VLSI DESIGN
2001, Vol. 12, No. 2, pp. 187–203
Reprints available directly from the publisher
Photocopying permitted by license only

© 2001 OPA (Overseas Publishers Association) N.V.
Published by license under
the Gordon and Breach Science
Publishers imprint.

Accurate Power Estimation for Sequential CMOS Circuits Using Graph-based Methods

MIRIAM LEESER[a,*] and VALERIE OHM[b,†]

[a]*Dept. of Electrical and Computer Eng., Northeastern University, Boston, MA 02115;*
[b]*Sequence Design Inc., 469 El Camino Real, Santa Clara, CA 95050*

(Received 20 June 2000; In final form 3 August 2000)

We present a novel method for estimating the power of sequential CMOS circuits. Symbolic probabilistic power estimation with an enumerated state space is used to estimate the average power switched by the circuit. This approach is more accurate than simulation based methods. Automatic circuit partitioning and state space exploration provide improvements in run-time and storage requirements over existing approaches. Circuits are automatically partitioned to improve the execution time and to allow larger circuits to be processed. Spatial correlation is dealt with by minimizing the cutset between partitions which tends to keep areas of reconvergent fanout in the same partition. Circuit partitions can be recombined using our combinational estimation methods which allow the exploitation of knowledge of probabilities of the circuit inputs. We enumerate the state transition graph (STG) incrementally using state space exploration methods developed for formal verification. Portions of the STG are generated on an as-needed basis, and thrown away after they are processed. BDDs are used to compactly represent similar states. This saves significant space in the storage of the STG. Our results show that modeling the state space is imperative for accurate power estimation of sequential circuits, partitioning saves time, and incremental state space exploration saves storage space. This allows us to process larger circuits than would otherwise be possible.

Keywords: Power estimation; Low power design; Binary decision diagrams; Design automation

1. INTRODUCTION

We present an accurate way to estimate average power dissipation in sequential CMOS circuits. We use symbolic power estimation with an enumerated state space to estimate the average power dissipated by the circuit. This approach is more accurate than simulation based methods which are biased depending on their starting state or states.

*Corresponding author. e-mail: mel@ece.neu.edu
†e-mail: vohm@sequencedesign.com

Our approach differs from similar approaches in two important ways. We enumerate the state transition graph (STG) incrementally using the state space exploration method [2] developed for formal verification. This saves significant space in the storage of the STG. In addition, we use automatic circuit partitioning which speeds up the run-time for power estimation. Combining incremental state space exploration with partitioning allows us to accurately estimate larger sequential circuits than was previously possible.

Our approach is similar to cell-based estimation, which has previously been applied to combinational circuits [6]. It differs in that we partition circuits automatically. This results in a more accurate estimation because partitions are generated to provide the best size for power estimation and do not depend on the size of a gate or macrocell.

Our work is similar to other sequential probabilistic power estimation, but is more accurate, while addressing issues of efficiency. For example, Monteiro *et al.* [7] construct the complete STG and compute state probabilities using the Chapman-Kolmogorov equation. They restrict themselves to circuits with a smaller number of flip-flops than we can handle. Chou and Roy [3] build an extended STG structure that includes primary input vectors in the states. This extended STG is considerably larger than a normal STG, which greatly increases the required storage space and the computation for finding state probabilities. Other researchers [9, 10] accommodate larger STGs by unrolling the circuit a few cycles to approximate the correlation. Tsui *et al.* [10] use Chapman-Kolmogorov to compute probabilities, while Schneider *et al.* [9] use Boolean equations. Najm *et al.* [8] run a Monte Carlo logic simulation on the sequential circuit to collect the signal and transition probabilities of the next-state lines, which are then fed into their combinational estimator. Our method is more accurate than these because it builds the whole STG; it requires less storage because the STG is built incrementally.

In Section 2 we describe combinational power estimation, which is used for sequential estimation. In Section 3 we describe the techniques used for sequential estimation. Our sequential method builds a STG for a circuit using state space exploration, calculates transition probabilities and switched capacitance for each STG edge, determines state probabilities using Chapman-Kolmogorov, then estimates power averaged over the STG. In Section 4 we discuss ways to improve on the basic algorithms to make the approach more efficient in both time and space. Results are presented in Section 5.

2. COMBINATIONAL POWER ESTIMATION

We estimate average power dissipated in combinational circuits by deriving a *behavior graph* and then computing the average dissipated power of that graph. We use the behavior graph to find average power dissipated over any or all possible input vector combinations. Large circuits are partitioned and behavior graphs are formed for each subcircuit, and the power estimate is computed hierarchically. Spatial correlation is dealt with by minimizing the cutset between partitions which tends to keep areas of reconvergent fanout in the same partition.

2.1. Behavior Graph

In the behavior graph, each vertex represents the state of the circuit or partition when it is subjected to a particular input vector(s). This is represented as a circuit vector of Booleans which are the values on the gate outputs. Figure 1 shows a circuit, its truth table grouped by circuit vectors, and the corresponding behavior graph. Multiple input vectors that result in the same circuit vector are a *compatible set* of input vectors and are stored with the vertex. State 10 has a compatible input set of 001, 011 and 101, as all three of these input vectors produce the same logic values on circuit nodes d and e.

The edges in the behavior graph represent possible transitions from state to state. In a purely

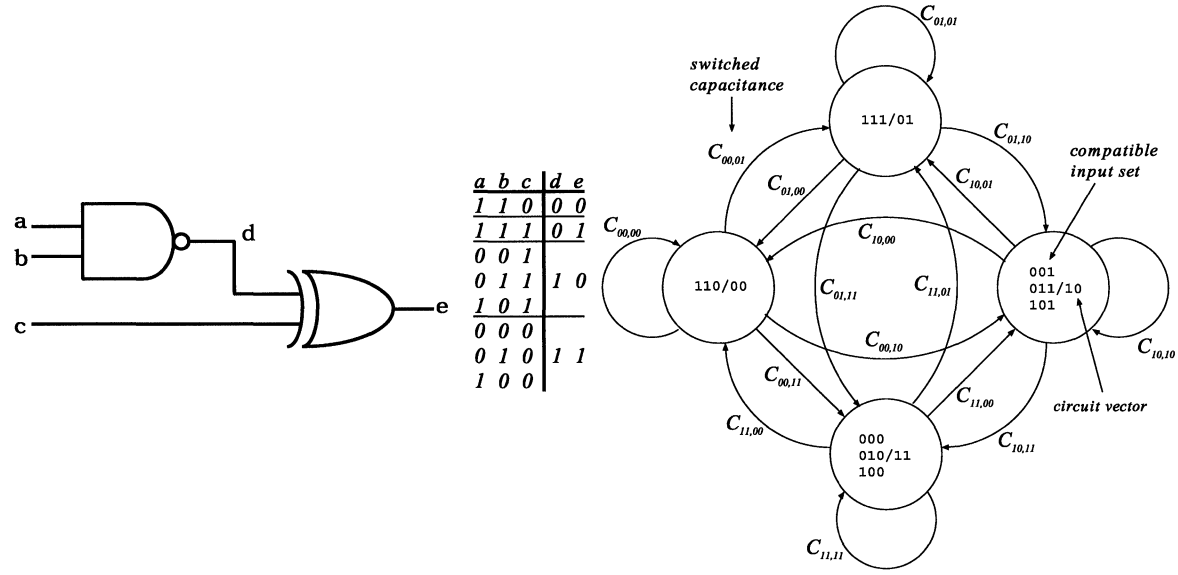


FIGURE 1 Behavior graph example.

