NOVA
A New Multi-Level Logic Simulator

L. Miles, P. Prins, K. Cameron
Microelectronics Research Center
University of Idaho
Moscow, Idaho 83843

J. Shovic
Advanced Hardware Architectures
Moscow, Idaho 83843

Abstract - **This paper describes a new logic simulator that was developed at the NASA Space Engineering Research Center for VLSI Design. The simulator is multi-level, being able to simulate from the switch level through the functional model level. NOVA is currently in the Beta test phase and has been used to simulate chips designed for the NASA Space Station and the Explorer missions. A new algorithm was devised to simulate bi-directional pass transistors and a preliminary version of the algorithm is presented in this paper. This paper also describes the usage of functional models in NOVA and presents performance figures.**

1 Introduction

Logic simulation is, arguably, one of the most important parts of the of the current VLSI design process. A fast, efficient, and accurate logic simulator can greatly enhance project turnaround time and personal productivity. The logic simulation system developed at the NASA Space Research Center for VLSI Systems Design located on the University of Idaho campus is designed to relieve the current logic simulation bottleneck. It is designed to be flexible and to meet the needs of VLSI designers for the near future.

NOVA is easily distinguished from other logic simulators currently available. First, it is fast; speed is gained by novel use of software data structures (which can be easily implemented in hardware) and by the development of a compiler (gibb) and flattener (glink) to take a hierarchical description of a circuit and compile it directly into a flat file for use by the simulator. NOVA also includes state models which incorporate most possible state values. It has been shown [2] that to completely describe a circuit with three distinct logic strengths, 21 states must be used; unlike other simulators, NOVA uses the required 21 states. This makes NOVA one of the most accurate simulators available. Third, NOVA is user-extendible. Not only can NOVA be made to interface with the most of the software that is currently available but even the internal primitives and functional models can be defined by the user to customize the simulator for the user's particular project. Lastly, NOVA is very versatile; it can simulate from the switch level through the large functional model level.

In this paper, an overview of the key architecture features of the simulator is presented in Section 2. A discussion of logic states and a preliminary version of a new bidirectional pass transistor simulation algorithm is presented in Section 3. User-defined Functional models are discussed in Section 4. A comparison of NOVA versus the performance of other simulators is presented in Section 5 and the superiority of NOVA in terms of speed for equivalent accuracy is shown. Future uses and enhancements to NOVA are discussed in the final section.

2 Overview

In this section, the set of programs written for the NOVA system will be discussed, the use of busses and bus notation will be explained and the true multi-level nature of NOVA will be shown.

2.1 Multi-level Simulation

NOVA is a true multi-level simulator. The simulation engine and the circuit description language are robust enough to allow the user great flexibility.

NOVA can be used as a switch level simulator. In fact, it can be used as a switch level simulator with user-defined timing delays. This allows the designer to describe the circuit in terms of transistors alone and simulate the entire system.

NOVA also has as its primitives, most of the common gate structures used. Delays on these gates may be allowed to default or may be set individually (rise time and fall time may be different) for maximum flexibility. The circuit description language allows modules to be declared which can then be thought of as user-defined gates.

Functional models are supported in NOVA and are described in Section 4. Briefly, one may define anything as a functional model whether it be a chip, a board, a gate, or any arbitrary set of transistors and gates.

Hence, NOVA is flexible and can simulate using any mixture of levels of abstraction of the chip under design.

2.2 The Program Set

The NOVA system is composed of two basic subsystems. One is the system dealing with the circuit description language and the other deals with the simulation engine itself.

2.2.1 Circuit Description in NOVA

A circuit is described in the BOLT language. This is a hierarchical language and has a modularity which is similar to Pascal-like languages [1]. A program named *gibb* compiles the BOLT file into an intermediate format calling upon *bpp* to preprocess bus notation into an expanded format, does the file inclusion, and performs string replacement. The intermediate format is then sent through *glink* which is the hierarchy primitive flattener. *glink* generates temporary files which are used by *greloc* to produce the flat file which is used by NOVA. A shell script is also created by *glink* if any functional models are present in the circuit description. This shell script directs the compilation of the user's 'C' code and links a program named *fmnova.*

2.2.2 The Simulator Programs

After successful circuit compilation, the simulator programs are used. The shell script *nova* is used to determine whether the program *fmnova* (circuit contains functional models) or the program *nfmnova* (no functional models) should be run [6]. Each of these programs runs the actual simulation.

