VLSI Timing Simulation with Selective Dynamic Regionization

Meng-Lin Yu Bryan D. Ackland

AT&T Bell Laboratories Holmdel, NJ 07733

Abstract

Accurate timing simulations are crucial to the design of MOS VLSI circuits, but can take prohibitively large amounts of time. This paper describes dynamic regionization techniques applied to an event based simulator for MOS timing simulation that have proven to be more efficient and as accurate as the static regionization method. The MOS network is first statically partitioned into groups of strongly coupled nodes called regions. Big regions are then incrementally and dynamically partitioned into and replaced by subregions. Subregions are treated just like normal regions in the event based simulation process. This simulator has been used to verify the timing and functionality of several large VLSI chips. Performance is 3 to 7 times faster than a static regionization method.

1. Introduction

The increasing density and level of integration of VLSI systems has necessitated a spectrum of simulation tools to verify large designs. Circuit simulation programs (such as SPICE) have provided the VLSI designer with an accurate, detailed waveform of the analog behavior of small (< 1000 transistors) circuits, but the simulation is very computationally intensive and is normally reserved for critical subcircuits. Logic and switch level simulators have been used to show the function and approximate timing characteristics of very large digital systems, but they cannot always provide sufficiently accurate timing Growing out of the need for accurate information. simulation with timing waveforms of very large circuits (>100,000 transistors at whole chip level), timing simulators present an interesting tradeoff between accuracy and speed. By using specific domain knowledge, i.e., making assumptions about the nature of digital MOS devices that lead to simple device models and allow for effective decoupling of circuit equations, timing simulators provide accurate waveform and timing information of MOS circuits at speed up to two orders of magnitude greater than circuit simulators.

Early timing simulators [1-3] used the unidirectional property of the MOS gate terminal to split the circuit up into subcircuits which were evaluated independently once each time-step. Subcircuit evaluation consisted of linearizing device characteristics around the operating point and integrating circuit equations over the time-step using simple forward or backward Euler approximations.

Permission to copy without fee all or part of this material is granted, provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission. Relaxation based simulators [4-6] have used iteration to improve accuracy and allow simulation of a broader class of networks (*e.g.*, floating capacitors). They also use a locally variable time-step to take advantage of the multirate behavior of MOS circuits and thereby improve efficiency.

The latency and multirate behavior of MOS circuits can be further exploited by developing algorithms which are incremental in node voltage, rather than time. A specified change in node voltage can be viewed as a "circuit event" and scheduled using an explicit event queue, in much the same way as logic events are scheduled in a logic simulator. Circuit evaluation consists of calculating the time at which these events should occur. Such an approach has been adopted by many simulators [7, 8, 10]. A very successful example is Emu2 [10], an MOS timing simulator offered by AT&T SYSCAD tools. Emu₂ provides a more accurate simulation, yet runs 2-5 times faster than the older time-step based Emu[3] (200-300 times faster than SPICE). Even for large VLSI circuits, Emu2 offers full chip simulation capability and has been used successfully in verifying the performance of a number of VLSI circuits. Note that since Emu2 is widely used in the custom design community inside AT&T, its capability, accuracy, and limitations are well understood by the designers through repeated usage.

While Emu2 is much faster than SPICE, it is still very slow for simulating large designs. Experiences indicate that simulating circuits with a large random access memory (RAM) is particularly slow, even though the level of circuit activities for the RAM is low. Since large RAMs are very common in large designs, simulation efficiency can be further improved if we can speedup RAM simulation.

This paper describes EMU2D, a simulator with selective dynamic regionization based on Emu2. While EMU2D uses the same circuit modeling, solution method, and event based technique as Emu2, regions with large numbers of nodes and with certain structural characteristics (e.g., those found in RAMs) are partitioned dynamically into subregions such that we can further take advantage of the multirate behavior of MOS circuits as each subregion can be evaluated according to its own rate. Further, the solution method computations for big regions are more efficient as there are now few nodes participating in the computation. As a result, EMU2D provides the same accuracy as Emu2 but runs 3-7 times faster than Emu2.

