AUTOMATIC CLASSIFICATION OF NODE TYPES IN SWITCH-LEVEL DESCRIPTIONS †

David T. Blaauw, P. Banerjee

Center for Reliable an High Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.

Jacob A. Abraham

Department of Electrical and
Computer Engineering
University of Texas at Austin
Austin, TX 78712, U.S.A.

ABSTRACT:

In switch-level simulation, nodes carry a charge on their parasitic capacitance from one evaluation to the next, which gives them a memory quality. A node is classified as temporary if its memory aspect is lost and cannot affect the circuit operation, whereas a node is classified as a memory node, if the memory of the node is maintained and can affect the circuit operation. Accurate classification of nodes into temporary and memory nodes increases the performance of compiled simulators and high-level model generators. We introduce a new approach for reliable automatic classification nodes in a switch-level description. Both an exhaustive, exponential-time algorithm and a polynomial-time heuristic are presented. The heuristic was implemented and tested for several large circuits, including a commercial microprocessor. For this processor, the proposed heuristics identified an average of 92% of all nodes as temporary nodes. The heuristic was applied in a high-level model generator and significantly increased its performance.

1. Introduction

In a switch-level description, the parasitic capacitance of a circuit connection is modeled by an implicit capacitor connecting the circuit node to ground. The so-called *node size* is an abstract measure of the size of this capacitance. Since every node in the circuit has a finite node size, every node carries a signal value from one evaluation to the next and has, therefore, a memory quality.

This memory quality of a node greatly complicates the evaluation of a circuit. Each circuit node can potentially affect the future evaluation and must be explicitly evaluated and stored. However, if the stored signal charge is immediately overridden by a different signal in the circuit, the retained signal is destroyed. Furthermore, if it can be 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 the node is called a *temporary* node. Conversely, a node which retains a signal value that can potentially affect the circuit operation is called a *memory* node. A simple example of a temporary node is the output of a static CMOS gate. The output of a static gate always has a driven strength and is only a function of the gate inputs. The retained output state is always overridden by the new gate output signal which is independent of its previous logic state. Therefore, the output state need not be saved between evaluations, and the node classifies as a temporary node.

Detection of temporary nodes can be used to increase the performance of compiled simulators [1, 2]. These programs use path tracing algorithms to generate evaluation code for dc-connected components in a circuit. For each node, a Boolean equation is produced to calculate the new node state given the states of the boundary nodes of the dc-connected component. Each node is thus explicitly evaluated. If, however, a node is identified as a temporary node, it is only important in how it affects the overall circuit evaluation and is not explicitly evaluated. By classifying the nodes as temporary and memory nodes, the evaluation can be restricted to only memory and boundary nodes. The storage requirement is also reduced with the identification of temporary nodes. A signal at a memory node must be saved from one evaluation to the next. A temporary node functions much like a temporary variable in a software function which is initialized at each execution. Since the temporary variable is re-initialized, its previous value is irrelevant and is not saved. Similarly, the state and strength of temporary nodes need not be saved between evaluations. If the percentage of temporary nodes in a circuit is high, a significant reduction in the simulation time and storage requirement can be achieved.

The concept of temporary and memory nodes has also been used in the generation of high-level models from switch-level descriptions [3, 4]. These methods extract from the circuit description a high-level model of the circuit behavior. Since temporary nodes affect only the current circuit evaluation, they are freely abstracted away and eliminated from the high-level model. However, memory nodes contribute memory to the circuit which must be incorporated in the generated model. Therefore, the high-level model generator handles memory nodes differently from temporary nodes to preserve the memory of a circuit in the abstraction process. Memory nodes thus complicate the model generation process and reduce the speed of the generated models. Accurate identification of temporary nodes, therefore, decreases the model generation time and increases the efficiency of the generated models.

The remainder of this paper is organized as follows. Section 2 discusses related work and outlines the approach used for identifying temporary nodes. Section 3 presents an exhaustive method for identifying temporary nodes, while Section 4

† This research was supported in part by Semiconductor Research Corporation Contract 89-DP-142, and in part by Motorola, Inc. Austin, TX.

describes an efficient heuristic based on the exhaustive method. Finally, Section 5 discusses the results obtained with the proposed methods and offers some concluding remarks.

