Improving the Accuracy of Circuit Activity Measurement

Bhanu Kapoor
Integrated Systems Laboratory, Texas Instruments, Dallas, TX 75243

Abstract

A novel measure of activity in digital circuits, called transition density, along with an efficient algorithm to compute the density at every circuit node, has been proposed in [1]. However, the efficiency of this algorithm is achieved at the cost of accuracy in the density values. This leaves much to be desired for its use in applications which require more accurate activity measurements at each node in the circuit e.g., circuit optimization problems with a low power goal.

The complexity of this problem lies in computing the Boolean difference probabilities at each node of the circuit. In this paper, an efficient algorithm for computing these probabilities is described. This allows the activity measurements, within a circuit partition, to be carried out in a more efficient manner compared to the well known approach of computing these probabilities.

Larger circuit partitions, where each node within a partition is solved accurately with respect to that partition, result in more accurate activity measurements. An efficient circuit partitioning algorithm, with the goal of maximizing the number of correlated nodes within each partition, has been developed. This allows more accurate measurements compared to a randomly selected set of partitions. These methods have been incorporated in an improved simulator for circuit activity measurement. Some results obtained on the ISCAS85 benchmark circuits are included.

I. INTRODUCTION

With the emergence of battery-operated applications that demand intensive computation in a portable environment, power analysis and optimization has become one of the most important tasks to be solved by computer-aided design tools. It has become a requirement that integrated circuits have limited power dissipation and an accurate simulation of power dissipation has therefore become highly desirable.

Power dissipation in integrated circuits is closely related to the choice of technology, circuit design, logic design, and the choice of architecture [5]. One of the major reasons for CMOS technology to become a major force in the current VLSI is that the power dissipation in CMOS circuits is significantly lower than in other technology circuits at comparable speeds. However, with the tremendous increase in the number of devices in VLSI and an ever-increasing number of portable applications requiring low power and high throughput, it has become increasingly more important to reduce the power dissipation.

Power dissipation in a CMOS circuit is directly related to the extent of switching activity of the internal nodes in the circuit. A direct and simple approach of estimating power is to simulate the circuit. Several *circuit simulation* based techniques have appeared in the literature [3][4]. Given the speed of circuit simulation, these techniques cannot be used to simulate large circuits for long-enough input vector sequences to get meaningful power estimates.

Recently, other approaches have been proposed [1][8] that require the user to specify typical behavior at the circuit inputs using *probabilities.* These may be called weakly pattern dependent. These techniques allow the user to cover a large set of possible input patterns with little effort. However, in order to achieve good accuracy, one must model the correlations between internal node values, which can be very expensive.

The use of symbolic simulation in order to produce a set of Boolean functions representing conditions for switching at each gate in the circuit has been proposed in [10]. A Monte Carlo simulation based technique has been proposed in [7]. The circuit is simulated for a large number of input vectors while gathering statistics on the average power. It is based on the approximation that the average power is distributed normally over a finite time.

In this paper, the transition density concept [1] is revisited. An excellent feature of this concept is that it allows measurements to be carried out in a pattern independent way. If some knowledge about the behavior of the input patterns is known then it can be taken into account by appropriate modification of the input probability values.

The complexity of this problem lies in computing the Boolean difference probabilities at each node of the circuit. In this paper, an efficient approach for computing these probabilities has been described. Larger circuit partitions, where each node within a partition is solved accurately with respect to that partition, result in more accurate activity measurements. An efficient circuit partitioning algorithm has been developed with the goal of maximizing the number of correlated nodes within each partition. These methods have been incorporated in an improved simulator for circuit activity measurement. Some results obtained on the ISCAS85 benchmark circuits are included.

II. TRANSITION DENSITY SIMULATION

A novel measure of activity in digital circuits called the *transition density* has been proposed [1] along with an algorithm to compute density at every circuit node. Transition density may be defined as the average switching rate at a circuit node. To briefly review the idea of transition density, consider the average power drawn by a CMOS gate. Let D be the transition density at the gate output, i.e., the average number of transitions per second. If the gate has output capacitance C to the ground, then the average power dissipated is given by:

$$P_{average} = 0.5 * C * {V_{dd}}^2 * D \qquad (1)$$

where V_{dd} is the power supply voltage. The power dissipation due to the through current during switching in CMOS [5] has been ignored.

