ZERO DELAY VERSUS POSITIVE DELAY
IN AN INCREMENTAL SWITCH-LEVEL SIMULATOR
Larry G. Jones
Department of Computer Science
University of Illinois at Urbana-Champaign

ABSTRACT
We present the methods used in the implementation of an incremental zero/integer-delay switch-level logic simulator for MOS circuits based on the MOSSIM II switch-level model. Zero-delay timing reduces spurious reevaluations caused by minor changes to signal timing that do not affect logic, while integer-delay timing provides an ability to model race conditions that do affect the logic. In experiments run on switch-level versions of the ISCAS combinational and sequential benchmarks, incremental switch-level simulation with mixed zero/integer delay was 4 times faster (on average) than incremental switch-level simulation with only positive integer delays.

1. INTRODUCTION
As MOS feature sizes continue to shrink and the complexity of circuits integrated on a single silicon chip grows, simulation becomes an ever more important step in the design process. After the initial design phase, designs are often subjected to small changes in an effort to fix design flaws or tune performance. The design then must be resimulated to check the success of the changes and to ensure that no new problems have been introduced. Many of the results of the computations performed in resimulating a modified design are identical to the results obtained during previous runs of the simulator. By reusing previous results incremental simulators often complete their task in a fraction of the time required by conventional simulation techniques [9] [7] [5] [11].

Previous incremental simulators have modeled signal delay as strictly positive. This limited the usefulness of the simulators since even slight modifications to the circuit (such as transistor sizing for performance) may cause all directly and indirectly dependent signals to experience a time-shift requiring massive resimulation, even when neither the signal values nor their relative arrival order has changed [11]. The incremental switch-level simulator we present (ICESIM) uses a delay model which allows the mixing of zero-delay with positive integer-delay elements. Zero-delay timing provides an abstraction that can be used for modeling digital circuits where signal delay does not affect logic. Positive integer-delay provides a capability for modeling race conditions that affect the logic of the circuit. The logical behavior of most MOS digital circuits can be modeled using a combination of both delay techniques.

It is well known that zero-delay simulation often reduces simulation overhead by reducing the number of events requiring scheduling and by reducing scheduling costs [13]. With respect to incremental simulation, zero-delay also reduces the simulator sensitivity to time-shifts. The result is that incremental simulation with zero-delay is often much faster than incremental simulation with only positive delays.

ICESIM operates on a flattened representation of the circuit graph but provides a high-level hierarchical interface by being tightly coupled to a schematic capture system (ICE) using an incremental netlist compiler[10]. Manipulations of the design at any level in the structural design hierarchy are quickly and automatically mapped into manipulations of the flat netlist by the compiler. Other differences which distinguish this system from previous incremental simulators are its simulation model (based on the MOSSIM II switch-level model [8]), and the mixing of static and dynamic subnetwork partitioning techniques to reduce overhead.

2. EVENT HISTORY
Before the incremental simulator can be used, the design must have been simulated at least once using the nonincremental simulator. During this initial simulation the nonincremental simulator produces a history of simulation events which, for each node, records each change in state and the time the event occurred. Subsequent incremental simulations update the event history to be consistent with any changes that have occurred in the logic since the previous (nonincremental or incremental) simulation. The incremental simulator does not make use of the state of every node for every moment in time. Instead, the states of nodes are extracted from the event history on demand as they are required to determine and evaluate nodes affected by the modification.

3. TIMING
The simulator models time using a zero/integer-delay timing model that assumes each node \( z \) has an integer transport delay \( D(z) \geq 0 \) associated with it. Node \( z \) reaches its steady-state \( D(z) \) units of time after all inputs to the
incremental subnetwork containing \( z \) become stable. The inclusion of zero-delay reduces the simulators sensitivity to feedback paths it is channel connected subnetwork containing \( z \).}

Further, when the circuit contains feedback paths it is necessary to introduce positive delays on specific nodes to remove zero-delay cycles (which have ill-defined semantics [6]).

**4. INCREMENTAL STATE EQUATION**

Let \( S_i(x,t) \) represent the state of node \( x \) at time \( t \) during simulation \( i \). Let \( D_i(x) \) represent the delay assigned to node \( x \) during simulation \( i \). For a set of nodes, \( X \), let \( S_i(X,t) \) represent the states of the nodes in \( X \) at time \( t \) during simulation \( i \). In nonincremental switch-level simulation the state of \( z \) at time \( t + D_i(z) \) during simulation \( i \) is given by

\[
S_i(z,t+D_i(z)) = F_i(S_i(A,t))
\]

(1),

where \( A \) is the set of nodes containing all nodes in the subnetwork containing \( z \), as well as all nodes driving the gates of transistors in the subnetwork containing \( z \). The function, \( F_i \), indicates the result, with respect to \( z \), of applying the evaluation procedure to the subnetwork containing \( z \). We refer to the nodes in \( A \) as the arguments of \( z \) and the subnetwork containing \( z \). We refer to nodes that drive gates of transistors whose source or drain is connected to \( z \) as the fan-in of \( z \). Similarly, we refer to nodes that serve as the source or drain of transistors whose gate is driven by \( z \) as the fan-out of \( z \).