2. Related Work

Currently, no systematic approach for automatic classification of nodes is known. One approach is to assume all nodes in the circuit to be temporary, unless they are manually flagged as memory nodes [3]. However, as the above discussion shows, all nodes are potentially memory nodes. Manual flagging easily overlooks a memory node which results in compromising the accuracy of the circuit simulation. A more appropriate assumption for switch-level descriptions is to assume that all nodes are memory nodes, unless they are specifically shown to be temporary nodes. This assumption might yield conservative results, but it guarantees the accuracy of the circuit evaluation. Furthermore, identification of temporary nodes should not rely on manual flagging, which is time consuming and error prone, but should be performed automatically.

A limited amount of node classification is performed in the COSMOS simulator [2]. The simulator eliminates the evaluation of certain nodes when not necessary for the overall circuit evaluation. The process of identifying nodes that are eliminated involves classification of nodes as memory and temporary nodes. However, the classification is embedded in and specific to the overall circuit analysis performed by the simulator. Therefore, it is difficult to determine how exhaustive the used method is and to use it independent from the simulator.

In this paper, we propose a new procedure for exhaustive identification of temporary nodes. Our approach uses a logic gate extractor to identify static logic gates in the circuit. The identification of temporary nodes is then performed by tracing all possible paths from a node to permanent signal sources. Permanent signal sources are power nodes, such as '*vdd*' and '*gnd*', and outputs of static gates. If it is shown for an examined node that there exists a path to a permanent signal source for all possible circuit states, the node classifies as a temporary node. Since the number of paths in a circuit grows exponentially with the size of the circuit, the exhaustive identification procedure has an exponential time-complexity. Therefore, we additionally propose a heuristic based on this exhaustive procedure. This heuristic is effective in identifying temporary nodes and is only of polynomial time-complexity with the number of nodes in the circuit.

3. Exhaustive Identification of Temporary Nodes

In a switch-level description, signal strengths form the following ordered set [5]:

$$\lambda < \kappa_1 < \kappa_2 < \cdots < \kappa_{max} < \gamma_1 < \gamma_2 < \cdots < \gamma_{max} < \omega$$

The strength κ_i refers to a charged signal strength from a node with node size i and is overruled by any signal with a stronger charge strength ($\kappa_{i+1}, \ldots, \kappa_{max}$) or with a driving (γ_i) or input (ω) strength. In turn, a driving strength γ_i is overruled by a signal with a stronger driving strength ($\gamma_{i+1}, \ldots, \gamma_{max}$) or with an input strength (ω). It can be seen that a retained signal on node n will have strength κ_i, where i is the nodes size of n. This retained signal is thus overruled by any signal of strength $\{\kappa_{i+1}, \ldots, \omega\}$; any signal with driving or input strength, or a charged signal from a node with a nodes size greater than i, will overrule the retained signal on n.

A temporary node is characterized by a retained signal that is always destroyed. Therefore, it must be shown that the retained signal on n is overruled by a stronger signal for all possible circuit states. To determine this, we introduce the function $F_{over}(m, n)$ which returns a boolean function of the control signals in the circuit. Node n is the node that is examined and node m is an adjacent node that is dc-connected to n through one transistor. The function $F_{over}(m, n)$ is defined such that if the condition $F_{over}(m, n) = 1$ is satisfied, node n is overruled by node m. If $G(m, n) = g$ is defined as the gate node of the transistor connecting m and n, $F_{over}()$ is expressed as follows:

$$F_{over}(m,n) = \begin{cases} G(m,n) & \text{if(} nodeSize(m) > nodeSize(n) \text{ // } m \text{ is a gate output // } m \text{ is an input or power node)} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

If $F_{over}(m, n)$ returns $G = g$, node m overrules node n if the condition g=1 is satisfied. In other words, if the transistor connecting n and m conducts, the state retained on node n is destroyed. On the other hand, if node m can not overrule node m, $F_{over}(m, n)$ returns 0 and the condition $F_{over}(m, n) = 1$ can not be satisfied.

