SNEL: A SWITCH-LEVEL SIMULATOR USING MULTIPLE LEVELS OF FUNCTIONAL ABSTRACTION †

D. T. Blaauw, R. B. Mueller-Thuns, D. G. Saab, P. Banerjee
Center for Reliable an High Performance Computing
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.

J. A. Abraham
Computer Engineering Research Center
University of Texas at Austin
Austin, TX 78758, U.S.A.

ABSTRACT:

Switch-level simulation has become a common means for accurate modeling of MOS circuit behavior. The SNEL simulator is a novel switch-level simulator which uses functional abstraction in a preprocessing step. Its functional abstraction algorithms use static circuit analysis to determine the overall circuit operation. This way, the operation of the circuit, rather than the full functionality of each individual circuit component is modeled during the simulation. A more abstract and high-level model of the circuit is therefore used, which greatly increases the simulation speed. Since the full switch-level behavior is captured by the functional abstraction, the accuracy of the simulation is maintained. The functional abstraction is performed at four circuit grain sizes or *levels*: individual circuit nodes, individual transistors, single dc-connected components, and multiple dc-connected components. At the highest level, the abstraction algorithm generates high-level software models for arbitrarily large circuit blocks. The proposed algorithms were implemented and tested for commercial circuits. By using all levels of abstraction, the simulation speed was increased by an order of magnitude.

1. Introduction

Switch-level simulation has become a widely accepted means for the logic verification of MOS circuits [1]. This popularity is largely due to the switch model's ability to accurately model MOS circuit phenomena, such as bidirectional signal flow, charge sharing, charge storage, the modeling of unknown signal values, and contention handling. In contrast to circuit simulation, the switch model is sufficiently simple to allow simulation of large designs with extensive test sequences. However, with the advance of circuit integration technology, the size and complexity of circuit designs has greatly increased. This has led to a new need for accelerated switch-level simulators.

In this paper, we present the SNEL simulator. SNEL is an event-driven switch-level simulator, based on the evaluation algorithms used in the SLS simulator [2]. The novelty of SNEL is its use of extensive circuit preprocessing to abstract the behavior of the circuit prior to simulation. This functional abstraction determines the set of switch-level phenomena occurring in individual or clusters of circuit elements. In practice, most switch-level phenomena, such as bidirectional signal flow and charge sharing occur relatively infrequently in a design. Functional abstraction determines which of these phenomena affect the circuit operation. This way, switch-level phenomena are only modeled where they are necessary for the accurate simulation of the circuit. Using functional abstraction, circuit evaluation is performed at a higher and more abstract level and is faster. Since the switch-level functionality of the circuit is captured in the abstraction, the simulation preserves its switch-level accuracy. Functional abstraction in the SNEL simulator is performed automatically and is completely transparent to the user. Since the abstraction algorithms use static analysis and are performed as a preprocessing step, their overhead is incurred only once for an entire simulation. Therefore, the functional abstraction overhead is easily offset by the resulting simulation speedup.

The remainder of this paper is organized as follows. Section 2 presents a definition and overview of functional abstraction. Section 3 surveys related work. Section 4 presents the developed functional abstraction algorithms. Section 5 contains the obtained performance results and offers concluding remarks.

2. Overview of Functional Abstraction

Each circuit element in a switch-level circuit description (transistor or circuit node) can display the full range of switch-level phenomena. For instance, all circuit nodes can potentially store a charge, and all transistors are potentially bidirectional. We call the possible operation of an element its *functional domain*. When a circuit element is considered as part of the overall circuit, however, its actual operation is only a subset of this functional domain, which we call its *functional application*. In an actual circuit, for instance, most transistors operate only unidirectionally.

The functional application of a circuit element depends both on its functional domain as well as the circuitry surrounding the circuit element. Functional abstraction is the process of determining the functional application of a circuit element from its functional domain. It must be performed such that the operation of the circuit as a whole is unaltered. Functional abstraction applies not only to individual circuit elements, but also to clusters of elements, such as dc-connect components and groups of dc-connected components. The functional domain of a group of circuit elements is the union of the functional domains of its components. Again, the functional application of a group of circuit elements is a subset of the functional domain of the group. For instance, the functional domain of the static gate implementation would include bidirectional signal propagation through each of its transistors, charge sharing effects between its nodes, and charge storage at each node. However, due to the complementary relationship of the pull-up and pull-down paths in the gate, most of these phenomena are eliminated. The functional application consists solely of its logic state, which is a Boolean function of the gate inputs, and its driving strength.