combinational circuit, the behavior graph is completely connected. Between every state i and j (including the case $i=j$) there exists an edge, $e_{i,j}$. Associated with each $e_{i,j}$ is a capacitance $C_{i,j}$ that represents the capacitance switched in the traversal from state i to state j. $C_{i,j}$ is

$$C_{i,j} = \sum_{k \in N} (x_i(k) \oplus x_j(k)) C_g fanout_k. \qquad (1)$$

where N is the set of all gate outputs in the circuit, $x_i(k)$ is the logical value on gate output k in state i, C_g is the default gate capacitance, and $fanout_k$ is the fanout of gate k.

2.2. Partitioning

Power estimation based on behavior graphs has many advantages. The behavior graph contains all essential information for determining switching activity, and once a behavior graph has been generated, power estimation is fast and accurate. Behavior graph formation is accomplished by evaluating the state of all gates in the circuit for every possible input, and as a result, is exponential in the number of circuit inputs, which leads to long run times. We solve this problem by partitioning the circuit, then building behavior graphs for the subcircuits. This results in fast run times with a minimal loss of accuracy.

Our goals in partitioning are to minimize the cutset between partitions and to keep the partitions relatively balanced. Minimizing the cutset corresponds to minimizing the number of nets between partitions, which in turn corresponds to minimizing the amount of spatial correlation data lost. We hierarchically partition the circuit until all partitions are at or below a size that results in efficient power estimation.

Reconvergent fanout results when two or more inputs to a gate are functions of the same signal. Figure 2 shows a circuit with two incidences of reconvergent fanout. Minimizing the cutset tends to keep reconvergent portions of the circuit in the same partition, as they frequently manifest themselves as well-connected components. This helps to improve accuracy as highly correlated signals are kept in the same partition and weakly correlated signals are separated. Keeping the partitions balanced prevents the execution time from being dominated by one partition that is significantly larger than the others.

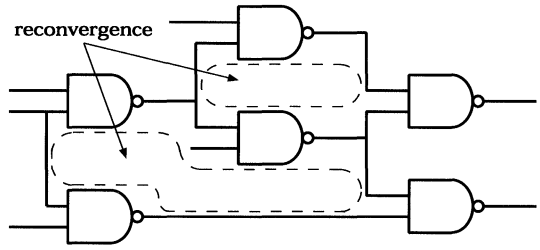


FIGURE 2 Circuit with reconvergent fanout.

To achieve these goals we partition the circuit using the *ratio-cut* algorithm [11]. Ratio-cut is a modification of the Kernighan-Lin algorithm [4] for variable-sized partitions. Rather than minimizing the cutset while perfectly bisecting the graph, it seeks to minimize

$$R(AB) = \frac{C_{AB}}{|A| \times |B|} \tag{2}$$

where C_{AB} is the size of the cutset of the partitions and $|A|$ and $|B|$ are the numbers of nodes in graph partitions A and B, respectively. Where Kernighan-Lin requires the graph to be exactly bisected, ratio-cut allows the sizes of the partitions to be changed. This allows the algorithm to find a smaller cutset without producing severely unbalanced partitions.

2.3. Power Estimation

The average dissipated power in a circuit or subcircuit is computed from the corresponding behavior graph. To compute the average power dissipation of the circuit, we use:

$$P_{avg} = \frac{1}{2} V_{dd}^2 f_{clk} C_{avg,total} \tag{3}$$

For each behavior graph we compute C_{avg}, the weighted sum of the capacitance switched over each edge in the behavior graph:

$$C_{avg} = \sum_{i \in V} \sum_{j \in V} p_i p_j C_{i,j} \tag{4}$$

Here V is the set of states in the behavior graph, $C_{i,j}$ is as described in Eq. (1), and p_i is the *state probability* of state i, as described in the next section. The C_{avg} values for all the behavior graphs are summed to get $C_{avg,total}$ which is the average capacitance switched over all possible input vector pairs.

2.4. State Probabilities

The state probability of state i, p_i is simply the probability that the circuit will be in state i at any given time. The sum of all the state probabilities is equal to 1. The general equation for the state probability of state i is:

$$\begin{aligned} p_i &= \sum_{j \in I_i} \prod_{k=1}^{n_{inp}} a, \quad \text{where } a \\ &= \begin{cases} p(k) & \text{if } x_j(k) = 1 \\ 1 - p(k) & \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

I_i is the compatible set of input vectors for state i, n_{inp} is the number of primary inputs to the circuit or subcircuit, $p(k)$ is the *signal probability* of primary input k, and $x_j(k)$ is the logic value on primary input k in input vector j. The state probability of state i is the probability that the circuit (or subcircuit) will be in state i at any given time. The signal probability of gate k is the probability that the logic value on gate k is equal to 1. The signal probability of a primary input is the probability that the input is 1.

In the case where all values of $p(i)$ are 0.5 (all input vectors are equally probable), Eq. (5) reduces to

$$p_i = \frac{|I_i|}{|I_{tot}|} \tag{6}$$

where $|I_{tot}|$ is the number of possible input vectors. This will be the case for unpartitioned combinational circuits when we assume that all input values are equally probable. Our approach can also be used where the probability of different

inputs is known. This allows us to process circuits hierarchically.

2.5. Power Estimation of Partitioned Circuits

We evaluate power hierarchically, based on knowledge of the signal probabilities of the inputs to the behavior graph. We assume that the probabilities of the primary inputs of the circuit are provided. We use the default value of 0.5 for primary input probabilities if no knowledge of the environment of the circuit is given. C_{avg} is evaluated from the behavior graph of a circuit or subcircuit when the signal probabilities are available for all the primary inputs of the subcircuit. Some of the primary inputs of a subcircuit may be outputs of gates from other subcircuits. For example, the output of gate D in Figure 3(a) is the output of subcircuit $p2$ and an input to subcircuit $p1$. Note that D is duplicated in $p1$ as a primary input to that subcircuit.

The signal probabilities of subcircuit inputs are computed from the behavior graph of the subcircuit that contains the corresponding gate. The input probabilities of that behavior graph must have been already computed as well. The equation for the signal probability for gate output j is:

$$p(j) = \frac{1}{|V|} \sum_{i \in V} p_i \cdot x_i(j). \tag{7}$$

where $x_i(j)$ is the logic value of gate output j in state i and p_i is the state probability for state i described above. The signal probabilities are stored in a probability vector as they are computed.

