Reducing the Scheduling Cost in Event-Driven Simulation Through Component Clustering*

David T. Blaauw

Engineering Accelerator Development
IBM Endicott
Endicott, NY, 13760

Larry G. Jones

Physical Design Technology
Motorola, Inc.
Austine, TX, 78735

ABSTRACT

Event-driven simulation has become a prominent method of circuit verification. Although the event-driven scheme effectively reduces the amount of circuit evaluation performed during the simulation, a substantial scheduling cost is incurred to determine which elements must be evaluated in a simulation cycle. In this paper, we propose a method to reduce this scheduling overhead. Circuit elements that are likely to be activated simultaneously are grouped into clusters and are scheduled and evaluated simultaneously. This way, the number of entities involved in the scheduling, and therefore the scheduling overhead, is reduced. Different algorithms for partitioning the circuit into clusters are presented and the relationship between the cluster size and the simulation performance is studied. The presented algorithms were implemented for a switch-level simulator and tested for several large circuit descriptions. It is shown that with a carefully picked cluster size and partitioning algorithm, the scheduling overhead in the simulation can be significantly reduced and the simulation efficiency improved.

1. Introduction.

Event-driven simulation has become an indispensable tool in the design of large scale integrated circuits. Initially, simulation was performed using the compiled simulation approach. In this approach, the circuit elements are sorted topologically and are evaluated according to this ordering. Each circuit element is, therefore, evaluate at each simulation cycle. More recently, the event-driven simulation approach was introduced [1,2]. This approach is motivated by the observation that for most circuits, only a small portion of all circuit nodes exhibit change in a simulation cycle. The event-driven simulator takes advantage of this fact by only evaluating the circuit elements that are affected by these nodes. The event-driven simulator uses a scheduler which tests circuit nodes for change and schedules the fan-out of these nodes for evaluation. For each simulated input pattern, only a small percentage of all circuit elements is evaluated and a significant amount of circuit evaluation is avoided.

Event-driven simulation consists of two distinct components, namely the scheduling algorithm and the evaluation algorithm. The scheduling algorithm determines which circuit components are evaluated and in which order. The evaluation algorithm determines the next state of a component given the inputs and initial state of the component. The total cost of event-driven simulation is the sum of the scheduling and evaluation costs. In switch-level simulation, the circuit components correspond to sets of channel-connected transistors, also called dc-connected components. Since in switch-level simulation, the evaluation algorithm has traditionally contributed the dominant cost of the event-driven simulation, much emphasis has been placed on developing fast evaluation techniques. Several methods using presimulation analysis have been developed that perform very efficient evaluation of dc-connected components [3-5].

With the decreasing cost of component evaluation, the cost of the event-driven scheduler has become a more significant factor in the performance of the simulation. In this paper, we propose a method for reducing the scheduling overhead involved in event-driven simulation. Since the scheduling cost is linear in the number of nodes in the circuit description, it can only be improved on by a constant factor. Algorithmic improvements are, therefore, limited in their scope. Instead, we focus on reducing the problem size, i.e. the number of circuit nodes and components involved in the scheduling. Circuit components that are likely to be activated simultaneously are grouped into so-called *component clusters*. Each component cluster is scheduled and evaluated as a unit during the simulation of the circuit. A cluster of components is evaluated each time any component in thecluster is affected by a change in the circuit state. Forming component clusters reduces the number of entities involved in the scheduling algorithm and, therefore, the scheduling overhead. When a cluster is evaluated, however, it might include components that do not exhibit change on their inputs in that particular simulation cycle. The cluster approach, therefore, can increase the evaluation cost of the simulation by performing unneeded component evaluation.

The cluster size presents an inherent trade-off between the scheduling cost and evaluation cost of the simulation. As the cluster size increases, the number of clusters and nodes that are involved in the scheduling decrease, thereby diminishing the scheduling overhead. However, the probability that the evaluation of a cluster includes unnecessary evaluation of circuit

* This research was supported in part by the Semiconductor Research Corporation Contract 90-DP-142, at the University of Illinois, and in part by NSF Grant MIP-87-10865.

components increases, thereby increasing the evaluation cost. If the cluster size is chosen too large, the simulation will evaluate a large number of unneeded circuit components, and incur a high evaluation cost. If, on the other hand, the cluster size is chosen too small, the simulation incurs a high scheduling overhead. The optimal cluster size, therefore, balances the evaluation and scheduling costs. In this paper, we perform an empirical study of the clustering approach using the SNEL simulator [4] and a number of large switch-level circuits. It was found that given a particular simulator and circuits with a node activity within a reasonable range, the simulation speed can be effectively increased using a fixed cluster size. The proposed method is applied specifically to switch-level simulation in this paper. Its application can, however, be also extended to other levels of event-driven simulation.