For accurate simulation, only the the functional application, rather than the entire functional domain must be modeled. The functional application is usually significantly smaller than the functional domain. Therefore, a large amount of unnecessary

† This research was supported in part by Semiconductor Research Corporation Contract 90-DP-142, and in part by Motorola, Inc. Austin, TX.

evaluation is avoided by obtaining the functional application before simulating the circuit and restricting the modeling to it. Since the behavior of circuit elements is data dependent, their exact functional application can only be determined dynamically during the simulation. However, by using static preprocessing algorithms, an approximation is obtained. This approximation is conservative, meaning it always includes too much functionality rather than too little. However, by extensively analyze the relationships of control signals and electrical circuit properties, they produce significant simplifications in the circuit model.

3. Related Work

Early work in functional abstraction for gate-level simulation is presented in [3]. Methods were developed to hand generate behavioral models from gate-level circuit descriptions. For MOS circuits, however, hand generation of functional models is both error prone and time consuming. Switch-level abstraction must be performed automatically. More recently, programs were developed to translate switch-level circuits into RTL models [4], into gate-level descriptions [5], or to verify switch-level circuits with a high-level description by proof of correctness [6]. In both cases, a high-level description is automatically abstracted from a switch-level description. However, the abstraction process makes many simplifying assumptions that compromise the switch-level model. Since the switch-level accuracy is not maintained, the used algorithms are not directly applicable to functional abstraction for switch-level simulation.

Algorithms to generate accurate evaluation code for dc-connected components were presented in [7, 8] and the COSMOS simulator [9]. They use so-called path tracing algorithms to translate all possible paths in a dc-connected component into a set of Boolean equations. Although the Boolean equations accurately evaluate the switch-level behavior, little functional abstraction is performed. The functionality of nodes and transistors is not analyzed and the full functional domain is modeled. Neither are CMOS logic gates detected, or is functional abstraction performed for multiple dc-connected components. These methods have the added disadvantage that their effectiveness is decreased for large dc-connected components. Barrel-shifters and other large dc-connected components are, therefore, difficult to model with these algorithms.

4. Abstraction Algorithms

To obtain significant simulation speedup, functional abstraction is performed at four grain sizes. The grain sizes or *levels* are, in sequence of increasing size: individual circuit nodes, individual transistors, dc-connected components, and clusters of dc-connected components. Information obtained from lower levels of abstraction is used at the higher levels. This way, the highest level of abstraction incorporates information obtained at all levels of abstraction. The presented algorithms are by no means exhaustive and can be easily extended. However, even with the most common simplifications in the circuit behavior, a significant increase in simulation speed was obtained. The individual levels of abstraction are presented in some detail below. The first two levels of abstraction have been presented in previous papers and are only treated briefly. For the two highest levels of abstraction, new research is presented.

4.1. Circuit Nodes: Abstraction of Temporary Nodes

Every node in a circuit has a finite parasitic capacitance and, therefore, carries a signal value from one evaluation to the next. This memory quality of a node greatly complicates the evaluation of the circuit. Each circuit node can potentially affect future evaluations and must be explicitly evaluated and stored. However, if the stored signal charge is immediately overridden by a different signal in the circuit, the stored signal is lost. If it is shown that this occurs for all possible circuit states, the retained signal is always destroyed and cannot affect the circuit operation. The memory quality of such a node is thus inconsequential to the circuit operation, and is removed from its functional application. The node is therefore classified as a *temporary* node. In [10], a new heuristic procedure for identification of temporary nodes is presented. The identification of temporary nodes is performed by tracing all possible paths from a node to permanent signal sources. Permanent signal sources are power nodes, and outputs of static gates, identified by automatic gate extraction. If it is shown for a node that there exists path to a permanent signal source for all possible circuit states, the node classifies as a temporary node.

4.2. Transistors: Abstraction of Unidirectional Transistors

The identification of unidirectional transistors reduces the complexity of the circuit model. However, the directional analysis algorithm can only restrict signal flow when it does not affect the circuit operation. A transistor is restricted to unidirectional signal flow for one of two reasons: either signal flow through the transistor in one direction does not occur at all or signal flow in that direction occurs, but does not affect the circuit operation. The directional analysis uses the electrical properties of the circuit such as the transistor and node sizes. These circuit properties are propagated along a graph representation of the circuit. This way, each transistor is aware of its surrounding circuitry. The graph is then examined locally at each transistor to detect those transistors that are unidirectional. For a more detailed description of the algorithm the reader is referred to [11].