Equation (1) examines only nodes immediately adjacent to node n. However, nodes further removed from n must also be considered in the classification of n. Equation (1) is therefore expanded to examine nodes a distance d removed from n. Function $F_{over,d}(m, n)$ is defined such that if the condition $F_{over,d}(m, n) = 1$ is satisfied, node n is overruled by a node $d-1$ transistors removed from m. If l_i is a node dc-connected to node m and $l_i \neq n$, $F_{over,d}()$ is described recursively as follows:

$$F_{over,1}(m,n) = \begin{cases} G(m,n) & \text{if(} nodeSize(m) > nodeSize(n) \text{ // } m \text{ is a gate output // } m \text{ is an input or power node)} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$$F_{over,d}(m,n) = \begin{cases} G(m,n) & \text{if(} nodeSize(m) > nodeSize(n) \text{ // } m \text{ is a gate output // } m \text{ is an input or power node)} \\ G(m,n) \cdot \sum_i (F_{over,d-1}(l_i, m)) & \text{otherwise} \end{cases} \quad (3)$$

$F_{over,d}(m, n)$ is thus a boolean function of the gate nodes of dc-connected transistors a distance $d-1$ removed from m. It should be noted that this resembles the justification problem encountered in test generation.

The boolean function generated by (3) is not fully complete since, no account is taken of the relationship between control signals of the circuit. The circuit is expressed as a mixture of gates and transistors and the gates nodes $G(n, m)$ are often related by a simple boolean function. However, in (3), all gate nodes are assumed to be unrelated. A function *Back*(g) is, therefore, defined to perform the conditional back-tracing step of test generation. If the gate g of the transistor connecting n and m is the output of a single logic gate, g is replaced with the boolean function of the gate inputs nodes. This process is recursively repeated, until no further back-tracing can be performed. If gates in a circuit are assumed to be only AND, OR, and INVERTER gates, the function Back(g) is expressed as follows:

$$Back(g) = \begin{cases} \sum_i (Back(k_i)) & \text{if(} g \text{ is output of an OR with inputs } k_i \text{)} \\ \prod_i (Back(k_i)) & \text{if(} g \text{ is output of an AND with inputs } k_i \text{)} \\ \overline{k_i} & \text{if(} g \text{ is output of an INV with input } k_i \text{)} \\ g & \text{otherwise} \end{cases}$$

If the control signal g is connected to multiple gate outputs or to the channel of one or more transistors, the relationship between g and other control signals cannot be expressed with a simple Boolean equation. In this case, the back-tracing of g is terminated. In equation (3), function $G(m,n)$ is now replaced by $Back(G(m,n))$.

The classification of node n is now determined by setting the distance d to the maximum non-cyclic path length in the circuit (d_{max}) and taking the sum S of (3) for all nodes m_i adjacent to n. If S simplifies such that $S = 1$, the retained state of node n is overruled independent of the circuit state. Node n is thus a temporary node. However, if S simplifies to something different than boolean 1, the retained state on node n is only overruled for circuit states that satisfy $S = 1$. In this case n is not overruled for all circuit states and n remains a memory node. Below, the function $Class(n)$ is shown, which determines the classification of node n.

$$Class(n) = \begin{cases} temporary & \text{if(} S = 1 \text{)}, S = \sum_i (F_{over,\, d_{max}}(m_i, n)) \\ memory & otherwise \end{cases} \quad (5)$$

4. Heuristic Method

Because (3) is recursive in nature, its time requirement is exponential with the number of nodes in a dc-connected component. Evaluation of the function $Class(n)$ is further complicated by the fact that, in order to show the identity $S = 1$ complete removal of the redundancies in the equation produced by $\sum_i (F_{over,\, d_{max}}(m_i, n))$ is required. This involves the tautology problem examined in [6] which is NP-complete in nature.