The remainder of this paper is organized as follows. Section 2 discusses related work and gives an iverview of clustered event-driven simulation. Section 3 presents the algorithms for clustered event-driven simulation and circuit partitioning. Section 4 presents the results of the clustering approach. Section 5 offers some concluding remarks.

2. Clustered Event-Driven Simulation

In switch-level simulation, it has been recognized that very large dc-connected components reduce the efficiency of the simulation. In [6], a method was proposed to dynamically partition large dc-connected components into sub-components during the simulation. It was found that reducing the cluster size in this sense increases the simulation performance. In [5] it was observed, on the other hand, that if the component evaluation cost of the simulator is small, it can be more effective to simply evaluate all circuit components at each simulation cycle and avoid the scheduling overhead. This approach is generally taken in hardware accelerators which perform very fast component evaluation and often do not support constructs needed to implement event-driven scheduling. For many simulators, however, the optimal cluster size might fall between the two extremes of event-driven and compiled simulation.

To illustrate the trade-off between the evaluation and scheduling costs in event-driven simulation, a simple gate-level circuit description is shown in Figure 1. In standard event-driven simulation, all of the input and internal nodes will be tested for change during the simulation. If node *inp0* changes state it will produce the evaluation of gate *nand1*. Depending on input *inp1*, the evaluation of gate *nand1* will result in a change in the state of node *n0*. This change will again be detected by the scheduling algorithm and will result in the evaluation of inverter *inv1*. If, however, gate *nand1* and *inv1* are placed in one cluster, they will both be immediately evaluated upon a change of input *inp1*. Since the scheduler does not need to test node *n0* for change and separately schedule gates *nand1* and *inv1*, the scheduling overhead is reduced. If, however, input *inp1*

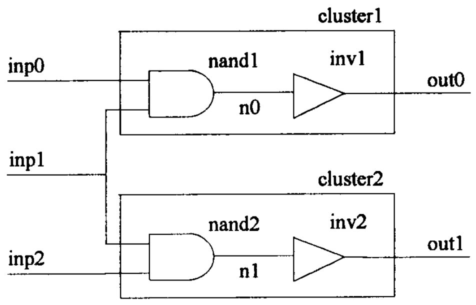


Figure 1. Example circuit for clustering approach.

is logic zero, the evaluation of gate *nand1* does not result in a change of node *n0*. In this case, the clustering approach performs an unneeded evaluation of inverter *inv1*, which potentially diminishes its efficiency.

The clustering method is, therefore, only effective if the partitioning algorithm and the cluster size are carefully assessed. The optimal cluster size, is a function of the relative efficiency of the scheduling and evaluation algorithms of the simulator. If the evaluation of a circuit component is relatively expensive compare to the effort involved in scheduling it, smaller cluster sizes will be more effective. Fast component evaluation methods, therefore, advocate larger cluster sizes. The node activity rate of the circuit nodes also affects the optimal cluster size. The node activity rate is the ratio of the number of circuit nodes that exhibit change in a simulation cycle over the total number of nodes in the circuit, and is circuit and input dependent. If the circuit has a very high node activity, the probability that a circuit component will be evaluated is high, and the probability of performing unneeded component evaluation in a cluster is low. A high node activity rate, therefore, increases the optimal cluster size.

3. Evaluation and Partitioning Algorithms

When the cluster-size is larger than one, an algorithm is needed to group circuit components into circuit clusters. A number of general algorithms for minimizing the number of nodes between partitions in a graph, also called min-cut algorithms, have been proposed. One of the most well known of these min-cut algorithms is the algorithm by Kernighan and Lin [7]. Min-cut algorithms seek to minimize the total interconnections between clusters in the circuit. Since the scheduling time is directly proportional to the number of nodes in the circuit, the min-cut algorithms will tend to reduce the total scheduling overhead. However, the evaluation time is not necessarily minimized by these algorithms. All circuit components in a cluster are evaluated each time one or more inputs of the cluster change state. It is, therefore, advantages to place tightly coupled circuitry that is effected by the same circuit node changes together. To minimize the evaluation time, one must foresee which circuit components are affected by an event and to place these components in one cluster.

