A Dynamically-Directed Switch Model for MOS Logic Simulation

Dan Adler
Silicon Compiler Systems
Martinsville Rd, P.O. Box 16
Liberty Corner, New Jersey 07938

Abstract

A new model for MOS transistors suitable for logic simulation of VLSI circuits is presented based on the concept of a Dynamically Directed Switch (DDS). In this model, transistors are represented by directed edges in a graph, capable of changing their direction dynamically. A new distributed algorithm for switch-level simulation is presented based on an incremental graph algorithm where edge and vertex labels are updated as a consequence of circuit events. The result is a switch-level algorithm that runs at speeds approaching gate-level logic simulators, while dealing with all the features associated with switch-level simulation : bidirectional signal flow, ratioed logic, RC-tree timing, and correct handling of transistor signal propagation in the presence of unknown signals. The implementation of this algorithm in the **Lsim Mixed-Mode Analog and Digital Simulator** is described, and some results and examples are presented.

1 Introduction

The demand for high-speed switch-level simulators has led to the development of many different models and algorithms. Generally, most models are either *continuous* (e.g. the RSIM model [1]) or *discrete* (e.g. the MossimII model [2]) in representing voltages, resistances and currents. In this paper we focus our attention on discrete signal models which give rise to two classes of simulation algorithms: *global* and *distributed*. In global algorithms, a solution is achieved by tracing paths [2] in a graph representing part of the network, or by solving a set of boolean path equations [3]. Distributed algorithms, on the other hand, aim to achieve the same result by considering only a single component at a time, and its interaction with neighboring components.

Hayes has developed a model of distributed switch-level simulation based on a *lattice* of signal strengths [4], which is capable of handling *some* of the phenomena associated with MOS VLSI. However, the *global* properties of the iteration scheme used to actually derive the steady-state response of the circuit have not been analyzed by Hayes. Indeed, the algorithm converges to the *wrong* state in some fairly straightforward cases as pointed out by Bryant [5].

In another recently proposed distributed algorithm [6], events are processed locally in terms of whether they *decrease* or *increase* the path resistance from the source. In the latter case, *pseudo-events* are generated to allow new paths to become dominant. This method is more general than Hayes' approach and yields the correct results where Hayes' model fails. However, the concept of *pseudo-event* propagation and the rule-based nature of the algorithm make it very difficult to analyze its complexity and limitations.

In this paper, a new approach to distributed switch-level simulation is developed based on a new transistor model and a set of algorithms for incrementally updating node states. Using this approach, it is possible to explicitly formulate the global behavior of the simulation algorithm, and compare its results with other graph-based switch-level simulators.

2 Network Model

A discrete signal is defined as an ordered pair $<l,s>$ where l is the logic level and s is the discrete *strength*. The set of possible signal strengths

$$\Psi = \{z, c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n, S\}$$

which is totally ordered

$$z < c_1 < \ldots < c_n < d_1 < \ldots < d_n < S$$

is partitioned into four groups of signals representing *hi-impedance*, *charged*, *driven*, and *supply* strengths respectively.

Two factors directly affect the accuracy of the switch-level model in predicting the correct steady state of a digital MOS circuit: the number n of different conductance values used, and the algebra of signals defined over the set Ψ. In Bryant's model [2], n is restricted to a small integer value, and the algebra $(\Psi,+,\cdot,z,S)$ is chosen to be a *closed semiring* algebra, by interpreting the operators '+' and '$\cdot$' as the *maximum* and *minimum* operators over the elements of Ψ. However, the path algebra may be defined differently, by leaving '$\cdot$' as the *minimum* operator, while defining '+' as a true additive operator over transistor resistances, which produces a more accurate path resistance for transistors connected in series.

2.1 The DDS Transistor Model

The most commonly used transistor model for switch-level simulation is that of a bidirectional *switch*, with a series *resistance* or *conductance*. This model is the basis of a wide variety of algorithms, ranging from the *Thevenin* model used in RSIM [1], through Bryant's MossimII model [2], and taken to an extreme in Hayes' Connector-Switch-Attenuator model [4], where signals actually flow through a transistor in both directions at the same time.