2.3 Busses in NOVA

Special syntax allows a bus, a range of similarly named nodes, to be specified in a compact way in the BOLT language and in NOVA commands.

The syntax is fairly simple:

bus-name[index1:index2]

where index1 and index2 are non-negative numbers.

The bus is then expanded internally to:

bus-name[index1], ..., bus-name[index2]

As an example, a BOLT module for a 16 by 3 bit adder may be declared as:

```
sum[15:0] .adder16x3 a[15:0] b[15:0] c[15:0];
```

3 Logic STATES

The circuit node or wire may be divided into three types referred to as:

1. State information
2. Strength information
3. STATE (capitalized) information – both state and strength information collectively

State information is the the logical value of the node (wire). A binary system (0,1) requires three states to represent the logical state of a node:

1) 0
2) 1
3) X where X represents an unknown logic value, or a half level (neither 0 or 1).

Informally, the simulator requires the state information to calculate the output of the individual elements comprising the circuit. It is the State information regarding the inputs, of say, a NAND gate or an N-channel transistor, that is used to calculate what the output of the element would be if no other other circuit element were connected to it.

Strength information is used to resolve conflicts between multiple circuit elements that are connected to the same node. Thus, a classical logic simulator, which does not allow more than one circuit element capable of driving a logic state to be connected to a node, does not require any strength information at all. Modern logic circuits are rarely built solely out of the classical logic elements (NAND, NOR, INVERTER, AND, OR). Pass-transistor, tri-state, and pre-charged logic require a more sophisticated simulation scheme.

In general, different strengths may be assigned to the logic signals produced by a circuit element. If two or more elements are attached to a node, the node will assume the logic state with the greater strength. If a node has two circuit elements attached to it, both of the same strength but trying to output different logic values, then the node is evaluated to an X state.

3.1 Representing Indeterminate STATEs

STATEs representing either a 1 or a 0 must contain two pieces of strength information:

1. The greatest strength the signal may be driven to
2. The weakest strength it is possible for the signal to be

The first is significant because it sets a *lower bound* on the signal strength required to overdrive the signal. The weakest possible signal strength is important because it sets an *upper bound* on the strength of signal it can overdrive. In a case of contention between signals, the logic state goes to an X if neither signal can overdrive the other.

A logic system with the three basic strengths: Active (a), Resistive (r), and Floating (f) requires 12 distinct STATEs to describe the zeros and the ones in the circuit.
(Remember: **Active > Resistive > Floating.**)

0_{aa}	0_{rr}	0_{ff}	1_{aa}	1_{rr}	1_{ff}
0_{ar}	0_{rf}		1_{ar}	1_{rf}	
0_{af}			1_{af}		

Table 1: Logic STATEs Representing 0 or 1 in a Binary System With Three Basic Logic Strengths

In general if the system has N basic logic strengths represented by integers from 1 to N, these STATEs may be represented by the triplet, K_{bd} , where:

K	represents the actual logic state (1 or 0);
b	represents the strongest possible strength of the signal and ranges from: $1 <= b <= N$;
d	indicates the weakest strength and ranges from $1 <= d <= b$.

The indeterminate, or X logic STATEs also require two pieces of strength information:

1. the greatest strength of any circuit element connected to the node that may be attempting to drive it to a 0;
2. the greatest possible signal strength driving the node to a 1.

The first is necessary because it sets a *lower* bound on the strength of a signal capable of overdriving the node to a 1; and the second piece of strength information is used to set a *lower* bound on the strength of a signal that can overdrive the X to a 0.
The logic system with three basic strengths described above therefore needs 9 distinct logic STATEs to represent the indeterministic or half-level logic conditions that may occur on a node.

X_{aa}	X_{ra}	X_{fa}
X_{ar}	X_{rr}	X_{fr}
X_{af}	X_{rf}	X_{ff}

Table 2: Logic STATEs Representing Indeterministic Node Conditions in a System with Three Basic Logic Strengths

In general, if the system has N basic logic strengths that are represented by integers from 1 to N, these indeterministic STATEs may be represented by the triplet, X_{pq}, where:

X	indicates the actual logic state is unknown or at half level;
p	represents strongest possible strength of any signal driving the node to a logical 0;
q	represents strongest possible strength of any signal driving the node to a logical 1;

Both p and q range from 1 to N.