The behavior graphs need to be processed in order of dependencies. To visualize this, we can form a directed graph of the subcircuits with edges representing dependencies. Figure 3(c) is such a representation of the circuit partitioning represented in Figures 3(a)–(b). As such a graph is acyclic, it is straightforward to process each behavior graph as the probability information required becomes available. In the case of a circuit like the one shown in Figures 3(d)–(f) that has

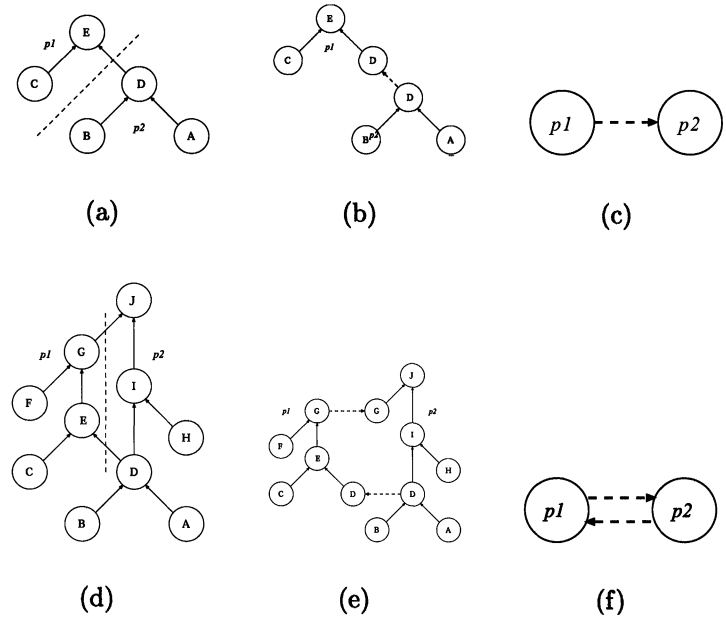


FIGURE 3 Partitioning examples (dotted arrows indicate dependencies).

cyclic dependencies, we need to do some additional work.

The following pseudocode shows how we can evaluate C_{avg} for all behavior graphs not involved in cyclic dependencies. We start by forming a list of all the behavior graphs called *bgraph_list*.

```
while BGRAPH_LIST decreasing in size
  for each BGRAPH in BGRAPH_LIST
    if all signal probabilities on
        inputs are available
      compute input probabilities
        for BGRAPH
    compute C_avg of BGRAPH and add to
      C_avg,total
    compute signal probabilities
      for other signals and store
      in probability vector
    remove BGRAPH from BGRAPH_LIST
```

For a circuit with no cyclic dependencies, the above procedure is sufficient to compute $C_{avg,total}$ for the entire circuit. Otherwise, *bgraph_list* will contain all the behavior graphs of subcircuits within cycles. To find the remaining C_{avg} values, we guess the unavailable signal probabilities (starting with a value of 0.5) and looping over all the behavior graphs until the signal probabilities converge. Convergence is achieved when all the elements of the probability vector vary by less than 0.01% in successive iterations.

```
    guess 0.5 for all unavailable
      signal probabilities
  while probability vector has not
    converged
for each BGRAPH in BGRAPH_LIST
  compute input probabilities for
    BGRAPH
    compute signal probabilities
      for other signals

for each BGRAPH in BGRAPH_LIST
    compute C_avg of BGRAPH and add to
      C_avg,total
```

3. SEQUENTIAL POWER ESTIMATION

Sequential power estimation builds on combinational power estimation and adds models for latches and state transitions. Figure 4 shows our sequential circuit model. When we analyze the combinational portion of a sequential circuit, we treat the present state lines as if they were combinational primary inputs, and the next state lines as combinational outputs, and use combinational power estimation.

We use a STG to represent a circuit. Each vertex in a STG is a *state* and is associated with a binary code that corresponds to the values of the present state lines. Edges correspond to possible transitions from one state to another and are associated with one or more input vectors. The circuit vector of a sequential circuit depends on the input vector *and* the previous state. Figure 5 shows an example of a sequential circuit and its state transition diagram.

Sequential power estimation consists of the following steps: the states of the STG are enumerated, transition probabilities and switched capacitances are calculated for each STG edge, state probabilities are found, and power is averaged over the STG. In the rest of this section, we describe these steps in more detail.

3.1. Building the State Transition Graph

We build the STG using *state space exploration* [2], a form of breadth-first search. Instead of having the graph structure already available, we derive it incrementally using the *transition relation*, which is

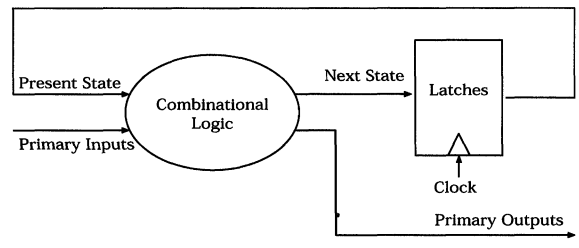


FIGURE 4 Sequential circuit model.

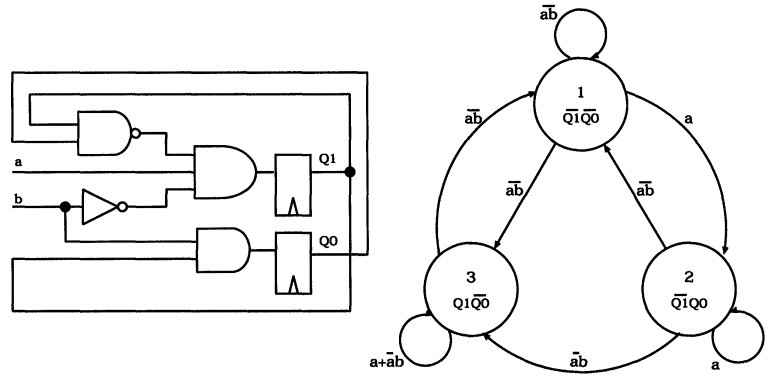


FIGURE 5 State transition diagram example.

derived from the circuit description. The transition relation is a list of *Binary Decision Diagrams* (BDDs), one for each next state variable, whose variables are the present state variables and the primary inputs. To find all of the possible next states, we evaluate each BDD in the transition relation with the current values of the present state variables. These partially evaluated BDDs are then in terms of the primary inputs to the circuit. From these, we derive the next states and the input vectors associated with them.

As edges in the STG are discovered, their transition probabilities can be computed from the signal probabilities of the primary inputs, which are either provided by the user or assumed to be 0.5. We store the transition probabilities in a matrix indexed by source and destination states. From the states of the edge and the input vectors, we construct a partial circuit vector with the present and next state values. The logical values on the other circuit nodes are unknown, so we record them as *don't cares*. The partial circuit vector and the compatible set of input vectors are stored with the edge for use later when we compute the switched capacitance.

3.2. Markov Processes and Steady-state Probabilities

We need to know the steady-state and transition probabilities to estimate the power. We compute the transition probabilities as we construct the STG. To find the steady-state probabilities, we use a technique from Markov process analysis called *limiting state probabilities*. Limiting state probabilities are derived from *n-step transition probabilities*. The 1-step transition probability, $p_{ij}(1)$ (or p_{ij}) is defined as the probability that the STG will transition from state i to state j in one time step, and is computed directly from the signal probabilities of the primary inputs. The n-step transition probability, $p_{ij}(n)$ is the probability that the STG will transition from state i to state j in n time steps. We compute it from a modified version of the Chapman-Kolmogorov equation: $p_{ij}(n) = \Sigma_k\, p_{ik}(n-1)p_{kj}$.