In Section 2 we describe some background on timing simulation by briefly explaining algorithms and models used in Emu2. We also explain why dynamic regionization is worth exploring. This is followed by detailed descriptions of dynamic region selection criteria and dynamic region manipulations in Section 3. Results which demonstrate the speedups over Emu2 are presented in Section 4.

2. Timing Simulation Basics and Motivation 2.1 Basics of Emu2

Emu2 models an MOS circuit as a collection of capacitive nodes interconnected by linear resistors and capacitors and non-linear voltage controlled current sources (MOS transistors). Prior to simulation, nodes are (statically) grouped into *regions* of strong coupling/connected subcircuits connected by a charge transfer path (resistor, capacitor, transistor channel). An example of region partitioning is shown in Figure 1.

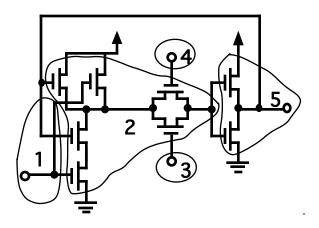


Figure 1. Region subdivision

Emu2 employs a piecewise linear numerical technique for circuit evaluation. Based on previous node voltages, the transistor currents and conductances are computed and assumed to stay constant for a small time step. Transistor evaluation is based on a simplified version of the CSIM model [9], using a mixture of arithmetic calculation and table look-up. Within regions, nodes are tightly coupled. Accurate simulation therefore requires that we solve these nodes simultaneously using newly calculated transistor currents and conductances. A combination of Backward Euler integration and a Gauss-Seidel linear relaxation iterating till convergence method is used to advance the state of each region over a small time interval (See [10] for A Forward Euler estimate provides an initial details). guess for the relaxation process. Between regions, all interactions occur via MOS gate terminals. The node coupling is therefore considerably weaker and always This allows us to evaluate regions unidirectional. independently using most recently calculated values for nodes in other regions. Regions interact whenever the voltage of a node in a region changes by more than a preset threshold. This is termed an *event*. Events are predicted, stored and scheduled on an event queue, using similar techniques to those employed in logic simulators. The interactions between regions are thus voltage step based, rather than time-step based and naturally accommodate varying circuit activity.

The Emu2 simulation algorithm can be summarized by the pseudo code in Figure 2.

loop forever until done {

for (every event <region r, and time t> at current time t) {
 estimate initial solution using forward Euler integration;
 apply iterative relaxation to find a solution for all node
 voltages in the region;
 if (voltage change of node N in r > threshold)
 then evaluate N's fan-out regions recursively;
}
for (every region R evaluated above) {
 compute current and conductance for every mosfet in R;
 predict next event time for R using forward Euler
 integration and post the event;
}

advance to the next smallest simulation time of all outstanding events;

Figure 2. Simplified Pseudo Code for Emu2 Timing Simulation Algorithm

2.2 Motivation

}

While Emu2 is much faster than SPICE, it is still very slow for simulating large designs. The problem is more severe/noticeable when designers do whole chip pre-fabrication simulation, after-fabrication debugging, or regression tests. Early study pointed out that there were some very large regions in most large circuits and that a large amount of time (70% - 90%) was spent in simulating these relatively few large regions (less than 1% to 5%.) Improving the efficiency of evaluating large regions can thus potentially lead to significant speedup.

Further analysis indicates that large regions are usually found in highly parallel multipliers, bus structures, registers, ROMs, RAMs, and switch structures (e.g., FPGAs). With the exception of the multiplier, these structures exhibit a behavior in which very few charge transfer paths (channels) are conducting at any given time. Rather than behaving as a single large region, the structure behaves as a number of small disconnected regions. The number and boundaries of these smaller regions, however, changes with the electrical state of the circuit. Many static connections (by mosfet channels) are effectively "noconnection" dynamically because the mosfets are cutoff and non-conducting. This observation implies that in theory more efficient timing simulation is possible if we can

dynamically define smaller regions and evaluate only those that are active rather than the original large static region.