Three types of partitioning algorithms were implemented: *depth-first-forward*, *depth-first-backward*, and *breadth-first*. The code for the depth-first-forward algorithm is shown in Figure 2.

```
depthFirstForward(targetSize)
{
  cluster = getNewCluster();
  component = componentNew = NULL;
  clusterSize = 0;
  while(componentsLeft > 0) {
    if(component) {
      do{  //work from a previously placed component //
        componentNew = getFanout(component, targerSize, clusterSize);
        if (componentNew) putStack(component);
        else component = popStack();
      } while(!componentNew && component);
      component = componentNew;

   if (!component) component = getNewComponent();

   if (newCluster(size(component), targetSize, clusterSize, circuitSize)) {
      cluster = getNewCluster();
      clusterSize = 0;
   }

   addComponentToCluster(component, cluster);
   clusterSize += size(component);
   componentsLeft --;
   circuitSize -= size(component);
} }
```
Figure 2. Partitioning algorithm for depth-first-forward scheme.

The algorithm performs a depth-first traversal of the circuit, starting from the circuit inputs. Circuit components are then placed together in the order that they are visited. Variable *targetSize* is the chosen cluster size. Variable *componentsLeft* initially contains the total number of circuit components in the circuit and is decremented as they are assigned to circuit clusters. The function *getFanout()* examines the fan-out of a component. The fan-out component that creates the best match with the target cluster size is then chosen. If a component has no unassigned fan-out components, a new component is obtained from function *getNewComponent()*. This function searches through the rank-table of the circuit and returns the lowest ranked, unassigned component. After a new component is found, function *newCluster()* decides if it is added to the current cluster or if a new cluster is started. The function uses variable *circuitSize* which contains the total size of all unassigned components in the circuit. It is was found that careful crafting of the function *newCell()* greatly balance the size of the generated clusters.

The *depth-first-backwards* partitioning algorithm operates similarly to the shown algorithm, except that it starts from the outputs of the circuit and progresses to the circuit inputs. The *breadth-first* algorithm performs a breadth-first traversal instead of a depth-first traversal.

The cost function used in the partitioning scheme (function *size()* in Figure 2) plays an important role in the partitioning process. It expresses how `expensive' the components are and determines how many and which components are placed in a cluster. The cost function should, therefore, reflect the actual evaluation cost of a cluster. In switch-level simulation, the evaluation cost of a dc-connected component depends on the used evaluation algorithm. In the SNEL simulator, the evaluation cost of a dc-connected component is linear in the number of transistor in the component. The cost function, in this case, simply returns the number of transistors in the cluster. Depending on the simulator, other cost functions must be used that accurately reflect the evaluation cost of a cluster.

The clustering approach requires only minor modification to the traditional event-driven simulation algorithm. Instead of evaluating individual components, the simulation evaluates clusters of components. Scheduled clusters are evaluated in a topologically sorted order, until no more clusters are scheduled for evaluation, and the simulator proceeds to the next time step. The evaluation of each cluster generally involves the evaluation of several components. The components within a cluster are, therefore, topologically sorted prior to the simulation, and are then evaluated during the simulation in this predetermined order.

Each time a cluster is evaluated the outputs of the cluster are tested forchange. If the cluster output change was produced by zero-delay circuit elements, the node change takes affect immediately and all fan-out clusters of the node are scheduled for evaluation in the current simulation time step. If the circuit description contains circuit components with unit or multiple delay, the cluster evaluation must take into account the delay of the individual components in the cluster. We define the ***relative delay*** of cluster node n as the difference between the time of a cluster input node change and the time of the subsequent change in node n. A cluster that is evaluated for time t, therefore, schedules an change in cluster output node n for time $t + \delta$, where δ is the relative delay of the node n. When the simulator increments its clock to time $t + \delta$, the logic state of the output node is updated with the scheduled value. If a change has occurred in its logic state, the cluster fanout of the node is then scheduled for evaluation.

To illustrate the evaluation procedure in the presence of delay, consider cluster 1 in Figure 1, when the two gates ***nand1*** and ***inv1*** are both assigned a delay of one unit. If the cluster is evaluated at simulation step t, the evaluation of ***nand1*** produces a potential change for node ***n0*** at time $t+1$ and the evaluation of inverter ***inv1*** produces a potential change for node ***out0*** at time $t+2$. The relative delay of node ***out0*** is, therefore, two delay units. Since node ***n0*** has no fan-out to other clusters, the cluster evaluation can simply access nodes ***inp0*** and ***inp1*** at time t, evaluate gate ***nand1*** and ***inv1*** and schedule the potential output change of node ***out0*** for time $t+2$. The scheduling of node ***n0*** is, therefore, eliminated from the simulation and is implicitly performed in the cluster evaluation. If node *n0* does have fan-out of other clusters, the cluster evaluation must schedule a node change of *n0* for time *t+1* in addition to scheduling node *out0* for time *t+2*.