In the STGs of practical circuits, we observe that as n grows, $p_{ij}(n)$ becomes independent of both n and i and converges to the steady-state probability P_j, which is the probability that the STG will be in state j at any given time.

At this point, each edge in the STG is associated with one or more partial circuit vectors. Multiple circuit vectors for a single edge (i,j) occur when more than one primary input vector applied to the circuit in state i result in a transition to state j. The next step is to fill in the *don't cares* so we can compute the switched capacitance. We use our combinational circuit estimation method which represents each partition of the circuit as a separate behavior graph. The brute-force method for this is to intersect each partial sequential circuit

vector with every circuit vector in every behavior graph. We use a more efficient method called circuit vector recombination that is described in Section 4.1. Vectors whose intersections contain clashes between ones and zeros represent an invalid state in the circuit and are discarded. As partial sequential circuit vectors fully specify the primary circuit inputs and the present state lines, each partial sequential circuit vector will produce one fully defined circuit vector and corresponding input vector after the intersection operation.

3.3. Computing Switched Capacitance

Switched capacitance in an STG depends on edge-to-edge information. We associate a single capacitance with each edge (i,j) in the STG, which is a weighted average of all edge-to-edge transitions that originate at (i,j). Some edges may have multiple circuit vectors corresponding to multiple sets of gate outputs that represent the transition from state i to state j.

Figure 6 shows a portion of an STG where some edges have multiple circuit vectors. We define the switched capacitance between two circuit vectors:

$$C_{ij^m,jk^n} = C_g(\textbf{fanout} \cdot [\mathbf{v_{ij^m}} \oplus \mathbf{v_{jk^n}}]) \tag{8}$$

where $\mathbf{v_{ij^m}}$ is the mth circuit vector on edge (i,j), $\mathbf{v_{jk^n}}$ is the nth circuit vector on edge (j,k), and C_{ij^m,jk^n} is the capacitance switched when the state machine transitions from state i to j to k *via* those circuit vectors.

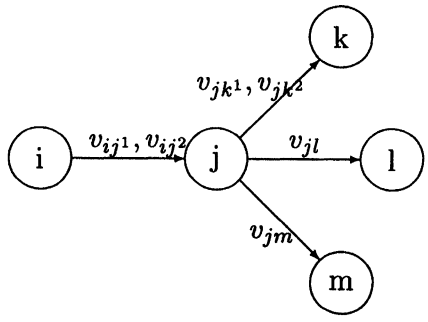


FIGURE 6 Capacitance computation example.

For the example in Figure 6 we compute the total average switched capacitance for edge (i,j) with:

$$\begin{aligned} C_{ij} = \frac{P_{ij^1}}{P_{ij}}(&P_{jk^1}C_{ij^1,jk^1} + P_{jk^2}C_{ij^1,jk^2} \\ &+ P_{jl}C_{ij^1,jl} + P_{jm}C_{ij^1,jm}) \\ + \frac{P_{ij^2}}{P_{ij}}(&P_{jk^1}C_{ij^2,jk^1} + P_{jk^2}C_{ij^2,jk^2} \\ &+ P_{jl}C_{ij^2,jl} + P_{jm}C_{ij^2,jm}) \end{aligned} \tag{9}$$

$$P_{ij} = P_{ij^1} + P_{ij^2} \tag{10}$$

We compute the total average switched capacitance for edge (i,j) with:

$$C_{ij} = \frac{1}{P_{ij}} \sum_s P_{ij^s} \sum_k \sum_t P_{jk^t} C_{ij^s,jk^t} \tag{11}$$

where

$$P_{ij} = \sum_n P_{ij^n} \tag{12}$$

is the total transition probability for the edge (i,j).

Along with the steady-state probabilities, the switched capacitance values are used to compute C_{avg}, the weighted sum of the switched capacitance switched over all edges in the STG:

$$C_{avg} = \sum_{i \in S} P_i \sum_{j \in S} p_{ij} C_{ij} \tag{13}$$

This accounts for the power dissipated due to combinational logic. We also must account for power dissipated by flip-flops. To do this we compute the power for each of the four possible transitions (0,0) (0,1) (1,0) (1,1) for a standard size flip-flop, and add them into the computation of the switched capacitance where appropriate.

4. IMPROVING SEQUENTIAL ESTIMATION

In this section we describe techniques to improve on sequential estimation as described in

Section 3. We use a more efficient technique to recombine circuit vectors. To improve on the efficiency in space and time of the power computation for sequential circuits, we use incremental state-space exploration to reduce the storage requirement, and circuit partitioning to reduce runtime.

4.1. Circuit Vector Recombination

Previously, we mentioned that filling in the don't cares of the partial sequential circuit vectors can be done by brute-force *circuit recombination*. This requires intersecting the partial sequential circuit vector associated with a STG edge with *every* circuit vector in *every* behavior graph, and is too expensive for large circuits. Some recombination is required for sequential power estimation to associate the sequential circuit vectors with the combinational ones. We use a tertiary search tree for this.

When we execute brute-force recombination of a partial sequential circuit vector with a behavior graph, we treat the behavior graph as if it were simply a list of circuit vectors with no regard to the graph structure itself. We start with the initial partial sequential circuit vector and conduct a trial intersection with each circuit vector in the behavior graph until we find a valid result. We replace the initial sequential circuit vector with the new result and intersect it until we find a another valid result. This process continues until all of the circuit vectors in the behavior graph are exhausted.

For real circuits, we find that the vast majority of circuit vector intersections do not yield a valid result. Our goal in formulating an algorithm to speed up this process involves substantially reducing the number of useless intersection operations without missing any of the necessary ones. We accomplish this by restructuring the list of behavior graph circuit vectors into a *tertiary search tree* such that we can quickly locate the circuit vectors that are useful and ignore most of the ones that are not.

4.1.1. Building the Tertiary Search Tree

Each circuit node represented in a circuit vector can take on one of three logical values: '0', '1' or 'X' (don't care). In a tertiary search tree, each level of the tree is associated with the index of the circuit vector. The root (the first level) of the tree represents the first circuit node, the second level the second circuit node and so on. Each *node* in the tree has three outgoing edges, each corresponding to one of the three logical values the circuit node for that level can take on.

Building the tertiary search tree for a set of circuit vectors is a recursive procedure. The procedure is passed the current list of circuit vectors and a vector index. The circuit vector list is divided into three lists – which list a circuit vector is added to depends on the logical value at the vector index. A new node is created and the three circuit vector lists are passed on to three more calls of the procedure with the index incremented. The new node is connected to those three results and the node is returned. A tertiary search tree built to the full depth of the length of the circuit vector will usually have far more terminal nodes than there are circuit vectors. We choose to terminate building a branch when the circuit vector list has only one vector left. We also avoid building empty terminal nodes by not recursing when the vector list is empty. Figure 7 shows the pseudo-code for this procedure and Figure 8 shows an example tertiary tree for a set of partial circuit vectors.