4.3. DC-Connected Components: Abstraction of Static Gates

For most circuit designs, a large percentage of all circuitry consists of logic gate implementations. An algorithm was developed to detect static logic gate implementations in the switch-level description and substitutes them with so-called *gate descriptors*. A gate descriptor contains both the logic function and the strength information of the gate. Evaluation of the gate descriptor is straightforward and does not involve most switch-level phenomena. Therefore, the gate descriptor evaluates much faster than its corresponding transistor implementation, while the simulation maintains its switch-level accuracy.

A static CMOS gate is characterized by pull-up and pull-down functions that are duals of each others. This means that, for all possible gate inputs, there is always a conducting path from the gate output to either power or ground. Furthermore, for all possible true input values (logic 0 or logic 1), there is either a path to power or ground, but not simultaneously to both. Because of these characteristics, the functional application of a logic gate is greatly simplified. Below, the switch-level phenomena that occur in the functional domain of a static logic gate, but not in its functional application, are treated.

1 - **Charge Sharing and Charge Storage**
Charge sharing occurs when two or more nodes, isolated from power and ground, become connected through a conducting path. Since in logic gates there is always a conducting path from power or ground to the gate output, stored charge on the gate output is overridden. Therefore, neither charge sharing nor charge storage affect the gate output.

2 - **Bidirectional Signal Flow**
In a logic gate, only the new value of the gate output is determined. This gate output is always connected to power or ground through one or more paths. Along these paths, signals propagate only in the direction toward the gate output. Bidirectional signal flow is, therefore, eliminated.

3 - **Conflicting Signal Resolution**
Signal resolution is necessary when two or more signals with different logic states attempt to drive the same node. When true-values (logic 0 or logic 1) are applied to a logic gate, simultaneous paths to *vdd* and *gnd* do not occur. In this case, signal resolution is, therefore, not needed. In the case that unknowns are applied to the gate, simultaneous paths to *vdd* and *gnd* can occur. Both of these paths will, however, contain at least one transistor with an unknown signal controlling its gate. Since both paths contain a transistor in an unknown state, the output always evaluates to unknown. Resolution of conflicting signals is, therefore, not needed.

Modeling a logic gate is further simplified by the fact that its inputs all drive transistor gates. Only the logic state of the inputs is needed and their strength can be ignored. The only feature of switch-level simulation that remains in the functional application of a logic gate is the strength of the pull-up or pull-down path of the gate output. Gate extraction for switch-level simulation consists of three phases. The first phase obtains the logic pull-up and pull-down functions and determines whether they are duals of each other. The second phase analyzes the strength of the pull-up and pull-down paths for logically valid gates. The third phase generates the gate descriptors and evaluation routines.

4.4. Circuit Blocks: Generation of High-Level Models

For the highest level of abstraction, a model generator was developed to translate circuit blocks of multiple dc-connected components into high-level software models. The models are generated in executable C-code and fully capture the switch-level behavior of the circuit block. The SNEL simulator then links in these software models and executes them in an event-driven fashion. The partition size effectively represents the event size. Therefore, this abstraction level has the added advantage of being able to control and optimize the event size. The partition size is unrestrained and can range from a single dc-connected component to the entire circuit description.

Prior to model generation, directional analysis, gate extraction, and temporary node detection is performed. The logic behavior of a circuit block is modeled by the generator as a finite state machine derived directly from the circuit description. For each of the identified memory (state) nodes and output nodes, a set of next state evaluation statements is generated. Usually, one set of executions of these statements is sufficient to evaluated the circuit. However, in the presence of feedback loops internal to the circuit block, several such evaluations might be necessary before the circuit converges to a steady state. The model generator detects and breaks internal feedback loops and marks the breaking points as so-called *convergence nodes*. The generated model will then iterate on the evaluation statements until all convergence nodes attain a steady state.

The model generation is performed in three phases. In the first phase, the circuit block is partition such that there is no bidirectional signal flow between partitions. In the second phase, code is generated individually for each partition using a depth-first, post-order traversal of the partition elements. During the backward traversal of each circuit element, evaluation code for that element is generated. Chains of bidirectional transistors are handled in a manner similar to that in SLS [2]. Figure 1 shows an example of a simple partition. Transistors *T1* and *T2* are unidirectional, as indicated. Figure 2(a) shows the sequence of evaluation statements generated in the traversal of the circuit. As can be seen, little functional abstraction is obtained, since each circuit element is explicitly evaluated. Therefore, several optimizations steps are performed to obtain a higher level of functional abstraction.