The relative delay of the nodes in a cluster is calculated by examining the circuit components in their topological ordering. All cluster input nodes are assigned a relative delay of zero. As circuit components are traversed, the output of a circuit component is assigned a relative delay equal to the relative delay of the component inputs incremented by the delay of the circuit component. It is possible that the inputs of a circuit component have a different relative delay associated with them. In this case, the circuit component can not be evaluated in the circuit cluster, since not all of its inputs will be available for the correct time step. The partitioning algorithm, therefore, calculates the relative delay of cluster nodes as circuit components are incorporated in the cluster. If a circuit component has inputs with an uneven relative delay, it is then excluded from the circuit cluster. This restriction on the partitioning ensures that all inputs of a circuit component in a cluster have an equal relative delay. It should also be noted that this restriction prevents cyclic constructs from being incorporated in a single cluster. Similarly, we exclude the presence of zero-delay cycles in the circuit. In [8], it is shown that zero delay cycles result in unpredictable simulation results.

4. Simulation Results of the Clustered Event-Driven Approach.

The effect of the cluster size on event-driven simulation is investigated using the switch-level simulator SNEL [4]. The SNEL simulator first partitions the circuit into clusters of one or more dc-connected components. For each component cluster, evaluation code is generated using a compiled simulation method. The circuit clusters are scheduled and evaluated in an event-driven approach. The circuit cluster size, can be varied from encompassing one dc-connected component, to encompassing the entire circuit description. The SNEL simulator can, therefore, emulate traditional event-driven simulation, complete compiled simulation, or a hybrid using a specified cluster size.

The optimal cluster size was studied for a number of switch-level circuits, including two commercial microprocessor. For each circuit, the cluster size was varied from a single dc-connected component to the entire circuit description. For all circuits, the depth-first forward partitioning method was used. Table 1 shows the tested circuits and their characteristics.

Circuits *c1908* through *c7552* are ISCAS-85 combinational benchmark circuits [9] implemented in static CMOS. Circuits *adderRC* and *adderLA* are, respectively, a 64-bit ripple-carry adder and a 32-bit look-ahead adder. Circuit *multiplier* is a 16-by-16 multiplier array. Processor 1 and 2 are commercial microprocessors implemented in custom CMOS.

circuit	number transistors	node activity	dcc size	cluster size	speed up
c1908	3482	6%	2.99	38.26	3.06
c3540	7632	5%	2.94	27.85	3.40
c5315	11270	5%	3.07	40.54	2.81
c6288	10112	10%	3.78	91.10	1.74
c7552	15396	6%	2,98	43.13	2.24
adderLa	2380	22%	4.59	25.32	1.42
adderRc	3328	4%	5.70	25.02	1.48
multiplier	13600	10%	5.70	24.37	1.48
proc1	20177	11.3%	6.1	34.79	1.36
proc2	41062	7.9%	5.3	30.22	1.58

Table 1.

With the exception of *proc1* and *proc2*, all circuits were simulated for 10,000 cycles with randomly generated test patterns. For circuit *proc1* and *proc2*, functional test patterns developed during the design of the processors were used. This gives these test cases the advantage that they realistically reflect the demands placed on logic simulation during the design process.

For each circuit, Table 1 shows the total circuit size in transistors (*number transistors*), the circuit node activity (*node activity*), and the average dc-connected component sizes (*dcc size*). The column *cluster size* shows the optimal cluster size in transistors as determined by the simulation runs. The column *speed-up* shows the speed-up of the simulation with the optimal cluster-size over traditional event-driven simulation. The simulation performance was improved by a factor ranging from 1.36 to 3.40 using the clustering approach.

Figure 3 and Figure 4 show the simulation performance of the two processors as a function of the cluster size. The total simulation time, as well as the scheduling time and evaluation time are shown. The scheduling time initially decreases sharply as the cluster size increases and then slowly levels of. Using the optimal cluster size, the scheduling overhead was reduced by a

Figure 3. Simulation time vs. cluster size for proc1

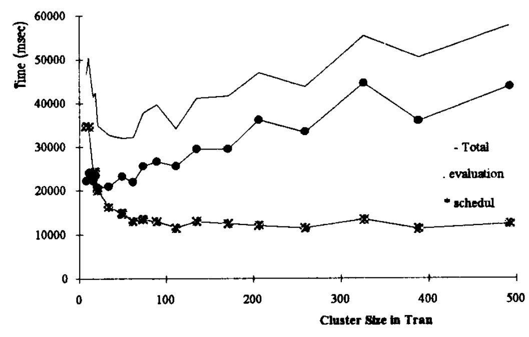
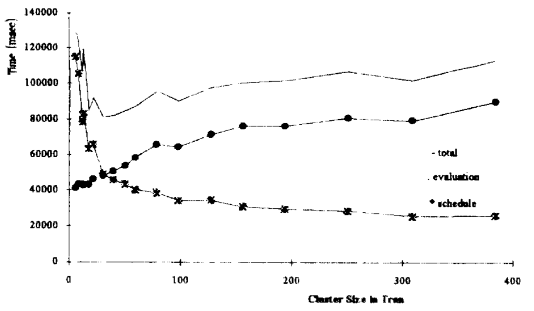