4.1.2. Recombination with Tertiary Search Trees

In the previous section, we described the construction of a tertiary search tree for a set of partial circuit vectors. Note that tertiary search trees are built very similarly to binary search trees: the *don't cares* are treated the same way as '0' and '1' values. When we search one of these trees, however, we consider the *don't cares* as a special case.

If we were to treat the *don't care* as a literal logical value, when we searched a tree with a new

```
node MAKE_TREE(bgraph BGraph)
1  return MAKE_TREENODE(VECTORLIST_FROM_BGRAPH(BGraph, 1)
node MAKE_TREENODE(list vectorlist, integer index)
1  list H, L, X
2  node parent = NEW_NODE(index)
3  if SIZE(vectorlist) == 1
4     parent.vectors = vectorlist
5     return parent
6  else
7     for each vec ∈ vectorlist
8        switch (vec[index])
9           case '1':
10             APPEND(H, vec)
11          case '0':
12             APPEND(L, vec)
13                case 'X':
14             APPEND(X, vec)
15    if (H) parent.high = MAKE_TREENODE(H, em index+1)
16    if (L) parent.low = MAKE_TREENODE(L, index+1)
17    if (X) parent.x = MAKE_TREENODE(X, index+1)
18    return parent
```

FIGURE 7 Pseudo-code for tertiary tree formation.

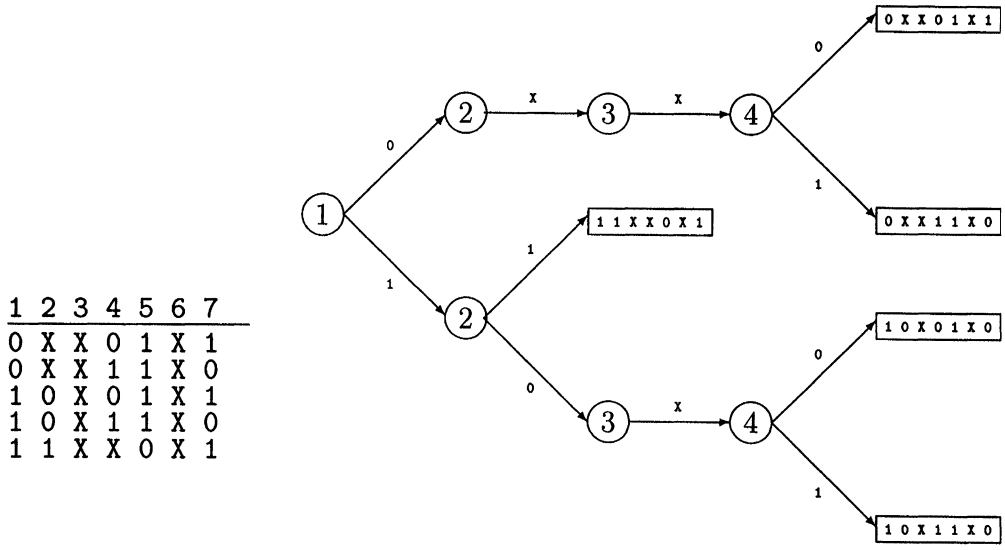


FIGURE 8 Example tertiary tree.

circuit vector, we would simply follow the path that corresponds to the values of that vector that returns one partial circuit vector when the terminal node is reached. Because the *don't care* can take on a '0' or '1' value, we need to follow more than one path to get all of the possible candidate vectors. In addition, a '0' or '1' in the new vector can also intersect with vectors in the 'X' branch of the tree.

```
list TRAVERSE_TREE(node current_node, vector vec, vectorlist result, int index)
  if (TERMINAL(current_node))
    return current_node.vectors
  switch (vec[index])
    case: '1':
      APPEND(result, TRAVERSE_TREE(current_node.high, vec, result, index+1))
      APPEND(result, TRAVERSE_TREE(current_node.x, vec, result, index+1))
    case: '0':
      APPEND(result, TRAVERSE_TREE(current_node.low, vec, result, index+1))
      APPEND(result, TRAVERSE_TREE(current_node.x, vec, result, index+1))
    case: 'X'
      APPEND(result, TRAVERSE_TREE(current_node.high, vec, result, index+1))
      APPEND(result, TRAVERSE_TREE(current_node.low, vec, result, index+1))
      APPEND(result, TRAVERSE_TREE(current_node.x, vec, result, index+1))
  return result
```

FIGURE 9 Pseudocode for searching a tertiary tree.

The search procedure will return a list of all of the vectors that *could* successfully intersect with the circuit vector in question. Figure 9 shows the pseudocode for the search procedure.

Though there is quite likely to be many more vectors in the result list than will successfully intersect, the size of this list is typically much less than the number that would be intersected by the brute-force method. Figure 10 shows the result of using tertiary search trees for several sequential circuits. The x-axis is the normalized partition size and the y-axis is the number of actual intersections performed divided by the number of intersections that would have been required using the brute force method. We find that for unpartitioned circuits, the number of intersections is reduced by over 80%. For reasonable numbers of partitions (up to four, in these examples) that there is still reasonable improvement.

4.2. Partitioning in Power Estimation

In combinational estimation of partitioned circuits, we avoided recombination due to the complexity. As a result, we also expect to lose a certain degree of accuracy by neglecting some correlation in the circuit. We experimented with estimating the power in sequential circuits without recombination to determine if there is a significant enough speed improvement that the cost in lost accuracy is acceptable. In unpartitioned estimation, with each edge we stored a list of pairs. Each pair consists of a circuit vector/input vector combination and the transition probability for each. When we include circuit partitions, we store a list of these pairs for *each* circuit partition. Some recombination is still required to combine the sequential circuit vector with each behavior graph. When we fill in the probability matrix we will already have an edge's total transition probability when we intersect it with the first behavior graph – all of the circuit inputs are already accounted for in the partial sequential circuit vector.

The switched capacitance computation is similar to the recombined case. We compute the switched capacitance for each partition and add each value into the capacitance matrix as they are computed. At this point, limiting state probability and total power dissipation computation are computed the same way as for unpartitioned and recombined circuits.