If the state of the transistor gate is logic 0, signal flow cannot occur across the transistor. Therefore, the transistor and subsequent circuitry do not need to be evaluated. For instance, if the gate of *T1* in Figure 1 is logic 0, evaluation of circuit elements *T1*, *g1*, *g2*, and *g3* is not needed. Advantage of this is taken by simply inserting an 'if' statement before each unidirectional transistor as shown in Figure 2(b). The generated code is now further optimized by detecting tree structures consisting entirely of gates; for instance, the tree formed by *g1*, *g2*, and *g3* in Figure 1. The evaluation of individual gates in the sub-tree is replaced with a single table lookup function. The code in Figure 2(c) shows the generated code where this optimization is performed.

The last optimization detects simple inversion relationships between control signals, such as signals *i4* and *n4* in Figure 1. The inversion relationship is incorporated in the 'if' statement and eliminates the explicit evaluation of the inverter driving the control signal. Furthermore, the two 'if' statements generated in the code of Figure 2(c) are replaced with a single 'switch'

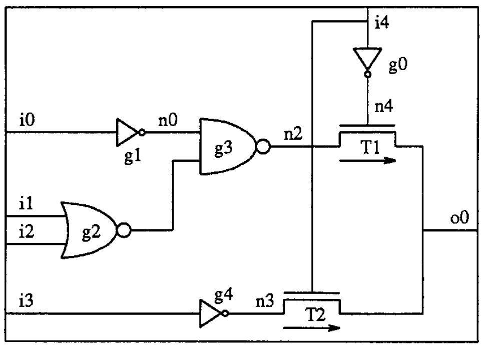

Figure 1. Circuit partition.

eval(*g* 0)	eval(*g* 0)	eval(*g* 0)	switch(stateOf(*i4*))
eval(*g* 1)	if (*n4*) {	if (*n4*) {	case 0: n2 = table()
eval(*g* 2)	eval (*g1*)	n2 = table()	eval(*T1*)
eval(*g* 3)	eval (*g2*)	eval(*T1*)	break
eval(*T* 1)	eval (*g3*)	}	case 1: eval (*g4*)
eval(*g* 4)	eval (*T1*)	if (*i4*) {	eval (*T2*)
eval(*T* 2)	}	eval (*g4*)	break
	if (*i4*) {	eval (*T2*)	case X: n2 = table()
	eval (*g4*)	}	eval(*T1*)
	eval (*T2*)		eval (*g4*)
	}		eval (*T2*)
(a)	(b)	(c)	(d)

Figure 2. Initial and optimized code generated for circuit in Figure 1.

statement, as shown in Figure 2(d). The evaluation of inverter *g0* is eliminated and the logic state of *i4* is only tested once using the switch statement. Furthermore, the code shown in Figure 2(d) effectively captures the 'decoder' behavior of the circuit, while accurately modeling the switch-level behavior of the circuit. After the optimization steps are performed, the element evaluation statements are replaced with code modeling the behavior of their elements. In the last phase, the partitions are rank-ordered. The evaluation code for the partitions is then concatenated according to their ranking to form a comprehensive model of the circuit block.

5. Performance Results and Conclusions

The SNEL simulator was implemented and tested on several types of circuits, including a large commercial microprocessor. For each circuit, the accuracy of the abstraction algorithms was verified. The circuits were simulated both with and without functional abstraction, and the resulting output signals were compared. For all test cases, the output signals were identical in logic state and strength.

Several functional abstraction statistics are shown in Table 1. Commercial circuits of various sizes and types were tested. The percentage of nodes that were temporary (*temp nodes*), transistors that were unidirectional (*unidir trans*), and transistors that were replaced with gate descriptors (*gate descr*) reflect the effectiveness of, respectively, the first, second, and third levels of abstraction.