Physically, the MOS transistor is never truly *bidirectional*. At any given time, the current I_{ds} through the channel is determined by the voltages V_{gs} and V_{ds}, and whenever I_{ds} is not zero, it flows in a *definite* direction. The *Dynamically Directed Switch* (DDS) model captures the transistor's ability to *dynamically* change the direction of current flow through the channel, according to the gate, drain and source conditions.

At any point in time, a MOS transistor circuit may be viewed as a set of mutually exclusive *channel graphs*. A channel graph G(V,E) is defined as a connectivity function between a set of vertices V, representing circuit nodes, and a set of edges E, representing on-transistors whose drain and source terminals lie within V. Each channel graph consists of a single connected component.

For two vertices $u, v \in V$, with strengths $s(u)$ and $s(v)$ respectively, the direction of the edge e_{uv} connecting them is defined as:

$$direction(e_{uv}) = \begin{cases} u \rightarrow v & \text{if } s(u) > s(v) \\ v \rightarrow u & \text{if } s(v) > s(u) \\ 0 & \text{if } s(u) = s(v) \end{cases}$$

This is analogous to defining the signal as flowing from the point of higher 'potential' to points of lower potential. When *both* nodes have the same strength, no direction is assigned, meaning that the edge cannot contribute to the strength calculation at either node. If all signal sources in the graph are of the same logic level, we may disregard such edges completely. However, if conflicting sources are present, undirected edges must transmit their logic levels in both directions. Thus undirected edges are ignored for strength calculation, and considered only for signal level calculation.

With each transistor we associate a resistance from the set

$$\Omega = \{r_1, r_2, \ldots, r_n\}$$

where resistances are ordered:

$$r_1 < r_2 < \ldots < r_n .$$

Note that the index n used here is the same as in the definition of Ψ. The transfer function of a transistor, which is used to calculate its contribution to the strength of its output node, is defined as:

$$\delta(s_k, r_j) = \begin{cases} s_{k-j} & \text{if } s_k \textit{ is driven/supply} \\ s_k & \textit{otherwise} \end{cases}$$

where we explicitly enforce $k - j \geq 1$ by choosing n sufficiently large (in Lsim $n = 4096$) to model all possible path resistances in a given technology.

A *directed path* of length k is defined as a sequence of edges $< e_{0,1}, e_{1,2}, \ldots, e_{k-1,k} >$, such that for $i = 0,1 \ldots k$ edge $e_{i-1,i}$ is an edge from vertex v_{i-1} to vertex v_i which satisfies:

1. $e_{i-1,i}$ is directed $v_{i-1} \rightarrow v_i$.

a *dominant directed path* is a directed path which also satisfies:

2. $\delta(s(v_{i-1}), r(e_{i-1,i})) = s(v_i))$.

An edge which is incident on vertex v and is on a dominant path to v is said to *dominate* v. A node may have more than one dominator, but all dominators must have the same strength. If only one edge dominates v it is called a *single dominator* of that node. The following proposition (which is given here without proof) forms the basis of our dynamic updating algorithms:

proposition 1: A channel-graph directed by the rules given in this section cannot contain any *directed* cycles.

Thus, the problem of determining the signal strength at each node can be restricted to dealing with *directed acyclic* graphs.

3 Shortest Path Algorithms

The problem of finding the steady state strength and level for each node in the circuit graph can be formulated as a shortest-path problem using the algebra defined in the previous section. Bryant's solution method [2] consists of applying a shortest path algorithm (which is a variation of Dijkstra's single-source algorithm using buckets [7]) three times to find the strongest charging path and the strongest discharging path to each node in the graph (the third pass is needed to deal with unknown signals). The steady state of each node is found by comparing the strengths of these paths. The disadvantage of this approach is that the network must be explicitly partitioned into channel-graphs for each event, and *all* node labels within each graph must be reevaluated. Although the worst-case time of this algorithm is linear in the number of edges in each graph, the actual run time is quite large due to the fact that each edge must be considered at least 7 times (1 for partitioning and at least 3×2 times for the shortest path calculations). The Algorithm is essentially *memoryless*, in that each new event is processed by creating new channel graphs to be solved.