The minimum number of internal logic STATEs required to represent the information necessary to accurately simulate a binary system (0,1) with N basic logic strengths is seen to be:

$$\#STATEs = N(2N+1) \tag{1}$$

where: N^2 STATEs represent unknown logic values and (N^2+1) STATEs represent the (0,1) STATEs. For a system containing three basic strengths (active, resistive, floating) equation 1 gives a minimum of 21 STATEs required for proper simulation.

3.2 The Number of Basic Logic Strengths

It should be noted here that adding NIL strength STATEs corresponding to no connection (placed one step lower in the hierarchy of strengths than a floating strength), can simplify the formulation of the transistor models significantly, but is not, strictly speaking, necessary. The addition of a strength placed one level higher in the strength hierarchy than the floating strength can simplify the simulation of charge sharing effects and is appropriate when a large capacitance charge shares with a small capacitance. On the other hand, if depletion transistors, pullup resistors, and other overdrivable devices are not among the logic elements used in the circuit, then the resistive strength may be eliminated. The number of basic logic strengths may be extended to as many or as few as desired. (In the Table below, the alphanumeric representation is used for display only. The STATEs are represented internally as in the first column.)

3.3 Simulation of Bi-directional Pass Transistors

Transistors are not inherently uni-directional, rather, they are bi-directional. In all but the most unusual cases, the designer can easily determine the source node and can, therefore, use the uni-directional model. For those instances when a bi-directional model of a transistor is required, an algorithm to correctly simulate the action of those nodes was developed. In the development, a couple of assumptions were made. First, a designer will not have long chains (greater than 8) of bi-directional pass transistors; due to the distributed RC time constants, the circuit slows down considerably when the chain becomes larger than 4 to 8 [5]. Second, the designer will not use the bi-directional model if a uni-directional model will work as well; one must realize that a simulation of bi-directional transistors will slow down the simulation. The algorithm, then, can be a separate routine rather than a routine that is exercised every time the simulator is used.

3.3.1 Extensions to the basic States

As a signal propagates through a bi-directional pass transistor, its ability to overdrive another signal lessens, ie. the signal strength deteriorates slightly. To accommodate this within the framework of states that are already in place, the state strengths are given subscripts. The highest strength subscript is a 0 and the lowest strength is a 7. As an example of the use of the subscripts, let us suppose that a signal of state $1ar$ enters a chain of bi-directional pass transistors. After propagating through the first transistor, the state is $1a_0r_0$, after the second transistor, the state is $1a_1r_1$, and after the i^{th} transistor, the state is $1a_ir_i$.

3.3.2 Overdriving a State

One state may overdrive another state using the established rules of precedence as given in [2]. The substrengths 0 through 7 may be overdriven by any higher strength. This property is better shown by example:

- $1a_3r_3 + 1a_5r_5 \rightarrow 1a_3r_3$
- $0r_5r_5 + 1f_1f_1 \rightarrow 0r_5r_5$
- $1a_4a_4 + 0a_2a_2 \rightarrow Xa_2a_4$
- $Xr_3a_6 + 1a_7a_7 \rightarrow 1a_6a_6$

It is easily shown using the theory of Hamiltonian cycles, that the algorithm will converge to a proper state even with opposite ends of a chain being driven to different states.

One must be careful with regards to the timing algorithm. Through empirical study, it was decided to allow any transition to an X state happen in zero delay time but the transition from an X to a 1 or 0 state would happen in the normal delay time. Of course, a simple strength change within the same state happens with zero delay.

3.3.3 Interaction with Uni-directional Transistors and Gates

One can think of a group of bi-directional transistors as a cluster. A state comes into the cluster and is appended with the extra subscripts, ie. the state Spq becomes the state Sp_0q_0. When the state finally leaves the cluster, the extra subscripts are simply dropped and the state takes on the original format.