The concept of dynamically partitioning the circuit in timing simulation is not new. In [7], an explicit solution method is used to solve the linearized nodal equations. A separate dynamic node connectivity list is kept for each node. Node connections with negligible electrical influences to a node are dropped from the node's dynamic connectivity list. Since equations are solved for each node, the overhead for dynamic partitioning is in checking and maintaining the dynamic connectivity list. This dynamic connectivity change is difficult to use in a region based simulator as the solution method solves node voltages for each region and the regions may fragment. Neither region recognition nor maintaining fragmented regions is easy in general. A complete dynamic regionization requires structural analysis after each region evaluation and its cost can be prohibitively high. Success of this concept clearly depends on efficient programming and data structures. In the next section, we will show techniques and implementations that make dynamic regionization feasible.

3. Dynamic Regionization

3.1 Techniques

A common feature of large regions that have sparse dynamic connections is the high static connectivity to a small number of nodes inside a region. This can be illustrated by the memory structure in Figure 3.

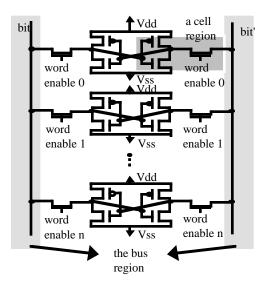


Figure 3. A Typical 6-Transistor RAM Region

Memory cells and the *bit* and *bit*' busses have been grouped into one large region by the word enable mosfet transistors. *bit* and *bit*' busses are often grouped together by mosfets in the write circuitry (not shown). In normal operation, only two of these word enable mosfets are conducting regardless of how many memory cells there are in the region. This means that the original region effectively becomes many isolated *cell regions*, (one per memory cell,) and a *bus region* connected to one or two memory cells. Memory cell regions and the bus region operate at a different pace. Memory cell regions are typically more stable and require infrequent evaluation until their respective word enable mosfet is turned on. The current solution method (static regionization), however, computes all word mosfet states and reevaluates all nodes including those in the inactive memory cells. It forces memory cells to be evaluated at a rate different from their own natural activity rate. Even if a cutoff check is put in place before the new current and conductance of a mosfet are computed, just traversing the data structure of all the word enable mosfets is very costly.

To effectively and efficiently compute dynamic regions, we developed a technique, known as incremental selective dynamic regionization. Incremental means we only examine the currently evaluated regions and test them to determine if those regions should be split or merged with others. While this cannot guarantee 100% dynamic regionization, it avoids the otherwise prohibitively high cost of examining all regions. Selective means that only those large regions that we know *a priori* to have the sparse dynamic connectivity are selected for applying this incremental technique. A static recognizer is designed to screen and select large regions displaying certain known characteristics. Since these circuits are mostly known structures, recognition can be achieved with high accuracy. For new/exotic structures, new recognition procedures can be added. An important requirement for our recognizer was that if the recognizer makes a mistake by recognizing some unknown structures as bus type regions, no simulation accuracy should be compromised (whereas simulation efficiency being adversely affected is acceptable.)

3.2 Implementation

In this subsection, details of the algorithms used to achieve selective and incremental dynamic regionization are described. For this paper, we concentrate on discussing recognition of memory regions.

Based on the goal of selective and incremental dynamic regionization, we devised the following approach. Each big region displaying memory characteristics is divided into subregions called bus region (only one) and cell regions. Dynamic data structures are built for bus region node structures and big node (bus node) mosfet structures. All cell regions are connected to the bus region through some mosfets (connectors) and can be easily attached to or detached from the bus region by the attach and detach routines described later in this section. Connectors are considered as part of the cell regions for memory recognition applications. Every time an enable word mosfet is evaluated, we can compute and check if the mosfet is conducting and if the cell region should be/remain attached or detached. Once detached, cell regions can be simulated like other small regions. If attached, a cell region structure is merged with that of its bus region

and disappears from the simulation list. Simulation proceeds as if the bus region and the cell region were one.

When a cell region is to be merged with its bus region, special care is needed to insure that both share a common simulation time stamp. The reason is as follows. Decoupled regions each schedule at their own pace. Two regions (before merging) can have node voltages, mosfet currents, and conductances calculated with respect to different time stamps. Such values represent the circuit status at different times and thus cannot be used together without causing time warp errors. Scheduling the bus region to simulate up to the current simulation time of the cell region solves this problem.

We now outline each of the major procedures to achieve selective and incremental dynamic regionization.