To avoid excessive analysis time, we proposed a heuristic with a polynomial-time requirement with the number of nodes in a dc-connected component. First, function $Back(g)$ is changed such that back-tracing is only performed for strings of inverters. This way the variable expansion is avoided, while simple inversion relationships between control signals are still detected. Secondly, we eliminate the recursive nature of (3) by setting $d_{max} = 1$. We now define $G(m_i, n)$ as the boolean equation describing the conductance of the transistor t_i connecting n and m_i.

$$G(m_i, n) = \begin{cases} 1 & \text{if(transistor } t_i \text{ is permanently on)} \\ 0 & \text{if(transistor } t_i \text{ is permanently off)} \\ Back(g_i) & \text{if(transistor } t_i \text{ is an N-type)} \\ \overline{Back(g_i)} & \text{if(transistor } t_i \text{ is a P-type)} \end{cases}$$

In order to take account of previously classified nodes, each node n is assigned an overruling strength, $Over(n)$, which indicates its minimum permanent overruling signal strength. If $Over(n) = \lambda$, n is a memory node. If $Over(n) > \lambda$, n is temporary node and has a overruling strength of $Over(n)$. Initially the overruling strength of all nodes is set to λ. Function $P(m,n)$ is defined as the overruling strength of n, if there is a path from m to n. If s_m and s_n are the node size of respectively m and n, $P(m,n)$ is defined as follows.

$$P(m,n) = \begin{cases} \gamma & \text{if(} m \text{ is a gate output // } m \text{ is a circuit input or power node)} \\ \kappa_{s_m} & \text{if(} \kappa_{s_m} > \kappa_{s_n} \text{)} \\ Over(m) & \text{if(} Over(m) > \kappa_{s_n} \text{)} \\ \lambda & otherwise \end{cases}$$

The function $Class(n)$ is now defined as follows:

$$Class(n) = \begin{cases} \min_i (P(m_i, n)) & \text{if(} S = 1 \text{)} \\ \lambda & otherwise \end{cases} \quad (6)$$

$$\text{where } S = \sum_i (G(m_i, n) \cdot (P(m_i, n) \neq \lambda))$$

It can be seen that S is a simple sum-of-product notation. Since $P(m_i, n) \neq \lambda$ is either boolean 1 or 0, the notation immediately reduces to a simple sum with terms from the set $\{n_i, \overline{n_i}, 0, 1\}$, where n_i denotes a node in the circuit and $\overline{n_i}$ denotes its complement. The simple nature of these terms allows exhaustive removal of redundancies in polynomial-time. The following set of simplifications is sufficient for complete reduction of the notation:

(1) $G_1 + G_2 = G_2$ if $G_1 = 0$

(2) $G_1 + G_2 = 1$ if $G_1 = 1$

(3) $G_1 + G_2 = 1$ if $G_1 = \overline{G_2}$

Simplifications (1) and (2) are performed with a linear examination of the terms, and simplification (3) is performed with an $O(n^2)$ search process. The overall simplification process has a polynomial-time requirement of the second order.

Setting $d_{max} = 1$ limits the procedure to examining only nodes that are immediately adjacent to node n. To expand the exchange of information, the function $Class()$ is executed repeatedly. Each iteration of function $Class()$ takes into account the overruling strength of nodes that are set temporary in previous iterations. An iteration of the function $Class()$ will set a certain number of nodes to temporary, which provides new information for subsequent iterations. When no additional nodes are set to temporary in an iteration, no new information is available for further iterations and the classification process is terminated.

In the worst case scenario, the number of needed iterations is equal to the number of transistors in the largest dc-connected component. Since the search process has a complexity of $O(n^2)$, the worst case scenario would require an overall computational time of $O(n^3)$, where n is the number of nodes in the circuit. However, in practice the number of iterations is small. For the tested circuits, which included a large barrel shifter, the maximum iteration count never exceeded 4.

The example shown in Figure 1 illustrates how repeated application of the function $Class()$ allows information to be exchanged across multiple transistors. The circuit consists of an inverter, implemented with transistors, feeding an input of a two input decoder. In the first iteration of function $Class()$ nodes $N0, N1, N2, N3, N5, N6, N7$ are immediately set to temporary with an overruling strength of γ, since they are gate outputs, power nodes, or circuit input nodes. For transistors $T1$ and $T2$ conditional back-tracing of the gate node is performed. $G(N4, N2)$

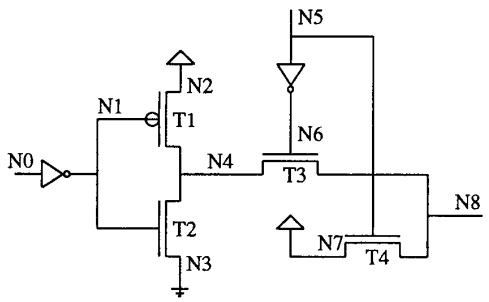

Figure 1. Exchange of information in a circuit.