4 Functional Modeling in NOVA

In today's complex design arena, a complete functional and behavioral language is a necessity for a complete logic simulation tool. Functional models are useful for modeling large blocks of circuitry. Examples of some blocks that one might want to create a functional model for include micro-processors, co-processors, and any number of large circuits that one might want to interface with. These models are also used during a design for top-level simulation and design and exercising gate/transistor level models. The NOVA functional modeling language is a super-set of 'C'. The user may write programs using I/O, sockets and other LAN access, math, graphics, etc.; the communications with the simulation engine is done through a set of carefully defined routines. The functional model language in NOVA is based on a set of 'C' language function calls. These calls organize the interaction between the NOVA engine and the functional models into an efficient model and provide integrity of the internal NOVA data structures.

There are two major parts to a functional model within NOVA.

1. the BOLT language module description
2. the 'C' language functional description

For a functional model to interface with other modules that may be functional models or may be transistor or gate level models requires information to be specified and passed to a functional model and back to the NOVA simulation. Following is a discussion of the major design choices to be made and the major paths of information flow from the user to the functional model and to the NOVA simulation.

4.1 Evaluation

The major means of passing information into the functional model from the rest of the NOVA simulation is through the input pins to the module block. This information can be used as input data, clocking information, and the forcing function for evaluating a functional model. The issue of when to evaluate a functional model can be a difficult decision. One must consider the architecture of the block that is being modeled (i.e. if the model only needs to be evaluated on a rising or falling clock edge) and weigh this versus having any change in any input evaluate the model. As NOVA is a timing simulator and all inputs do not change at exactly the same time, having the functional model evaluate on all changes to all inputs will cause the simulation to proceed at a much slower pace than is necessary. For this reason the execution speed tradeoff between building a functional model and using the gate and/or transistor level BOLT model may not be as obvious as initially thought. A clocked synchronous model will typically execute much faster than an asynchronous block such as a RAM of comparable size.

4.2 Local Storage

If a user wishes to only have one instantiation of a given functional model, creating local storage is simple. The user merely needs to declare the variables that he wishes to remain between calls to the functional model as static within the 'C' program. If there is to be more than one call to the functional model, then the user may create local storage which is allocated by the NOVA simulator on a per instantiation basis. One can allocate variable amounts of storage by using "malloc" and storing the pointer to the allocated structure in the NOVA local storage.

4.3 Communicating with NOVA

All the communication from the functional model to the NOVA simulation is done strictly through use of Output pins from the module. As this is how information is passed in a gate/transistor model, the functional model must utilize the pins to send information to the NOVA simulation.

4.4 External Communication

The functional model user is free to do anything that the 'C' language is capable of doing. An example might be to initialize a RAM or ROM model by reading a file or to prompt the user for information through a terminal. More elaborate uses might be building a "Virtual Logic Analyzer" by connecting a functional model through a graphical interface (such as X windows) allowing the user to view the simulation in a human oriented manner as it progresses and allowing the user to add asynchronous events if desired. NOVA has been specifically designed to allow functional models the full use of the UNIX operating system and the I/O available.

4.5 Scheduling Other Events

One problem with using the functional model system as presently constructed is the limitation of NOVA being able to only schedule one event on a node at a time. When constructing a functional model, one often knows that certain events will happen in the future at a given time. This state information must be stored by the functional model which can lead to some awkward coding practices. A set of functional models constructed at Advanced Hardware Architectures have implemented a method for handling this in a non-obtrusive manner and will be discussed in a future paper.

4.6 Other Uses For Functional Models

A number of additional uses for functional models (as opposed to modeling blocks that will be present in the design) have been developed. These include:

- a module for comparing differences in incoming vectors and documenting the time and variation

- a module describing a block and running in parallel to the gate/transistor model of the block and feeding into a comparison block
- analysis of the incoming vectors and providing a human readable analysis of the vectors to a file
- synthesizing complex input patterns that require feedback from the simulation to the pattern. The tool used by Advanced Hardware Architectures for pattern generation, PATGEN, lends itself well to these applications as it also is a super-set of 'C'.