All simulations were performed on a SUN SPARC work station. Several of the SLS benchmark circuits were run on SNEL to provide a reference for its simulation speed without functional abstraction. The normalized simulation speed [2], which is obtained by dividing the total simulation time by the number of cycles and the number of transistors in the circuit, was compared between SLS and SNEL for these benchmarks. When account was taken for the difference between the speed of the host machines, the simulation speed of SNEL, without the use of functional abstraction, was within a factor of two of SLS. The use of C++ and the lack of code optimizations in SNEL account for this difference. It shows that even without the use of functional abstraction, SNEL is an efficient simulator, comparable to SLS. Table 1 shows the speedup obtained from functional abstraction (*sim speedup*) which ranges from 4.85 to 15.3.

circuit		functional abstraction			sim
name	num of trans	temp nodes	unidir trans	gate descr	speedup
static latch	8	100%	100%	75.0%	4.85
exclusive-or	16	100%	100%	100%	5.32
alu func unit	20	100%	90.0%	60.0%	11.3
random logic1	28	70.5%	92.9%	42.9%	12.1
8 input mux	36	100%	88.9%	22.2%	12.3
random logic2	63	85.3%	96.8%	73.0%	12.9
bus control	82	82.2%	100%	60.9%	13.0
inc register	186	89.2%	100%	82.2%	15.3
register file	1,190	100%	100%	83.9%	9.6
shift register	1,228	97.0%	66.3%	38.1%	11.9
alu unit	2,422	78.8%	89.6%	55.4%	13.6
proc1	20,177	92.1%	81.3%	59.7%	11.3

Table 1. Simulation results of SNEL.

The circuit labeled *proc1* is a commercial microprocessor consisting of approximately 20,000 transistors. After functional abstraction is performed, only 18% of all transistors remain bidirectional, and 60% of all transistors are replaced with gate descriptors. The processor contains a 32-bit barrel shifter, for which functional abstraction has traditionally been difficult to perform. In light of this, the number of remaining bidirectional transistors is quite low. The processor was simulated with 10,000 test vectors.

In conclusion, a new switch-level simulator, called SNEL was presented. The SNEL simulator preprocesses the circuit description to abstract its functionality prior to simulation. Functional abstraction was concisely defined in terms of the functional domain and the functional application of circuit constructs. SNEL uses four algorithms that operate on levels ranging from single circuit elements to multiple dc-connected components. Since the functional abstraction preserves the complete functionality of the circuit, the accuracy of the simulation is maintained. However, SNEL models the circuit at a higher and more abstract level, which increases its simulation speed. The presented algorithms were implemented and tested on commercial designs. Without functional abstraction, the simulation speed of SNEL is competitive with current simulators. When functional abstraction was used, the simulation speed increased by more than an order of magnitude.

REFERENCES

[1] R.E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Trans. on Computers*, pp. 160-177, 1984.

[2] Z. Barzilai, D. K. Beece, L. M. Huisman, V. S. Iyengar, and G. M. Silberman, "SLS - A Fast Switch-Level Simulator," *IEEE Transactions on CAD*, pp. 838-849, 1988.

[3] S. A. Szygenda and A. A. Lekkos, "Integrated Techniques for Functional and Gate-Level Digital Logic Simulation," *Proc. IEEE International Design Automation Conference*, pp. 159-172, 1973.

[4] A. Brish, R. Keinan, and Y. Ravid, "A Smart System that Compiles RTL Models from Schematics," *VLSI System Design*, pp. 32-35, Feb. 1988.

[5] M. Boehner, "LOGEX - An Automatic Logic Extractor from Transistor to Gate Level for CMOS technology," *Proc. IEEE International Design Automation Conference*, pp. 517-522, 1988.

[6] V. E. Kelly and L. I. Steinberg, "The Critter System: Analyzing Digital Circuits by Propagating Behaviors and Specifications," *Proc. Conference on Artificial Intelligence*, pp. 284-289, 1982.

[7] I.N. Hajj and D.G. Saab, "Symbolic Logic Simulation of MOS Circuits," *Int. Symp. on Circuits and Systems*, pp. 246-249, 1983.

[8] G. Ditlow, W. Donath, and A. Ruehli, "Logic equations for MOSFET circuits," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 752-755, 1983.

[9] R.E. Bryant, D. Beatty, K. Brace, K. Cho, and T. Scheffler, "COSMOS: A Compiled Simulator for MOS Circuits," *Proc. IEEE Int. Design Automation Conference*, pp. 9-16, 1987.

[10] D. T. Blaauw, P. Banerjee, and J. A. Abraham, "Automatic Classification of Node Types in Switch-Level Descriptions," *Proc. IEEE International Conference on Computer Design*, 1990.

[11] D. T. Blaauw, D. G. Saab, J. Long, and J. A. Abraham, "Derivation of Signal Flow for Switch-Level Simulation," *Proc. European Design Automation Conference*, pp. 301-305, 1990.