I. Recognition:

find all regions larger than a pre-defined threshold; find nodes with large connectivity insides these regions, and mark them as big nodes.

for every large region R found above{

```
for every MOS device m connected to a big node,
if (removing m generates an isolated region
bounded only by MOS gate terminals, power
and/or ground wires)
then the isolated region is a cell region;
if (the number of cell regions > threshold)
then this is a memory region; setup dynamic data
structures to keep track of cell regions
}
```

More intelligence can be built into the recognizer. Since it is only evoked once, efficiency and complexity is not an issue. We can have a collection of recognition routines and each performs a more detailed pattern matching for a specific big region type.

II. Attach(region r):

set cell region r's flag attached to true; add nodes and mosfets in r back to its bus region;

III. Detach(region r):

set cell region r's flag attached to false; delete nodes and mosfets in r from the bus region;

A routine *to_be_attached?* is needed to check if a cell region should be/remain attached to the bus region. Conceptually, we need to check if the connector mosfets of a cell region are on or off (by checking if V_{gs} (= V_g - V_s) > V_{th}). If any of them is on, then the cell region should be/remain attached to the bus region. Otherwise, the cell region should be/remain detached from the bus region. However, this implies that whenever the source terminal of a connector changes voltage (V_s), we should reevaluate the connector. Unfortunately, in a memory structure there may be many connector sources connected to a bus node. This means that every time the bus node (bit line) changes, we have to reevaluate hundreds or thousands of connectors. This is a very costly operation and defeats the merit of dynamic regionization. To solve this problem, we use the more conservative check $V_g > V_{th}$. A limitation with this checking method is that there cannot be any floating capacitors in the big region. The reason is that floating capacitors connected to the bus may cause the bus to go below Vss at transient and thus invalidate the checking.

IV. to_be_attached(region r):

if gate voltage Vg of every connector in the cell region r is smaller than Vth then return false; else return true;

We can now describe the algorithm of EMU2D in Figure 4 in simplified pseudo code after region recognition.

```
loop until done {
 for (every event < region r, and time t> at current time t) {
  if (r is not a cell region) {
     estimate initial solution;
     apply iterative relaxation to find a solution;
  } else { /* r is a cell region*/ let r's bus region be rb;
     if (r's attached flag is true)
        evaluate rb instead;
     else {
        estimate initial solution;
        apply iterative relaxation to find a solution;
        if (to be attached?(r) is true) /*resynchronize*/
           evaluate rb;
    }
 }
  if (voltage change of node N in r > threshold)
     then evaluate N's fan-out regions recursively;
 for every region R evaluated above
   if (R is a detached cell region && to_be_attached?(R))
      attach R to its bus region;
   else if (R is an attached cell && -to_be_attached?(R))
      detach R from its bus region;
   compute current and conductance for every mosfet in R;
   predict next event time for R and post an event;
 advance to the next smallest simulation time ;
```

```
Figure 4. Simplified Pseudo Code for EMU2D
```

Note that there is no change of accuracy with dynamic regionization as the solution method and the effective simulation granularity are not changed.

4. Experiments and Results

EMU2D has been implemented in C and C++ under UNIX and is widely used. Data from some experiments, most of them real designs, are presented in Tables 1-3. Several examples are fairly large, as shown in Table 1. Emu2 could simulate the entire decoder (the largest in Table 1) for over 2100 vectors in about two weeks. (While two weeks is a long time, Emu2 is at least capable of delivering timing check before fabrication.) EMU2D greatly extends that capability as it reduces the simulation time to about 3 days and makes such simulation more affordable. It also shows that our recognizer geared toward recognizing memory structures has successfully avoided selecting large regions in mult16, which is a parallel multiplier. Table 2 shows the times for Emu2 and EMU2D respectively and the speedup. For mult16 circuit, since there are no dynamic regions at all, we actually see a reduction of speed by about 6%, indicating that the amount of overhead is very modest. For all the other cases, we see speedups of 3 to 7, depending on the amounts of memory in the circuits. Last column of Table 3 shows that the times spent on big regions have reduced by 80-90% in most cases, which indicates that this implementation is close to optimal (limits) in what we can achieve by selective dynamic regionization. Also in Table 3, the event ratios are the ratios of dynamic region events counts vs normal region events counts. The time ratios are the ratios of the estimated time spent for processing dynamic region events vs. the *estimated* time for processing normal region events. If we divide the time ratio by the event ratio, we can observe that the results stay within a tight range with the exception of example acm. Example acm is a content-addressable memory. While it is a memory dominant circuit and has a lot of cell regions, nearly one third of them are always connected to the bus at any given time because of its special functionality. The highly concurrent nature of this circuit makes its speedup modest.