It is possible to simulate the circuit for a large number of input transitions and find an approximate value for D. It is impossible to determine *a priori* how long the simulation should be carried on to get a reasonably good value as the number of possible input transitions grow exponentially with number of inputs. However, in [1], it is shown that if the transition density at the circuit primary inputs are given then they can be propagated into the circuit to give the transition density at every internal and output node.

The density propagation procedure works as follows: Let $P(x)$ denote the *equilibrium probability* of a logic signal $x(t)$, defined as:

$$P(x) \triangleq \lim_{T\to\infty} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x(t)dt \qquad (2)$$

This gives the fraction of time a signal is high. It has been shown [1] that, if $y = f(x_1, x_2, ..., x_n)$ is Boolean function and the inputs x_i's are independent, the density of output y is given by the following expression expression:

$$D(y) = \sum_{i=1}^{n} P\left(\frac{\partial y}{\partial x_i}\right) D(x_i) \qquad (3)$$

where $\frac{\partial y}{\partial x}$ is the *Boolean difference* of y with respect to x and is defined as:

$$\frac{\partial y}{\partial x} \triangleq y_{x=1} \oplus y_{x=0} \qquad (4)$$

where $\oplus$ is the logical exclusive-or operation. Given the probability and density values at the primary inputs of a logic circuit, a single pass over the circuit, using Equation 3, gives the density values at every node.

As an example, consider the simple case of an OR gate: $y = x_1 + x_2$.

$$\frac{\partial y}{\partial x_1} = x_2 \oplus 1 = \overline{x_2}, \frac{\partial y}{\partial x_2} = x_1 \oplus 1 = \overline{x_1}$$

$$D(y) = P(\overline{x_2})D(x_1) + P(\overline{x_1})D(x_2)$$

$$D(y) = D(x_1) + D(x_2) - P(x_2)D(x_1) - P(x_1)D(x_2)$$

In more complex Boolean functions, Ordered Binary Decision Diagrams (OBDD) [2] can be used as an efficient tool to carry out the probability computation [1].

III. INACCURACIES IN DENSITY SIMULATION

The algorithm suggested in [1] uses the lowest level partitioning of a circuit to propagate the transition density values. Each gate in the circuit forms a partition. Due to the correlation amongst various signals in a circuit, a result of reconvergent fanout structures in the circuit, such a method can either underestimate or overestimate the density value at a node. This can be illustrated using two simple examples:

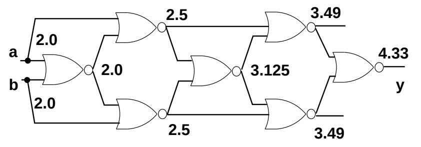

Figure 1: A case of overestimation

Fig. 1 shows an interconnection of NOR gates illustrating a case of overestimation. The primary inputs are assigned a transition density value of 2.0 eps (events per second) and a equilibrium probability of 0.5. The density value at node y, computed using the lowest level partitions, is 4.33 eps.

Equation 3 suggest an upper bound on the value of transition density at any node, given by:

$$D_{max}(y) \leq \sum_{i=1}^{n} D(x_i) \qquad (5)$$

Thus, the upper bound on the value of density at node y is 4.0 eps. However, the correct value of density at node y is 2.0 eps. Even if the algorithm ensures that $D(y) \leq D_{max}(y)$, the transition density value at node y is overestimated by 100%.

Fig. 2 shows a situation where the density value is underestimated at node y. The computed density value is 1.0 eps, and the correct value is 1.5 eps. In this case, the density value at node y has been underestimated by approximately 33%.

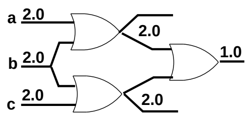

Figure 2: A case of underestimation

The errors in estimation can get worse with increasing depth of a circuit. If the goal of computation is to estimate the average value then the errors due to overestimation and underestimation may cancel each other and the average value may appear reasonably accurate. However, this poses a severe problem in using such a tool in solving certain circuit optimization problems where a reasonably accurate estimation of activity is desired at each node. An example of fanout manipulation, to reduce the power dissipation of a circuit under a delay constraint, explains this point clearly:

3.1 Role of Accuracy in Circuit Optimization

The average power dissipation of a circuit is given by:

$$P_{average} = \frac{1}{2} {V_{dd}}^2 \sum_{i=1}^{n} C_i D_i$$

where C_i is the capacitance and D_i is the transition density at node i in a given circuit. In this discussion, we assume that the capacitance at a node has a linear dependence on the fanout of the node. Fig. 3 shows a general combinational network with two fanout nodes, i and j, such that the density estimation algorithm

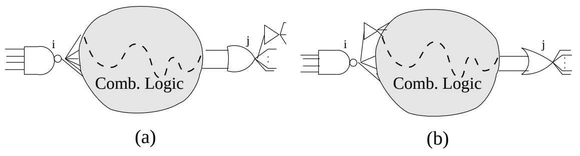

Figure 3: An example of power optimization

overestimates the density value at node i and underestimates the value at node j. Let the fanout at the two nodes be f_i.