We say that the state of node \( z \) at time \( t \) deviates during simulation \( i \) if and only if the state of \( z \) at time \( t \) during simulation \( i \) differs from its state at time \( t \) in simulation \( i-1 \), i.e., \( S_i(z,t) \neq S_{i-1}(z,t) \). The state of node \( z \) at time \( t \) can deviate during simulation \( i \) only if one or more of the following three conditions are true:

(a) the subnetwork containing \( z \) has been modified, i.e., \( F_i \) has changed,

(b) the delay assigned to a node in the subnetwork containing \( z \) has been modified, i.e., \( D_i(y) \neq D_{i-1}(y) \) for \( y \in \text{subnet}(z) \), or

(c) an argument of \( z \) deviates at time \( t \).

Conditions (a) and (b) represent explicit changes to a subnetwork by the user that occur between simulation runs. Condition (c) is a result of state changes propagating through the circuit from touched nodes (nodes involved in explicit changes by the user). We say node \( z \) is active at time \( t \) for simulation \( i \) if it is touched, it deviates at time \( t \), or the state of a fan-in of \( z \) deviates at time \( t \). Otherwise the node is considered to be inactive. We say a subnetwork (dynamic or static) is touched if it contains a touched node, active at time \( t \) if it contains an active node at time \( t \), and inactive at time \( t \) if it does not contain any active nodes at time \( t \).

The state of a node whose subnetwork is inactive at time \( t \) for simulation \( i \) is identical to its state at time \( t \) for simulation \( i-1 \). We can avoid reevaluating inactive subnetworks during resimulation by accepting the state of their nodes from the previous simulation. The value of \( S_i(z,t+D_i(z)) \) can therefore be determined from:

\[
\begin{align*}
F_i(S_i(A,t)) & \text{ if subnet}(z) \text{ active} \\
S_{i-1}(z,t+D_i(z)) & \text{ if subnet}(z) \text{ inactive}
\end{align*}
\]

(2),

that is, whenever a node is in an active subnetwork its state is derived from the state equation applied to the current state of the arguments of the node. Whenever a node is in an inactive subnetwork, its state is derived from the results of the previous simulation.

For any node \( z \) during simulation \( i \) the state of \( z \) at time \( t + D_i(z) \) can differ from its state at time \( t \) if one of its arguments changes state at time \( t \). We say a subnetwork is stimulated at time \( t \) if one of its arguments change state at time \( t \), i.e., for some \( y \) in \( A \), \( S_i(y,t) \neq S_{i-1}(y,t-1) \). The value of \( S_i(z,t+D_i(z)) \) is now given by:

\[
\begin{align*}
F_i(S_i(A,t)) & \text{ if subnet}(z) \text{ active and stimulated} \\
S_i(z,t+D_i(z)-1) & \text{ if subnet}(z) \text{ active but not stimulated} \\
S_{i-1}(z,t+D_i(z)) & \text{ if subnet}(z) \text{ inactive}
\end{align*}
\]

(3).

Whenever the subnetwork of a node is active and stimulated its state is derived from the evaluation procedure applied to the current state of the arguments of the node. Whenever the subnetwork of a node is active but not stimulated, its state can be derived from the previous time. Whenever the subnetwork of a node is inactive, its state can be derived from the history of the previous simulation.

**5. INCREMENTAL SIMULATION**

Equation (3) shows that during incremental simulation state values can be obtained from either the current simulation state history or the previous simulation state history, depending on the active status of the subnetwork. To ensure correct results and to minimize the number of subnetwork evaluations that occur during incremental simulation it is important to recognize the active status of subnetworks as soon as a change from active to inactive or inactive to active occurs.

There are two ways that a subnetwork can change its active status: either some argument changes state during the current simulation, or some argument changes state during the previous simulation. In the former case, whenever a node is assigned a new state, it is necessary to redetermine the active status of the fan-outs of that node. In the latter case, an active subnetwork must have its active status rechecked for each time in the history trace of the previous simulation that an argument changes state (but only as long as the subnetwork remains active).