5. Conclusions

A dynamic regionization method that uses knowledge about common circuit structures and behaviors to selectively and incrementally apply dynamic regionization is described and proved effective and efficient in this paper. While the simulation accuracy remains the same as that of a static regionization method, speedups between 3 to 7 have been observed. The techniques used in the prototype has become a standard in our local timing simulation tool.

Circuit	Node	Devices	Regions	Cell	Bus
				Regions	Regions
block	12874	19198	4672	5912	128
fsmem	37958	166223	4174	29376	192
vle	41243	124063	7119	12576	292
decoder	368165	956172	64671	197640	1205
mult16	2576	7109	415	0	0
acm	14539	41781	1172	12738	33

Table 1 Characteristics of Examples Used in the Experiments

Circuit	# of	period	EMU2	EMU2D	Speedup
	vectors	(ns)	(sec)	(sec)	
block	130	4.0	1491	443	3.36
fsmem	153	20.0	31631	4512	7.01
vle	5630	16.0	48.7hrs	11.2hrs	4.35
decoder	2109	22.2	336.5hrs	74.5hrs	4.51
mult16	9	75.0	165	176	0.94
acm	12	40.0	2712	922	2.94

Table 2. Performance Comparisons of the Examples

Circuit	Normal Region	Bus Region	Cell Region	Event Ratio	Time Ratio	Reduc- tion
block	6738	163	31	0.029	0.12	92.4%
fsmem	19026	5920	5062	0.58	1.67	87.3%
vle	399542	79735	14751	0.24	0.79	84.5%
acm	513	147	411	1.09	18.3	60.0%

Table 3. Event Ratios and Time Ratios of Dynamic Regions vs. Normal Regions

The first three columns are event counts in thousands.

References

- [1] Chawla, B., Gummel, H. and Kozak, P. "MOTIS an MOS timing simulator", IEEE Trans. Circuits and Systems, Vol 22, pp. 901-909, 1975.
- [2] Fan, S. et al. "MOTIS-C: A New Circuit Simulator for MOS LSI.", ISCAS, pp. 700-703, 1977.
- [3] Ackland, B. and Weste, N. "Functional Verification in an Interactive Symbolic IC Design Environment", 2nd. Caltech Conf. on VLSI, Jan. 1981, pp. 285-298.
- [4] Saleh, R., Kleckner, J. and Newton, R. "Iterated Timing Analysis in Splice1", ICCAD'83 Digest, pp. 139-140, Sept. 1983.
- [5] Chen, C. and Subramaniam, P. "The Second Generation MOTIS Timing Simulator -- An Efficient and Accurate Approach for General MOS Circuits", ISCAS,, pp. 538-542, 1984.
- [6] Lelarasmee, E. and Sangiovanni-Vincentelli, A. "RELAX: A New Circuit Simulator for Large Scale Mos Integrated Circuits", Proc. 19th DAC, pp. 682-690, 1982.
- [7] Vidigal, L., Nassif, S. and Director, S., "CINNAMON: Coupled Integration and Nodal Analysis of MOS Networks", Proc. 23rd DAC, pp. 179-185, 1986.
- [8] Odyrna, P. and Nassif, S. "The ADEPT Timing Simulator Algorithm", VLSI SYSTEMS DESIGN, pp. 24-34, March 1986.
- Bauer R. et al., "XPSim: A MOS VLSI Simulator", ICCAD'88, pp. 66-69, Aug. 1988.
- [9] Carelli, J. "An Improved Transistor Model for EMU and an Associated Parameter Extraction Tool: EMUFIT", 52156-851213-01TM.
- [10] Ackland. B. and Clark, R. "Event-EMU: An Event Driven Timing Simulator for MOS VLSI Circuits", ICCAD'89 Proceedings, pp. 80-83, Nov. 1989.