4.3. State-space Exploration

A third problem that must be addressed is the storage requirements of enumerating and saving the entire STG. While our method clearly requires the full enumeration of the STG, there is no

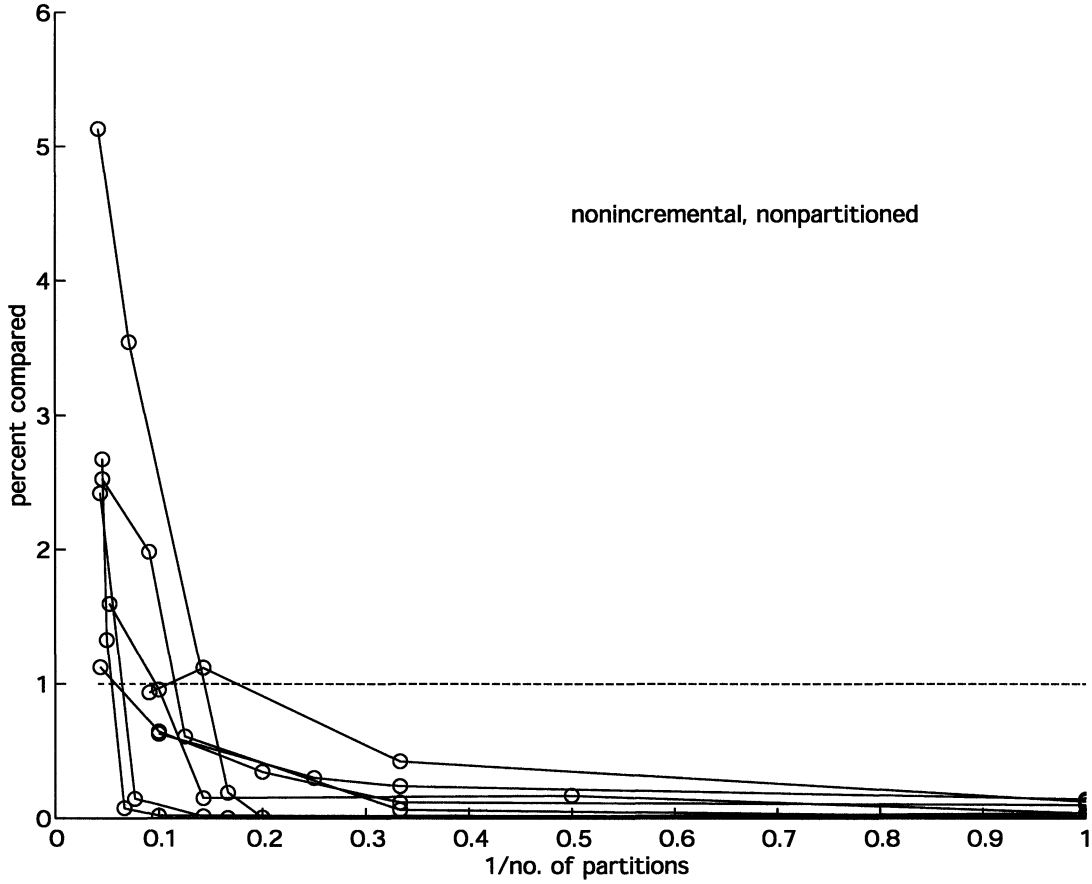


FIGURE 10 Circuit vector intersection improvement with tertiary search trees.

critical reason to have the entire STG available in memory for power estimation. Transition probabilities are computed directly from the STG edges—having other edges available in memory at the same time is not required. Computing the switched capacitance for an edge (i,j), however, requires both the edge (i,j) and all of js outgoing edges. Other than that, nothing else is needed. We therefore use a modification to state-space exploration called *incremental state space exploration* (ISSE). This method involves generating new edges and nodes in the STG as we need them, then discarding them when no longer required.

During ISSE we maintain three lists of states called *tiers*. The first tier is a list of one or more states in the STG. The second tier is a list of all of the next states from all of the states in the first tier. Note that the second tier may contain states that are also in the first tier, which would be the case if one or more states in the first tier has an outgoing edge that points back to itself. Similarly the third tier contains all of the subsequent states to the states in the second tier. Only three tiers of states are required to be held in memory at any given time, allowing us to discard unnecessary states and hence save storage.

In addition to the three tier lists, we also maintain a data structure in which we indicate which STG edges have already had their capacitances computed. This prevents the procedure from re-processing any states that have already been done. Each state in the STG also has a value

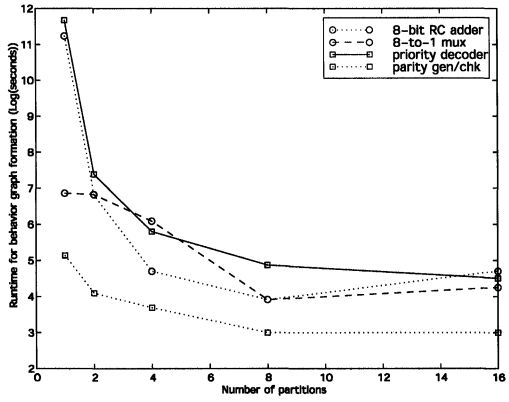
associated with it that indicates how many tier lists the state is currently in, which is used to determine which states are still needed and which are not.

5. RESULTS

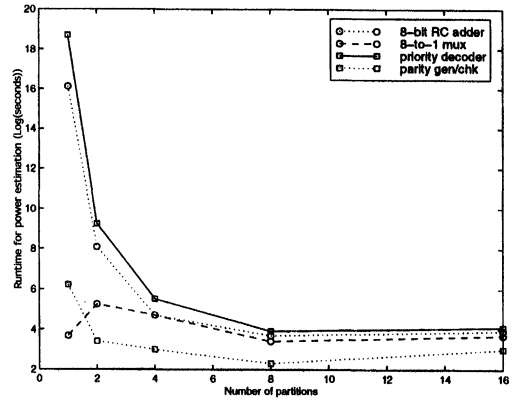
We ran a number of experiments on both combinational and sequential circuits to show the efficacy of our approach. These results of these examples are given in this section.

5.1. Combinational Results

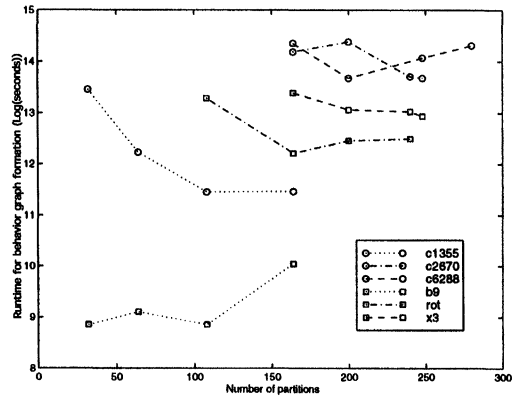
Our power modeling and estimation algorithm was implemented on a Sun Ultrasparc 1 in C. We ran trials on several circuits to evaluate the complexity of behavior graph formation with and without partitioning. We also measured runtimes for power estimation and the accuracy of power estimation in the presence of partitioning. In Figures 11(a) and (b) and Table I we show the execution times of the behavior graph



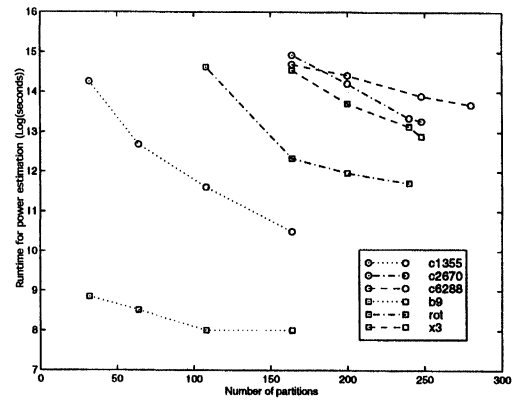
(a) Behavior Graph Formation (small circuits)



(b) Power Estimation (small circuits)



(c) Behavior Graph Formation (larger circuits)



(d) Power Estimation (larger circuits)

FIGURE 11 Program execution times (in ms).