Let the estimated transition densities be given by:

$$D_i(\text{est}) = D + \delta D,\ D_j(\text{est}) = D - \delta D$$

Let the true value of densities be given by:

$$D_i(\text{act}) = D - \delta D,\ D_j(\text{act}) = D + \delta D$$

Node i is overestimated and node j is underestimated, each by $2 * \delta D$. We have assumed equal amounts of error for the sake of simplicity. The total power dissipation P_T of the circuit is given by:

$$P_T(\text{est}) = P_T(\text{act}) = P_r + 2 * \alpha * D * f_i$$

where P_r is the power dissipation of the rest of the circuit and α is a constant having the dimensions of $C * V_{dd}^2$. It should be noted here that the total and the average power dissipation of the circuit remains unaffected despite the error.

Let us assume that node i and node j appear on the critical path this circuit. Consider an algorithm, optimizing power under delay constraint, decides to move the buffer from node j to node i. This decreases the fanout of node i and increases the fanout of node j. Let the change in fanout at both the nodes be δf_i. The delay is assumed to remain the same. The power dissipation estimate for the new circuit is given by:

$$P_T(\text{est}) = \alpha(D+\delta D)(f_i - \delta f_i) + \alpha(D - \delta D)(f_i + \delta f_i) + P_r$$

It is assumed that the same buffer has been moved to the new position, and there is no change in the power dissipation due to this exchange. The estimated power shows a decrease of $2 * \delta D * \delta f_i$.

$$P_T(\text{est}) = 2 * \alpha * D * f_i - 2 * \alpha * \delta D * \delta f_i + P_r$$

However, consider the change in the actual power of the circuit:

$$P_T(\text{act}) = \alpha(D-\delta D)(f_i - \delta f_i) + \alpha(D + \delta D)(f_i + \delta f_i) + P_r$$

$$P_T(\text{act}) = 2 * \alpha * D * f_i + 2 * \alpha * \delta D * \delta f_i + P_r$$

The transformation which was supposed to decrease the power dissipation of the circuit by an amount $2 * \alpha * \delta D * \delta f_i$, ended up increasing the power dissipation by the same amount. The error is proportional to the change in fanout. A power optimization tool, working with such inaccuracies, can increase the power dissipation of the circuit rather than decreasing it. This simple example suggests a need for more accurate computation.

An efficient method for computing the Boolean difference probabilities is described next.

IV. BOOLEAN DIFFERENCE PROBABILITIES

A simple method to compute the Boolean difference probabilities is based on Equation 4. However, when we deal with larger functions, this approach becomes rather inefficient. At every node in a given circuit, one must compute the OBDD of the node, use two RESTRICT operations [2] for each x_i to compute $y_{x_i=1}$ and $y_{x_i=0}$ and an APPLY operation [2] for each x_i to compute $y_{x_i=1} \oplus y_{x_i=0}$. This may be viewed as a top-down method because first the OBDD at a node must be created and then the OBDD can be used to compute the difference probabilities.

There is an inherent problem with this top-down approach of computing the difference probabilities. It does not make use of the work already done to compute the difference probabilities of the child nodes of a node being considered. The OBDDs required to compute the difference probabilities can be computed in a recursive fashion, just like the APPLY operation creates the OBDDs. It improves the efficiency of computation in producing maximally reduced OBDDs, on a need to compute basis. A new operation over OBDDs, called the DIFFERENCE operation, has been defined.

4.1 The DIFFERENCE Operation:

The DIFFERENCE operation generates Boolean difference functions by applying algebraic operations to other functions. Given argument functions $F(x)$ and $G(x)$, their Boolean difference with respect to x, and the binary Boolean operator $< op >$, (e.g., AND or OR), it returns the function $\frac{\partial(F(x) < op > G(x))}{\partial x}$. The DIFFERENCE operation is used to construct an OBDD representation of Boolean difference of each gate output, with respect to a variable it depends on, according to the gate operation and using the OBDDs created for its inputs. Each DIFFERENCE operation can be implemented as a sequence of APPLY operations.