We propose an alternative method which keeps track of *incremental* changes occurring in the graphs, and develop algorithms to dynamically update the graph labels. The incremental updating method, however, must be based on a distributed shortest-path algorithm.

As the basis for our approach, we choose a distributed version of Ford's algorithm [8], which is attributed to Moore [9]. We use Hayes notation #(node) to denote the operation of finding the steady state of a node as a function of all signals coming into that node through the channels of conducting transistors.

Algorithm SHORT

```
(0) short(u) {
(1)   foreach edge $e_{uv}$
(2)     if ($s(u) \geq s(v)$) {
(3)       if ($s(u) > s(v)$) direction($e_{uv}$) = $u \rightarrow v$;
(4)       calculate($\delta(s(u), r(e_{uv}))$);
(5)       $s_{new}$ = #($v$);
(6)       if ($s_{new} \neq s(v)$) {
(7)         $s(v) = s_{new}$;
(8)         que(v);
(9)       }
(a)     }
(b) }
```

Note that lines (4) and (5) in the algorithm, when performed incrementally, are analogous to the standard operation in shortest path algorithms:

$$s_i = minimum\,(s_i, s_j + d_{ij})$$

The algorithm is 'activated' by scheduling the source node for execution, and the call que(v) places node v on a FIFO queue for execution of this algorithm.

The worst-case complexity of this algorithm is $O(|V| \cdot |E|)$, however, the *levelized* implementation using FIFO ordering of events prevents the worst case from occurring in graphs arising from MOS circuits. Experimental results have shown this algorithm to be more efficient than Dijkstra's method for large classes of graphs. [10] [9].

In the following sections we present algorithms for maintaining and updating the labels of a graph using strictly local information. Three types of events may occur in switch-level channel graphs: edge addition and deletion corresponding to transistors turning on or off, and changes in drain/source strengths entering transistors which occur as a result of an edge event or a change in an external source. The latter type of events will be dealt with in the context of edge events.

4 Incrementally Adding an Edge

Let $G_1(V_1,E_1)$ and $G_2(V_2,E_2)$ be two graphs which have been directed and labeled by the SHORT algorithm. The graphs are either mutually exclusive or $G_1 = G_2$. Each vertex is labeled by $s(v)$ and each edge is directed and labeled with its resistance. The graphs, as we have claimed, posess no directed cycles.

Now assume that a transistor which was formerly off - turns on. This corresponds to the creation of a new edge e_{vu}. Let $v \in V_1$ and $u \in V_2$. The addition of the edge e_{vu} will result in a new graph G(V,E), whose vertices are V = $V_1 \cup V_2$, and whose edge set is E = $E_1 \cup E_2 \cup \{e_{vu}\}$. Since both graphs were assumed to have reached their steady state, the direction of the new edge e_{vu} is found by comparing $s(v)$ and $s(u)$ as defined earlier. Assume that the edge is found to be directed $v \rightarrow u$.

When the edge e_{vu} turns out to be a new single dominator of u, only the subgraph rooted at u node may have to be updated. This is accomplished by *queuing* the target node for execution of Algorithm SHORT, and allowing the shortest path calculation to proceed from there. The important point to note here is that once the direction of the edge e_{vu} is determined to be $v \rightarrow u$, the state of node v and all its predecessors cannot be changed by adding the new edge because there can be no directed feedback paths from $G_2(V_2,E_2)$ to $G_1(V_1,E_1)$ which affect the dominant path to v (this would result in a directed cycle). If the edge e_{vu} is found to be undirected ($s(v) = s(u)$), then the state of *both* nodes may change, causing both nodes to reconsider their state. This time the effects of the changes will only propagate through undirected edges or edges directed outward.