TABLE I Program execution times (in ms)

Circuit	Description	Gates	1 part		2 parts		4 parts		8 parts		16 parts	
			bgr	*pwr*	*bgr*	*pwr*	*bgr*	*pwr*	*bgr*	*pwr*	*bgr*	*pwr*
adder8	8-bit RC adder	60	76E3	10E6	928	3318	115	123	55	42	125	5
fastadd4	4-bit lookahead adder	35	800	810	450	130	350	100	155	65	125	4
mux8l	8-to-1 mux	23	950	40	940	200	455	112	70	33	75	4
pridec	8-bit priority decoder	33	170	500	78	38	45	20	38	18	38	2
parity	8-bit parity generator	92	12E4	13E7	1657	10475	332	258	137	60	108	6

TABLE II Program execution times (bgraph formation/power estimation, in ms)

Partitions	ISCAS85 benchmarks			LGSynth89 benchmarks		
	c1355	c2670	c6288	b9	rot	x3
32	6940/15640	–	–	70/70	–	–
64	2040/3220	–	–	90/50	–	–
108	940/1100	–	–	70/30	5860/22380	–
164	950/400	14510/30400	17090/23980	230/30	2000/2270	6510/20730
200	–	17620/14830	8660/18200	–	2560/1570	4700/9000
240	–	8920/6300	–	–	2660/1220	4520/5100
248	–	8670/5830	12870/10920	–	–	4130/3970
280	–	–	16370/8750	–	–	–

formation (columns labeled *bgr*) and power estimation (columns labeled *pwr*) phases of our power estimator for some small circuits. Figures 11(c) and (d) and Table II show similar data for some larger circuits from the ISCAS85 [1] and LGSynth89 [5] benchmark sets.

Behavior graph formation is exponential in the number of inputs to the circuit or subcircuit. It is here that partitioning provides reduction in execution time. As shown in Figures 11(a) and (c), in some cases the execution time drops off exponentially as the number of partitions increases. This does not happen in all cases because the execution time is more closely correlated to the number of inputs in the subcircuit than the number of gates.

Partitioning also has a dramatic effect on the complexity of power estimation. Figures 11(b) and (d) show the substantial decreases in execution time from even minimal partitioning. In Table III we show average power estimates for the same circuits. In all cases we assume the signal probabilities of the primary inputs are all equal to 0.5 (all possible input vectors are equally likely), transistor gate area is 1.5 μm^2, gate capacitance per square micron is 1.7 fF and the clock period is 10 ns. Note that as a result of partitioning, inputs to partitions internal to the circuit might have different input probabilities, which are accounted for in the power estimation. The first column of results in the table shows the average power over all possible input vector combinations for an unpartitioned circuit, as calculated by our program. The following columns show the same calculation over partitioned circuits. Table IV shows some typical power estimates on the partitioned benchmark circuits from Figures 11(c) and (d). Figure 12 shows the partitioned estimates never deviate more than 3% from the estimates on unpartitioned circuits.

5.2. Sequential Results

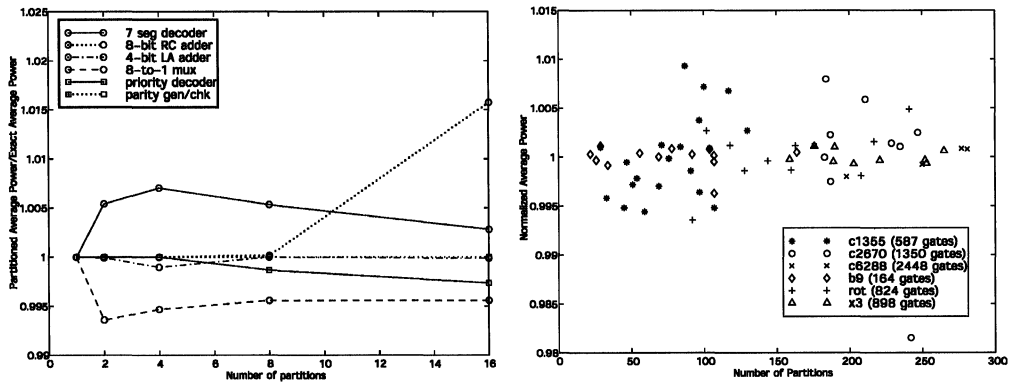
Table V compares average power dissipation estimates for sequential circuits. The line labeled *comb* corresponds to circuits run ignoring the

TABLE III Comparison of average power estimates (in μW) for small static CMOS circuits, $T_{clk} = 10$ ns, W = 1.5 μm, L = 1 μm, $C_g = 1.7$ fF/μm^2. (see also Fig. 12(a))

	Partitions				
Circuit	1	2	4	8	16
7seg	24.9	26.0	26.0	26.0	26.0
adder8	33.8	33.8	33.8	33.9	34.4
fastadd4	26.9	26.9	26.9	26.9	26.9
mux8l	10.3	10.2	10.2	10.2	10.2
pridec	15.1	15.1	15.1	15.1	15.1
parity	57.3	57.3	57.3	57.2	57.1

TABLE IV Comparison of average power estimates (in, μW) for larger static CMOS circuits, $T_{clk} = 10$ ns, W = 1.5 μm, L = 1 μm, $C_g = 1.7$ fF/μm^2. (see also Fig. 12(b))

	ISCAS85 benchmarks			LGSynth89 benchmarks		
Partitions	c1355	c2670	c6288	b9	rot	x3
32	419.59	–	–	104.99	–	–
64	420.11	–	–	105.08	–	–
108	419.17	–	–	105.16	569.40	–
164	422.99	1057.02	2520.03	105.13	568.54	622.58
200	–	1054.41	2512.15	–	569.69	621.69
240	–	1037.56	–	–	570.65	621.69
248	–	1059.69	2515.29	–	–	621.70
280	–	–	2515.29	–	–	–



(a) Small Circuits (b) Larger Circuits

FIGURE 12 Estimation accuracy for small and larger CMOS circuits.

effects of the sequential logic; the line labeled *seq* shows power estimates using state space exploration in order to take into account state space correlation. Clearly, incorporating sequential correlation is critical to accurate power estimates.

TABLE V Comparison of power estimates

circuit	gray5	cnt7	cnt8	s386
comb	39.25	14.16	16.31	140.34
seq	6.52	2.31	2.32	8.05

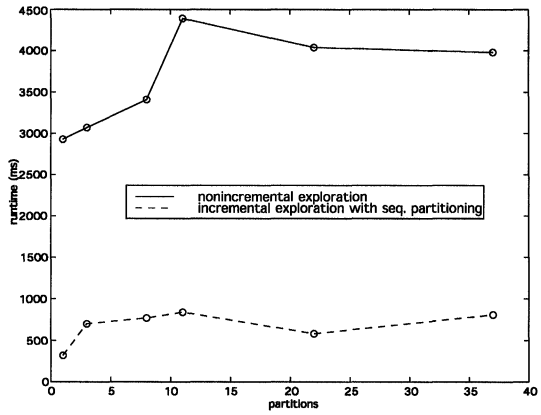


FIGURE 13 Runtime reduction with partitioning.