5 Performance Issues

To a designer, time is of the essence. One would like to be able to perform as many simulations as possible in a given time frame both to decrease development time and to increase confidence in the chip being designed. As has already been discussed, logic simulation proceeds in two steps. First, the circuit description is compiled, then the circuit is simulated. For comparison purposes, the results of identical simulations using three other simulators have been compared to NOVA. All four simulators were tested using the Quick Simulator Benchmark [3] and were run on the HP9000/370 [4]. The only major difference between the runs was that NOVA was tested on a machine with 16MB of memory and the other three were tested on a machine with 32MB.

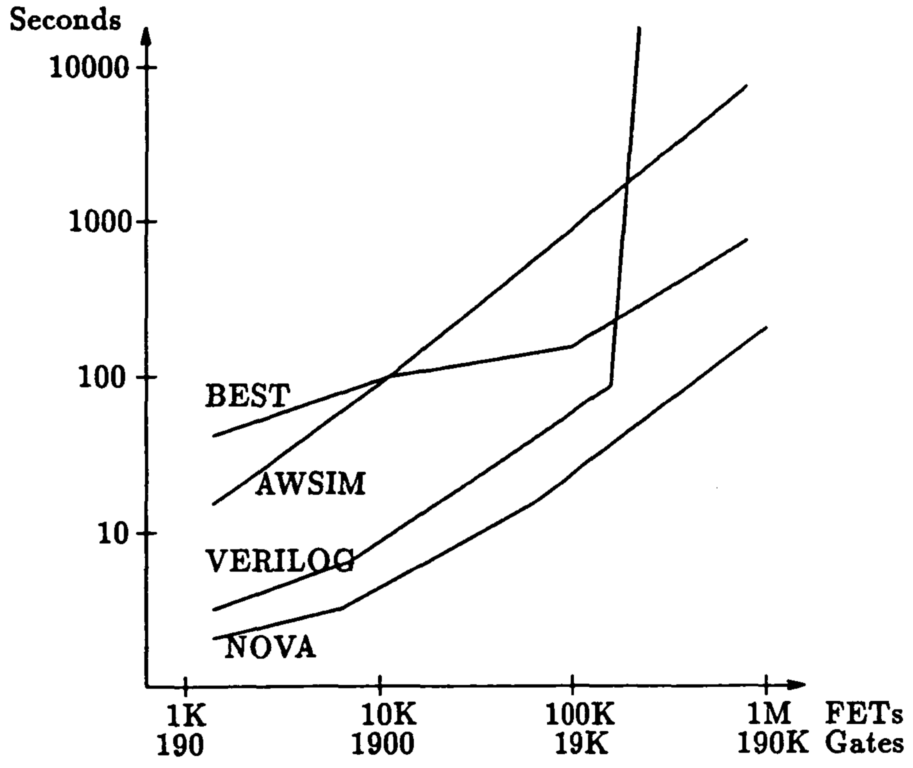

Figure 1: Compile Time Comparison

Figure 1 shows the compile time comparisons. As can be seen from the graph, NOVA (gibb) compiled circuits faster than AWSIM, BEST and VERILOG. It is interesting to note that VERILOG hit a physical memory limit at about 60K gates. This is because VERILOG uses about 400 bytes of memory per gate (whereas NOVA uses 32 bytes).

Simulation time is compared in Figure 2. The results are somewhat deceiving. None of the other simulators have as many features as NOVA so it is hard to compare. AWSIM is a zero delay simulator; therefore, it should be simulating much faster as propagation delay is not taken into account. BEST uses only 11 states and should be faster than NOVA but it isn't. VERILOG is faster than NOVA for the circuits that it can simulate. It should be noted that VERILOG cannot simulate as many gates as can NOVA due to memory limitations. VERILOG also cannot simulate at the transistor level. Neither BEST nor VERILOG can handle arbitrary delays; hence, they are tuned to run at a certain delay speed.