Figure 1 shows an example of incremental edge addition. When the top transistor turns on, its direction is found to be $n_1 \rightarrow n_2$ by comparing $s(n_1) = d_8$ with $s(n_2) = d_6$. Since the strength of node n_2 is not affected by this event - no further action is taken.

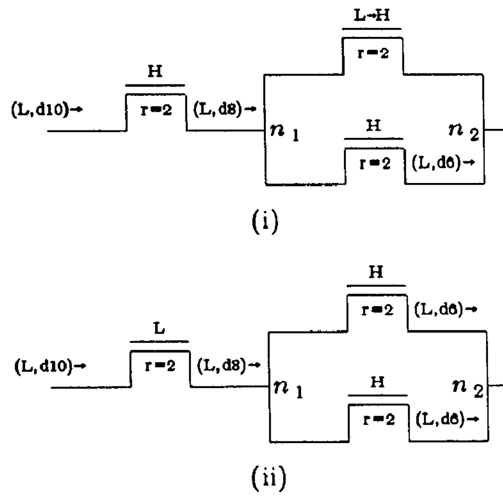

Figure 1: Transistor turning on

If the resistance of the switching transistor were $r = 1$ instead of 2, it would become a new single dominator of n_2, and all node strengths further down the path would be reevaluated by applying the SHORT algorithm from n_2.

5 Incrementally Deleting an Edge

Incremental edge deletion due to a transistor turning off is slightly more complicated because it may cause one signal path to be broken, and another path to override it at some other node. Furthermore, when a path is broken, all signals which are further down the path are no longer valid, and even their edge directions may no longer be valid. On the other hand, if an edge e_{uv} (directed $u \rightarrow v$) which is *not* a dominator of v is deleted, node v and all nodes further down the path will not be affected at all.

The problem then becomes, how to consistently keep track of the dominators of each node. This is solved by keeping, for each transistor output terminal, a marker indicating whether or not it is *dominating* the output node, and a dominator-count for each node. When a single dominator is removed from a node, its previous signal is invalidated and the node must find a new dominator. If none exists, the node will remain at its old logic level, with a charged strength proportional to its capacitance. All this is built into the **#(node)** operation mentioned earlier.

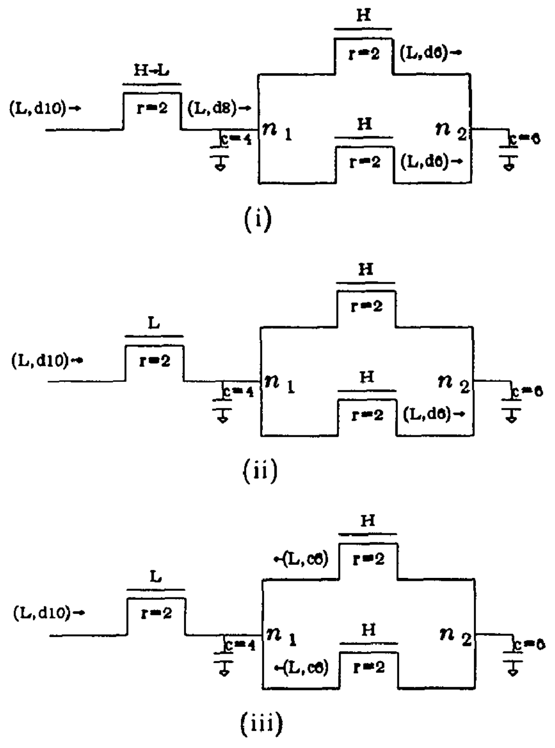

Figure 2: Transistor turning off

Figure 2 will be used to demonstrate the edge deletion algorithm. In (i), the steady state is shown before the transistor turns off. Node n_1 is driven to a state $<L,d_8>$ by a single dominator, and node n_2 is driven to $<L,d_6>$ by two dominators. When the single dominator of n_1 is deleted, its state becomes charged since no other transistors are directed into the node. Both outgoing transistors are then scheduled. Assume the top transistor runs first, it determines that it *was* a dominator of n_2 but not a single dominator, so it decrements the dominator count of n_2 by one and returns (ii). Now the other transistor runs, and this time it *is* a single dominator of n_2, so the signal $<L,d_6>$ is invalidated and n_2 settles to its charged state $<L,c_6>$ (since there are no other drivers present), which is then propagated back to n_1 through both transistors as shown in (iii).