By equation (3) the only time a subnetwork requires reevaluation during incremental simulation is when it is both
active (it contains an active node) and stimulated (some argument changes state). During the period in which a subnetwork is active it is necessary to reevaluate the subnetwork whenever there is a change in state on any argument. An argument which stimulates the active subnetwork can itself be either active or inactive. If it is inactive, the same state change also occurs at time $t$ in the previous simulation, and an evaluation of the argument is not required even though an evaluation of the stimulated subnetwork is. If the argument is active, the state change has occurred due to an evaluation of the argument during the current simulation. During the span of time that a subnetwork is active, the state change also occurs at time $t$ in the previous simulation. If node changed state then update history; schedule check event for fan-outs at time $t$; endif endfor

for all time $t$ check events (in topological order) do expand the subnet; cancel all pending check events for subnet; extract state of node; if subnet is active and stimulated then evaluate subnet; schedule settle events for positive delay subnet nodes; for each zero delay node in subnet do if node changed state then update history; schedule check event at time $t$ for fan-outs; endif endfor endfor schedule check event at time $t$ for fan-outs of all subnet nodes that started/stopped deviating; let $r$ be the time of the EFE of the subnet; schedule check events at time $r$ for active nodes in subnet; endwhile

Fig. 1. The incremental simulation algorithm.

6. PERFORMANCE

ICESIM is implemented in C on a SUN SPARCstation 2. The system supports both nonincremental and incremental simulation, and applying either simulation method yields identical results. Experiments were carried out on the ISCAS combinational [2] and sequential [1] benchmark test circuits. Although these are gate logic designs, for our purposes, the gates were defined using standard CMOS design techniques. The size of the resulting circuits ranged from 414 transistors to 89,046 transistors.

One hundred randomly generated test vectors were applied to the primary inputs of each circuit (the sequential circuits also required a clock signal). Each design was subjected to a series of ten randomly generated design flaws. Each flaw was generated as follows: A single subcomponent pin was selected at random. The pin was then (randomly) either disconnected from whatever wire it was connected to leaving it unconnected, or disconnected and reconnected to
some other (randomly chosen) wire in the same schematic. After each design flaw was introduced, the design was simulated for all one hundred test vectors, the flaw was corrected, the design was incrementally resimulated, and then (for comparison) nonincrementally simulated. The statistics presented in this section were gathered from the latter two simulations for each corrected flaw. The experiments were carried out twice for each design. The first using zero-delay wherever possible, the second using unit-delay on every node. Both sets of experiments used the same test vectors, and the same design flaws. For the zero-delay experiments, zero-delay cycles were broken by introducing unit-delays on feedback nodes (necessary for the sequential circuits).

Shown in Figure 2 is a summary of the relative speedup encountered using the incremental simulator on the ISCAS test circuits. Due to lack of space, we give only the results for the larger circuits (those having more than 10,000 transistors). The reader is referred to the complete paper[8] for more detailed results of the experiments. The relative speedup for a particular application of the incremental simulator is determined by dividing the CPU time used by the nonincremental simulator by the CPU time used by the incremental simulator on the same circuit. For each design and simulation delay, the table gives the median relative speedup which occurred during the ten resimulations. The median relative speedup is indicative of the typical speedups achieved using incremental simulation.

The speed advantage zero-delay has over unit-delay for nonincremental simulation is well known. The performance difference comes from two factors. First, zero-delay event processing is simpler than with unit-delay. Second, zero-delay tends to generate fewer events due to the rank-ordered synchronization of evaluations. In our experiments the combination of these factors meant that zero-delay nonincremental simulation was about 1.6 times faster (on average) than unit-delay for the same circuit. These differences are more pronounced when considering incremental simulation due to the higher complexity of processing positive-delay nodes in the incremental evaluator and the increase in events caused by time-shifts. Overall, zero-delay incremental simulation was about 4 times faster (on average) than unit-delay incremental simulation for the same circuit modification.

<table>
<thead>
<tr>
<th>test</th>
<th>delay</th>
<th>zero delay</th>
<th>unit delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_{5288}$</td>
<td>10112</td>
<td>15.66x</td>
<td>3.18x</td>
</tr>
<tr>
<td>$c_{5315}$</td>
<td>11270</td>
<td>201.05x</td>
<td>132.75x</td>
</tr>
<tr>
<td>$c_{7552}$</td>
<td>15306</td>
<td>118.74x</td>
<td>58.74x</td>
</tr>
<tr>
<td>$s_{5234}$</td>
<td>19854</td>
<td>714.33x</td>
<td>49.06x</td>
</tr>
<tr>
<td>$s_{13207}$</td>
<td>28927</td>
<td>3427.20x</td>
<td>68.82x</td>
</tr>
<tr>
<td>$s_{15850}$</td>
<td>34933</td>
<td>1800.00x</td>
<td>14.98x</td>
</tr>
<tr>
<td>$s_{58634}$</td>
<td>75846</td>
<td>267.72x</td>
<td>98.14x</td>
</tr>
<tr>
<td>$s_{38417}$</td>
<td>80996</td>
<td>4620.32x</td>
<td>341.42x</td>
</tr>
<tr>
<td>$s_{38558}$</td>
<td>89046</td>
<td>503.32x</td>
<td>27.75x</td>
</tr>
</tbody>
</table>

Fig. 2. Relative speedup using incremental simulation.

7. REFERENCES


