Fichtner, and M. Morf, eds.). Kluwer Academic Publishers, Boston, Massachusetts.

Texas Instruments (1987). *Advanced CMOS Logic Designer's Handbook.* Texas Instruments Inc., Dallas, Texas.

Trimberger, S.M.(1987). *An Introduction to CAD for VLSI.* Kluwer Academic Publishers, Boston, Massachusetts.

Ulrich, E., and Suetsugu, I.(1986). Techniques for Logic and Fault Simulation. *VLSI Systems Design,* October.

AUTHOR'S BIOGRAPHIES

JOHN A. BENAVIDES is a graduate student in the Department of Computer Science at the University of North Texas. He received his BS in Electrical Engineering from the University of Texas at Austin in 1983. His research interests include simulation and VLSI design. He is currently a member of ACM, and IEEE.

Department of Computer Science
University of North Texas
Denton, Texas 76203
(817) 565-2767

DANA L. WYATT is an Assistant Professor of Computer Science at the University of North Texas. Her research interests include simulation, software engineering, databases, and distributed systems. She received her Ph.D. in 1986 from Texas A & M University where she worked on an NSF funded distributed simulation project. She is currently a member of ACM, IEEE, and SCS.

Department of Computer Science
University of North Texas
Denton, Texas 76203
(817) 565-2767