Consider the effect of having a third transistor driving node n_2. If its strength were weaker, it would have a chance to become dominant when the previous single winner is removed. This transistor may have initially been directed outward from n_2, in which case it would run when n_2 becomes charged and determine that its direction is no longer valid - duplicating the case shown in Figure 2(i). If it were stronger, then the two parallel transistors would be pointing the other way, and the transistor event depicted would not entail any updates. If the second source were equal in strength to the driver of n_1, then the two parallel transistors would be undirected (having equal drain and source strengths) and the transistor event shown would only require two transistor updates and one node update.

6 Correctness and Complexity

Shortest-path updating algorithms like the one described here have been studied extensively in connection with routing problems in distributed packet-switching network. A proof of correctness of the updating procedure is given by Tajibnapis [11] for the case of equal edge costs. It is also shown there that the algorithm remains correct even if edge events occur while the effect of other events are still being updated, and that the algorithm behaves correctly on startup.

Jaffe and Moss [12] have shown that the updating procedure takes *linear* time if some order is imposed during the update procedure. Such an order is indeed imposed in our algorithm by the fact that DDS transistors are only considered by the node they are currently driving.

7 Unknown Handling

Proper handling of unknown signals at the gates of transistors is necessary for correct simulation of MOS VLSI. The over-pessimistic approach of extended gate-level simulators [5], where an unknown signal at the gate of a transistor always drives its output to unknown, makes it impossible to correctly simulate many important structures such as NOR gates, PLAs etc. in the presence of unknown signals.

Bryant's solution [2] is to distinguish between *definite* paths, which only pass through on-transistors, and *indefinite* paths, where some of the transistors on the path may have an unknown gate state. Indefinite paths can be *blocked* by stronger definite paths, or by definite paths of the same strength which carry the same signal level (high or low). The algorithms presented so far can be extended to deal with unknown signals in a similar way based on the following two propositions which are given here without proof.

proposition 2: The SHORT algorithm can be extended to find the shortest path from a set of m (>1) sources $\{s_0, \dots s_{m-1}\}$ to all other nodes simultaneously. m can be limited to two by combining all ***high*** sources and all *low* sources [9].

proposition 3: The signals propagated through the network can be extended to consist of a ***definite*** component and two ***indefinite*** components.

The propagation of generalized ***signals*** can be shown to be identical to propagating ***strengths*** and then comparing them as in Bryant's approach, with the added benefit that the strength comparison becomes part of the signal propagation action.

8 Timing Simulation

The *resistance* of the strongest dominant path from all signal sources is inherently embedded in the signal strength at each node (this follows from the definition of the δ function). If the range of path resistance is large enough, an RC delay constant for each node can be formulated using this quantity. The time constant corresponds to an RC-tree with no side-branch capacitances, and produces less accurate results for intermediate nodes [13].

This approximation, coupled with the algorithm's **que** operation, forms the basis of a switch-level timing simulation mode in the **Lsim**™ **Mixed-mode Digital and Analog Simulator** [14]. Since Lsim also contains a circuit-level Timing Simulator called ADEPT [15] as well as a SPICE-level analog simulator, the switch-level algorithm is expected to produce only crude first-order delay approximations, where the emphasis is more on maximizing speed and minimizing memory requirements than on detailed waveform accuracy.

9 Mixed-Mode Simulation

The **Lsim Mixed-Mode Analog and Digital Simulator** consists of an extensible set of simulation algorithms, all sharing a common infrastructure. All algorithms (except for the SPICE-level analog simulator) are based on ***cellular*** component models that interact with node signal ***arbitrators*** which determine the least common level at which the component models can communicate.