Figure 4. Simulation time vs. cluster size for proc2



factor of 2.0 for processor 1 and 2.35 for processor 2. The evaluation time shows a steady increase with the cluster size. For both processors the total simulation time initially decreases sharply with the cluster size and then, after passing the optimal cluster size, increases slowly.

Comparison between the simulated circuits shows that the optimal cluster size varies for the different circuits. For processors 1 and 2, the optimal cluster size is, respectively, 34.79, and 30.22 transistors. For circuits *mult*, *adderRC*, and *adderLA*, the optimal cluster size fell between 24.37 and 25.32, and for the ISCAS bench-mark circuits the optimal cluster size fell between 27.85 and 91.10. This raises the question of how to choose the optimal event-size for a circuit that has not yet been simulated. A possible approach is to simulate the circuit for a range of event-sizes with a small subset of the input vectors. The optimum event-size for this subset is then determined and used for the entire simulation. This approach, however, can involve a large amount of overhead.

A simple and effective approach is to use a fixed cluster size for all simulations. For a cluster size in the range of 25 to 50 transistors, the simulation performance for all simulated circuits is within 15% of its optimal performance. Although the simulation time increases drastically for very large and very small cluster sizes, it is relatively constant in this "middle range". Any cluster size from this range is, therefore, likely to perform well for most simulated circuits. In the SNEL simulator, the cluster size is set at 30 transistors. Using this cluster size, all tested circuit operated within 12% of their optimal performance.

The three partitioning algorithms discussed in Section 2 were also compared for the tested circuits. Of the three partition algorithms depth-first approach showed the best performance. The approaches differ most for large cluster sizes. Within the cluster size range of less than 60 transistors (10 dc-connected components), the partition algorithms all produced simulation results within approximately 20% of each other. Within this range only a few dc-connected components are grouped together. The coupling between these components is likely be high regardless of the partition algorithm. The algorithms, therefore, show similar performance in this range.

5. Conclusions

In this paper, a new approach to reduce the scheduling overhead in event-driven simulation is proposed and is implemented for a switch-level simulator. Circuit components are grouped into clusters which are then scheduled and evaluated as a unit. Three algorithms for partitioning the circuit into clusters, as well as, the needed modifications to the event-driven algorithm were presented. The cluster size represents a trade-off between the scheduling and evaluation costs of the simulation. The optimal cluster size balances these costs and was found to significantly increase the performance of the simulation. Although the optimal cluster size is a function of the circuit characteristics, it was found that good performance was obtained for all test circuit using a fixed cluster size.

References

[1] P. W. Case, H. H. Graff, L. E. Griffith, A. R. LeClerq, W. B. Murley, and T. M. Spence, "Solid Logic Design Automation," *IBM Journal*, pp. 127-147, April 1964.

[2] E. G. Ulrich, "Event Manipulation for Discrete Simulation Requiring Large Number of Events," *Communications of the ACM*, pp. 777-785, September 1978.

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

[4] D. T. Blaauw, R. B. Mueller-Thuns, D. G. Saab, P. Banerjee, and J. A. Abraham, "SNEL: A Switch-Level Simulator Using Multiple Levels of Funcational Abstraction," *International Conference on Computer Aided Design, pp. 66-69, 1990.*

[5] Z. Barzilai, D. K. Beece, L. M Hiusman, Iyengar, and G. M. Silberman, "SLS - A Fast Switch-Level Simulator," *IEEE Trans. on CAD*, pp. 838-849, August 1988..

[6] L. McMurchie, C. Anderson, and G. Borriello, "Hybrid Compiled/Interpreted Simualation of MOS Circuits," *Proc European Design Automation Conference,* 1991.

[7] G W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Tech. Journal*, pp. 291-307, 1970.

[8] S. Gai, F. Somenzi, and M Spalla, "Fast and Coherent Simulation with Zero Delay Elements," *IEEE Trans. on CAD*, pp85-92, January 1987.

[9] F. Brglez and H. Fujiwara, "A Neural Netlist of 10 Combinational Bench Mark Circuits and a Target Translator in FORTRAN," *Proc ISCS-85*, pp. 151-158, June 1985.