TABLE VI Memory savings from exploration

		% states in mem	
circuit	total states	avg	peak
cokemach	15	63	100
s386	13	65	85
gray5	32	13	16
cnt7	128	3.3	3.9
cnt8	256	1.6	1.9
s1488	48	20	44

Figure 13 shows the difference in run times with and without sequential partitioning for a counter. Sequential partitioning results in significant speed-up. Incremental state space exploration results in a small percentage of the total state space stored in memory for sequential machines, as shown in Table VI.

6. CONCLUSIONS

We have presented a symbolic, probabilistic technique for combinational and sequential circuit power estimation. Our approach is more accurate than simulation based approaches, and more efficient than many symbolic approaches. We incorporate automatic circuit partitioning, behavior graph recombination, and incremental state space exploration to efficiently and accurately estimate power in digital circuits.

Our combinational results show the circuit partitioning results in a large gain in performance with a very small loss in accuracy. Our sequential results show that modeling the state space is imperative for accurate power estimation of sequential circuits, partitioning saves time, and incremental state space exploration saves storage space. This allows us to process larger circuits than would otherwise be possible.

References

[1] Franc Brglez and Fujiwara, H. (1985). A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In: *IEEE International Symposium on Circuits and Systems*, pp. 695–698.

[2] Burch, Jerry R., Clarke, Edmund M., Long, David E., McMillan, Kenneth L. and Dill, David L., Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **13**(4), 401–424, April, 1994.

[3] Tan-Li Chou and Kaushik Roy (1995). Accurate estimation of power dissipation in CMOS sequential circuits. In: *IEEE International ASIC Conference and Exhibit*, pp. 285–288.

[4] Kernighan, B. W. and Lin, S., An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, **49**(2), 291–308, February, 1970.

[5] LGSynth89 Benchmark set, http://cbl.ncsu.edu/CBL_Docs/lgs89.html 1989.

[6] Jiing-Yuan Lin, Wen-Zen Shen and Jing-Yang Jou (1997). A power modeling and characterization method for macrocells using structure information. In: *IEEE/ACM International Conference on Computer-Aided Design*, pp. 502–506.

[7] José Monteiro, Srivinas Devadas and Bill Lin (1994). A methodology for efficient estimation of switching activity in sequential logic circuits. In: *ACM/IEEE Design Automation Conference*, pp. 12–17.

[8] Najm, Farid N., Shashank Goel and Hajj, Ibrahim N. (1995). Power estimation in sequential circuits. In: *ACM/IEEE Design Automation Conference*, pp. 635–640.

[9] Schneider, Peter H., Senn, Matthias A. and Bernd Wurth (1996). Power analysis for sequential circuits at logic level. In: *European Design Automation Conference*, pp. 22–27.

[10] Chi-Ying Tsui, José Monteiro, Massoud Pedram, Srinivas Devadas, Despain, Alvin M. and Bill Lin (1995). Power estimation methods for sequential logic circuits. In: *IEEE Transactions on VLSI*, pp. 404–415.

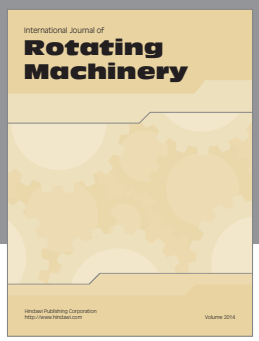
[11] Yen-Chuen Wei and Chung-Kuan Cheng, Ratio cut partitioning for hierarchical designs. *IEEE Transactions on Computer-Aided Design*, **10**(7), 911–921, July, 1991.

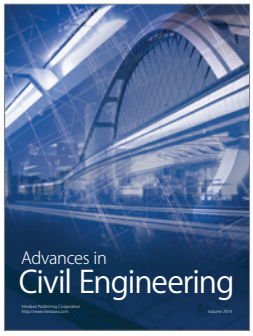
Authors' Biographies

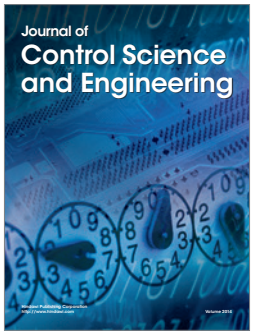
Miriam Leeser is an Associate Professor at Northeastern University, Department of Electrical and

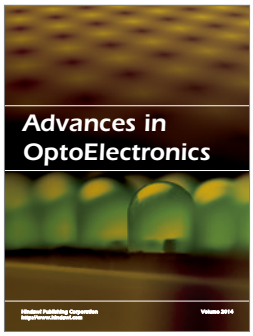
Computer Engineering. She received her BS degree in Electrical Engineering from Cornell University, and Diploma and Ph.D. Degrees in Computer Science from Cambridge University in England. After completion of her Ph.D., she joined Cornell University, Department of Electrical Engineering. In January, 1996 she joined the faculty of Northeastern University, where she is a member of the Center for Communications and Digital Signal Processing and the Computer Engineering group, and head of the Rapid Prototyping Laboratory. In 1992 she received an NSF Young Investigator Award. Her research interests include hardware description languages, high level synthesis, and reconfigurable computing for signal and image processing applications. She is a senior member of the IEEE, and a member of the ACM.

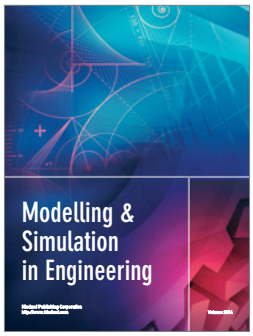
Valerie Ohm received her S.B. in Electrical Engineering from the Massachusetts Institute of Technology in May, 1993. She received a Master of Engineering degree in May, 1994 and a PhD in May, 1999, both in Electrical Engineering at Cornell University in Ithaca, NY. Since graduation she has been working at Sequence Design, Inc. Her research interests include design tools for low power, and computer-aided design tools for VLSI circuits.

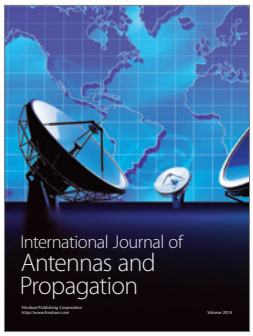


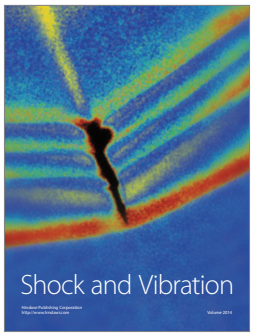


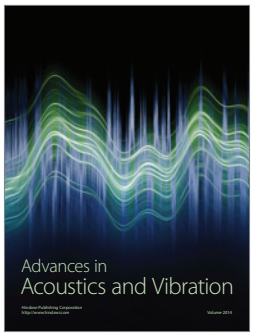












Hindawi

Submit your manuscripts at
http://www.hindawi.com