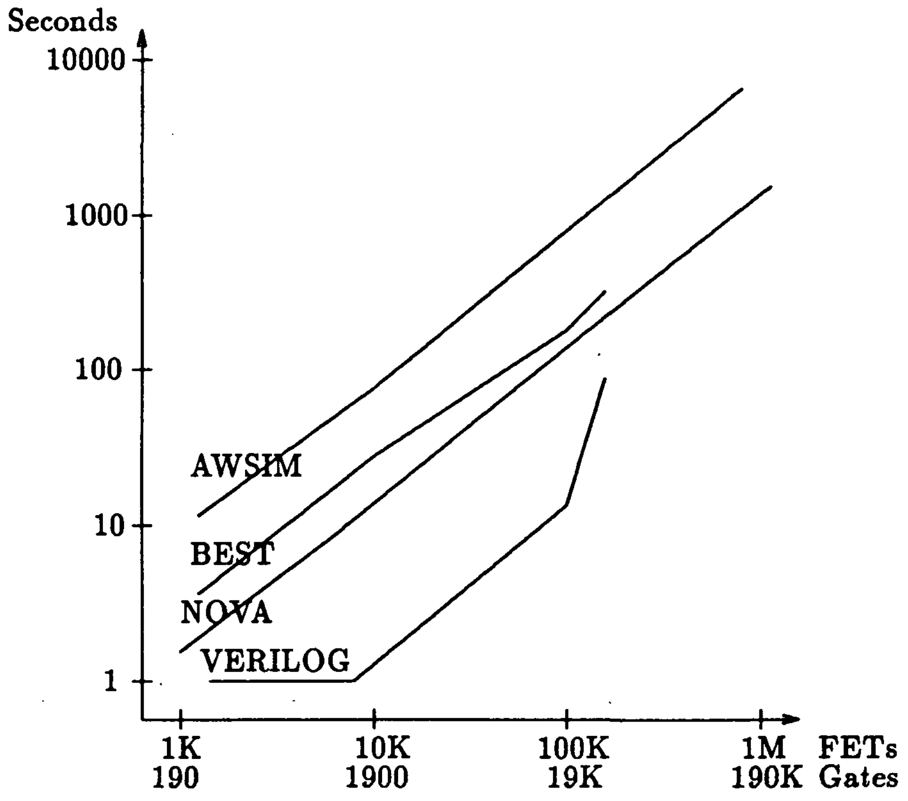

Figure 2: Simulation Time Comparison

Lastly, the relative performance (and portability) of NOVA is shown in Table 3. Using the HP9000/340 as a base for comparison, a suite was run to determine the relative simulation time. It is interesting to note that neither the size of the circuit nor the length of the simulation affected the relative simulation time appreciably. All four systems in the table are UNIX systems and it is the feeling of the ones porting the code, that NOVA could easily be made to run on any UNIX based machine. It might be noted that the DN10000

Type of System	Relative Speed
HP9000/340	1.0
HP9000/375	3.02
Apollo DN10000	5.08
Cray X-MP	6.56

Table 3: Relative Performance on Various Computers

is a two processor machine for which there is a very efficient 'C' compiler. The relatively low performance of the Cray X-MP can be partially explained because only about 6% of the code in the simulator could be vectorized. Much of the code is involved in pointer manipulation and in integer arithmetic which are the most inefficient operations on the Cray.

6 Future Uses and Enhancements to NOVA

Future enhancements to NOVA are currently focusing in two areas. One is the improvement in functional models and the other is coupling the I/O with the X-window system.

The industry seems to be standardizing upon two languages for describing behavioral and functional blocks. These languages are VHDL and VERILOG. A compiler can be designed that will translate these languages into the appropriate 'C' code and then compiled into the NOVA system for efficient execution and close tying of the functional blocks to the extant gate/transistor models. This allows a Top-down design with the same simulation tools being used throughout the design.

An X-window interface is currently being implemented and tested. The ability to see transitions graphically will greatly enhance NOVA's usefulness.

References

[1] *BOLT/gibb User's Manual,* Idaho Research Foundation, 1990.

[2] Cameron, K. B. and Shovic, J.C., "Calculating Minimum Logic State Requirements for Multi-Strength Multi-Value MOS Logic Simulators", *Proceedings of the 1987 IEEE International Conference on Computer Design: VLSI in Computers*, pp. 672-675; 1987.

[3] Greer, D.L., "The Quick Simulator Benchmark", *VLSI Systems Design,* pp. 40-46; November 1987.

[4] Heikes, C. and Koerner, C., "Logic Simulator Investigation Results," *Internal Hewlett-Packard Report*, 1989.

[5] Mead, C. and Conway, L. *Introduction to VLSI Systems,* Addison-Wesley, 1980.

[6] *NOVA User's Manual,* Idaho Research Foundation, 1990.