evaluates to $\overline{N0}$, while $G(N4, N3)$ evaluates to $N0$. For node $N4$, S becomes '$N0+\overline{N0}$', which simplifies to boolean 1. The condition for a temporary node is satisfied and node $N4$ is classified as such. In the first iteration, equation S for node $N8$ is simply '$N5$'. However, in the second evaluation of *Class*() node $N4$ is set to temporary and S changes to '$N5+\overline{N5}$' for node $N8$. This simplifies to $S = 1$ and node $N8$ is thus also set to temporary. The two applications of *Class*() has propagated the effect of power nodes $N2$ and $N3$ to node $N8$, so that it is set to temporary. The third iteration of *Class*() produces no further temporary nodes and terminates the classification process.

5. Node Identification Results and Conclusion

The presented heuristic was implemented on Sun-4 workstations and tested on several circuit descriptions. Table 1 shows the results of the classification process. The percentage of nodes identified as temporary nodes ranges from 80% to 100% for the different types of circuit blocks. Table 1 also shows the average number of iterations needed to perform the classification process. The 8-input multiplexor shows the highest average number of iterations. This is caused by the long chains of pass-transistors used in this circuit. The maximum number of iterations needed in any of the tested circuits never exceeded 4. Due to the low average iteration count, the execution time of the heuristic was very low. For the tested circuits, the classification process required an order of magnitude less time than the parsing of the circuit. Since the classification process is only performed once in the simulation process, its added overhead is negligible compared to the overall simulation time.

The heuristic was used in the high-level model generator presented in [4]. The accurate identification of temporary nodes

circuit	number transistor	number nodes	nodes set temporary	number iterations
latch	8	14	100%	1.00
xor	16	27	100%	1.00
alu unit	20	20	100%	1.50
rand	28	25	80%	1.50
mux	36	38	100%	2.50
proc	20,920	8,337	92%	1.55

Table 1. Results of the classification process.

circuit			evaluation time (msec)		
name	trans.	nodes	without class.	with class.	speedup
latch	8	14	1,370	930	1.5
xorr	16	27	1,700	1,300	1.3
alu unit	20	20	1,410	720	2.0
rand	28	25	2,290	1,780	1.3
mux	36	38	5,820	3,000	1.9
proc	20,920	8,337	147,320	113,680	1.3

Table 2. Simulation with and without node classification.

proved vital to the generation of efficient high-level models. Performance of the generated models, with and without node classification, is shown in Table 2. The evaluation speed of the models increased between 1.3 and 2.0 times with the use of the node classification method.

In conclusion, an accurate and efficient method for identifying temporary nodes in a switch-level description has been presented. The identification process is based on showing that the stored charge on a node is overridden by a stronger charge source for all possible circuit states. The process uses a logic gate extractor as a preprocessing step. An exhaustive algorithm was presented which requires exponential evaluation time with the size of the circuit. Additionally, an effective heuristic with a polynomial time requirement was proposed. The heuristic was implemented and tested for several large circuits, including a commercial microprocessor. For these processors, the proposed heuristics identified 92% of all nodes as temporary nodes. With the node classification process, a significant increase in the performance of generated high-level models was obtained.

REFERENCES

[1] I.N. Hajj and D.G. Saab, "Symbolic Logic Simulation of MOS Circuits," *Proc. International Symposium on Circuits and Systems*, pp. 246-249, 1983.

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

[3] R. H. Lathrop, R. J. Hall, G. Duffy, K. M. Alexander, and R. S. Kirk, "Advances in Functional Abstraction from Structure," *Proc. 25th IEEE Design Automation Conference*, pp. 708-711, 1988.

[4] D. T. Blaauw, P. Banerjee, and J. A. Abraham, "SNEL: A Switch-Level Simulator Using Multiple Levels of Functional Abstraction," *Submitted: IEEE International Conference on Computer Aided Design*, 1990.

[5] R.E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Transactions on Computers*, vol. C-33, No.2, pp. 160-177, Feb. 1984.

[6] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Boston: Kluwer Academic Publishers, 1984.