The Dynamically Directed Switch model described in this paper has the advantage of easily and directly interfacing to other logic level components in the circuit, since at any given time the transistor model appears to be a ***unidirectional*** device to the node it is driving. Thus, the same node may be driven by a *behavioral* model, a *gate-level* model and DDS models simultaneously and correctly determine its resulting state.

Mixing DDS level models with ADEPT components is achieved by a state-variable mapping from level-strength pairs to current-conductance pairs using Thevenin/Norton transformations. Finally, any transistor level subcircuit can be toggled on the fly between ***switch*** and ***circuit*** simulation modes, making it possible to simulate critical paths in context, run analog subcircuits (e.g. sense amps, bootstrap drivers) as part of a large logic simulation, and use the switch-level algorithm to initialize a circuit for more detailed circuit simulation.

10 Performance

The DDS model and algorithm have been used to simulate a large variety of circuits. Cells and small blocks are typically simulated interactively as they are being designed, while full VLSI chips with hundreds of thousands of transistors can be simulated in batch mode for many clock cycles while the results are automatically verified against those of a high-level behavioral model.

Circuit	Trans	Cycles	CPU (min)	Events/Sec
static alu	1,100	90	0.8	5,530
addr data-path	2,200	30	1.1	4,860
i/o data-path	6,100	15	1.2	4,100
multiplier	12,000	45	9.2	6,400
mixed chip	16,200	200	12.1	5,840
micro store	48,700	13	4.4	5,180

Table 1: Lsim DDS Simulation Benchmarks

The table above shows simulation statistics for a number of small to medium sized circuits. All benchmarks were performed on a SUN 3/260 with 32 MByte of main memory. The *mixed chip* entry represents an actual chip where only two blocks (including the data path) were simulated in transistor mode with the other blocks simulated as behavioral models. The *Events/Sec* metric measures the number of DDS transistor evaluations per CPU second.

When compared to global algorithms such as MossimII, it is possible to come up with examples where the algorithm will not provide any improvement over the ***memoryless*** approach. However for a typical channel graph and a typical mix of transistor events, the total number of label-updates is found to be significantly smaller using the algorithms presented here than using the global approach.

As an example of the efficiency of incremental updating in MOS circuits, consider the CMOS parity-generator circuit in Figure 3 which is made up of dynamic XOR gates with shared terms. The simulation results for this circuit are shown in Figure 4 using the multi-mode capabilities of Lsim. First, the circuit is run in ADEPT mode for several cycles, and then all transistors are toggled into DDS mode. Note

™ Lsim is a trademark of Silicon Compiler Systems the accuracy of the delay calculations in DDS mode in comparison to the ADEPT waveforms. During the evaluation phase, the signal *clk* goes high causing two n-channel transistors to turn on and two p-channels to turn off. When the n-channel transistors turn on, a path to ground is created either to *ODD_* or to *EVEN_*. Updating this path requires only one operation, because the signal is already 'waiting' at the input of the enabling transistor.

11 Acknowledgements

I would like to thank Carl Christensen, Peter Odryna, Kevin Nazareth, Mike Purnell, and Mehmet Cirit whose contributions to the creation of the Lsim Mixed-Mode simulation environment far exceed my own.