Following properties of the Boolean difference operation are used to enable the recursive computation: $X = (x_1, x_2, ..., x_i, ..., x_n)$

$$\frac{\partial \overline{F(X)}}{\partial x_i} = \frac{\partial F(X)}{\partial x_i} \tag{6}$$

$$\frac{\partial (F(X)G(X))}{\partial x_i} = F(X)\frac{\partial G(X)}{\partial x_i} \oplus G(X)\frac{\partial F(X)}{\partial x_i} \oplus \frac{\partial F(X)}{\partial x_i}\frac{\partial G(X)}{\partial x_i} \tag{7}$$

$$\frac{\partial (F(X)+G(X))}{\partial x_i} = \overline{F(X)}\frac{\partial G(X)}{\partial x_i} \oplus \overline{G(X)}\frac{\partial F(X)}{\partial x_i} \oplus \frac{\partial F(X)}{\partial x_i}\frac{\partial G(X)}{\partial x_i} \tag{8}$$

$$\frac{\partial (F(X) \oplus G(X))}{\partial x_i} = \frac{\partial F(X)}{\partial x_i} \oplus \frac{\partial G(X)}{\partial x_i} \tag{9}$$

Equations 6-9 define the difference function for the outputs **NOT, AND, OR,** and **XOR** gates, respectively. The equations for **NAND, NOR,** and any complex gate can be defined in a similar fashion.

These equations point out some extremely useful properties of the DIFFERENCE operation, and can be used to enable more efficient computation. These equations allow the required OBDDs to be constructed in a bottom-up manner. This implies that the OBDD for a node in the circuit can be created using the maximally reduced OBDDs which have already been computed. Also, only the minimum number of required OBDDs need to be created. For example, OBDDs for the nodes which only feed to inverters and XOR gates need not be created at all. The output node of an inverter requires no computation at all if the input node has already been computed. Also, the OBDDs for the primary outputs of a circuit need not be created at all.

Each DIFFERENCE operation can be represented as a sequence of APPLY operations. For example, Equation 9 can be interpreted as follows:

$$\text{DIFFERENCE}((F(X) \text{ XOR } G(X)), x_i) = \text{APPLY}(\frac{\partial F(X)}{\partial x_i} \text{ XOR } \frac{\partial G(X)}{\partial x_i})$$

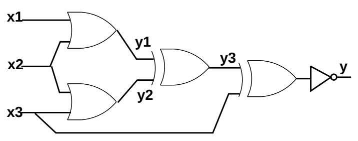

Figure 4: An example combinational circuit

Fig. 4 shows an example illustrating the efficiency of the DIFFERENCE operation. For the circuit shown in Fig. 4, all the OBDDs needed to compute the density values using the DIFFERENCE operation are shown in Fig. 5.

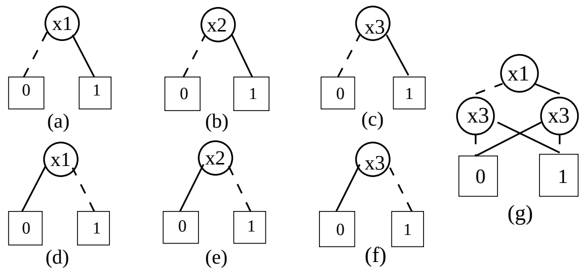

Figure 5: All the OBDDs created for the circuit in Fig. 4 using the DIFFERENCE operation

However, if the simple method, using Equation 4, is used then the OBDDs for all the intermediate and output nodes must be created. On these OBDDs, a total of 26 RESTRICT and 13 APPLY operations will be required to compute all the difference probabilities. The OBDDs involved in computation are larger as well.

The efficiency of the DIFFERENCE operation allows larger partitions to be used in computing the difference probabilities. This brings us to the problem of partitioning a circuit to make the density computation more accurate. If partitions are formed so that each partition packs more correlated nodes then the accuracy of computation will be better than a randomly chosen set of partitions. The next section describes such a circuit partitioning algorithm.

V. CIRCUIT PARTITIONING

A combinational circuit is modeled as a directed acyclic graph, $G(V, E)$. Every edge in this graph is an ordered pair of distinct vertices. A directed edge $[v, w]$ *leaves* v and *enters* w. If v is a vertex in the graph then its *in-degree* is the number of edges $[u, v]$ and its *out-degree* is the number of edges $[v, w]$. Nodes of indegree 0 are the primary inputs to the circuit, and nodes of outdegree 0 are the primary outputs of the circuit. We refer to the sets $out(v) = \{[v, w] \in E\}$ and $in(v) = \{[u, v] \in E\}$, for each $v \in V$, as *out-adjacency list* and *in-adjacency list* of a vertex, respectively.