References

1. C. Terman , *Simulation Tools for Digital LSI Design* , Ph.D. Diss., VLSI Memo no. 83-154, Mass. Inst. of Tech. (September 1983).
2. R. Bryant , "A Switch Level Model and Simulator for MOS Digital Systems," *IEEE Transactions on Computers* c-33(February 1984).
3. R. Bryant, 'Boolean Analysis of MOS Circuits ," *IEEE Transactions on CAD* **CAD-6** (July 1987).
4. J. Hayes , 'Pseudo-Boolean Logic Circuits," *IEEE Transactions on Computers* c-35(July 1986).
5. R. Bryant , "A survey of Switch Level Algorithms ," *IEEE Design & Test* , (August 1987).
6. R. Sunblad and C. Svensson , 'Fully Dynamic Switch Level Simulation of CMOS Circuits," *IEEE Transactions on CAD* CAD-6(March 1987).
7. E. Denardo and B. Fox , 'Shortest-Route Methods: Reaching, Pruning, and Buckets," *Operations Research* **27**(Jan 1979).
8. S. Even , *Graph Algorithms* , Computer Science Press (1979).
9. U. Pape, 'Implementation & Efficiency of Moore-Algorithms for the Shortest Route Problem ," *Mathematical Programming* **7** pp. 212-222 (1974).
10. R. Dial , F. Glover, D. Karney, and D. Klingman, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," *Networks* **9** pp. 215-248 (1979).
11. W. Tajibnapis, "A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Coputer Network," *Communications of the ACM* **20** pp. 477-485 (July 1977).
12. J. Jaffe and F. Moss, "A Responsive Distributed Routing Algorithm for Computer Networks," *IEEE Transactions on Communications* Vol. COM-30 pp. 1758-1762 (July 1982).
13. P. Penfield, Jr. and J. Rubinstein, 'Signal Delay in RC Tree Networks," pp. 613-617 in *Proceedings of the 18th Design Automation Conference*, (1981).
14. P. Odryna, K. Nazareth, and C. Christensen, "A Workstation-Based Mixed Mode Circuit Simulator," in *Proceedings of the 23rd Design Automation Conference*, (1986).
15. P. Odryna and S. Nassif , 'The ADEPT Timing Simulation Algorithm," *VLSI Systems design*, (March 1986).

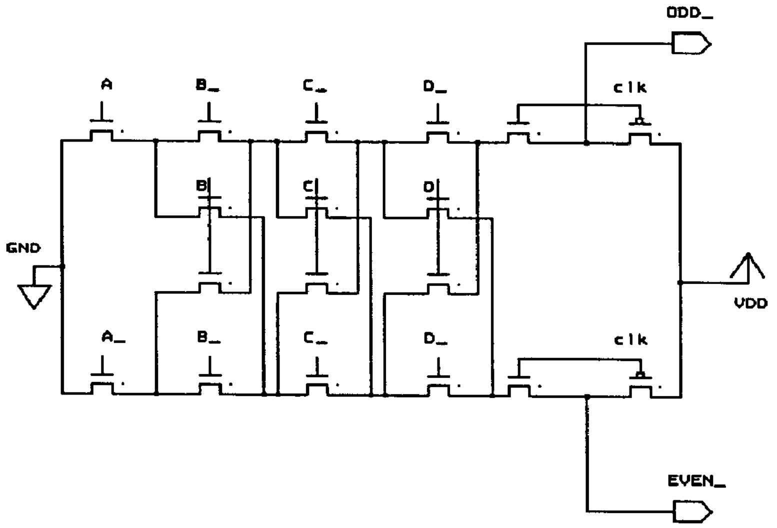

Figure 3: Parity Generator Circuit.

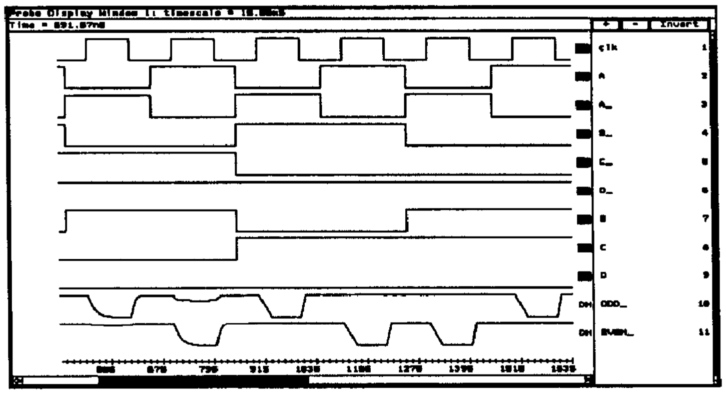

Figure 4: Multi-Mode Simulation Results.