The *support* of a function f, denoted as $S(f)$, is the set of variables f explicitly depends on. $|S(f)|$ is the the cardinality of $S(f)$.

The function f is a *feasible function* if $|S(f)| \leq k$, where k is the maximum allowed number of inputs in a partition. The function f is an *infeasible function* if $|S(f)| > k$.

A node v in a Boolean network η is *minimally infeasible* if $|S(v)| > k$ and for every node w such that there exists an edge $[w, v]$, $|S(w)|$ is less than or equal to k.

5.1 Partitioning Algorithm:

The partitioning strategy is of extreme importance here. The accuracy increases as the size of the partition grows, however, the algorithm will slow down as larger OBDDs will be created. Also, the memory usage increases with the increasing size of partitions. For large circuits, if care is not taken to keep the size of the partitions small enough, rapid growth in the size of the OBDDs can make the computation run into memory overflow problems. The strategy for partitioning should be such that the partitions are kept small while maximizing the number of correlated nodes within each partition.

How should we determine the size of a partition? We know that the number of inputs is the most important factor in determining the size of an OBDD. This is because the size of an OBDD can grow exponentially with the respect to the number of inputs, for a given variable ordering. Also, if number of inputs is small, the size of the OBDD will remain manageable even if there are large number of gates in a partition. For this reason, the number of inputs has been chosen as the parameter to determine the partition size. The circuit is partitioned in such a way that each partition has a single output which depends on at most k variables. These variables are either the primary inputs or the outputs of the various partitions created by the algorithm. The partitioning algorithm works as follows:

First, the support set of every signal in the network is computed. At this stage, the support set of a signal consists of all the primary input signals in its cone of influence. The computation of support sets can be accomplished using the breadth-first search technique. The breadth-first search starts at every primary input node in turn, and adds that primary input to the support set of every node that can be reached during the search. This is a linear algorithm with respect to the number of nodes in the circuit.

These support sets are then used to find the set of minimally infeasible nodes in the digraph. This simply requires a single pass over the list of vertices while examining the size of the support sets of their immediate children. As some of the children nodes of the minimally infeasible nodes get selected to form partitions, a new set of minimally infeasible nodes gets created. This process continues until all the nodes in the circuit are made feasible.

Once the set of minimally infeasible nodes is found, the next step makes these nodes feasible. This is accomplished by creating a partition on one or more children of the node under consideration. At this point, a heuristic is used to decide which child should be partitioned first. This heuristic computes the cost of creating a partition and the node with the minimum cost is chosen first.

The cost of creating a partition is computed as follows: For each node in the in-adjacency list of the minimally infeasible node, the intersection of its support set with the support sets of all other nodes in the list is determined. The cost is the sum of the cardinality of the sets obtained as a result of various intersections. The cost of partitioning the child node v_i is given by:

$$C(v_i) = \sum_{j=1, j \neq i}^{n} |S(v_i) \cap S(v_j)|$$

An example of cost computation is shown in Fig. 6. Let the maximum partition size be 5, $k = 5$. Node y is a minimally feasible node. This node can be made feasible by creating a partition at either of its three child nodes ($y1$, $y2$, $y3$). The cost of creating partitions for the three nodes are 0, 2, and 2 respectively. Node $y1$ having the minimum cost is chosen to form a partition. In this example, the transition density values of all the nodes shown in the figure will remain accurate despite partitioning. Such will not be the case if either node $y2$ or node $y3$ is chosen to form the partition.

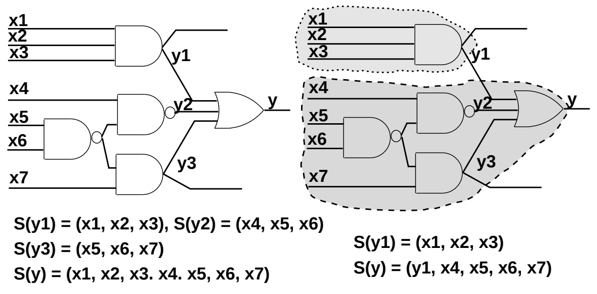
x1
x2
x3
y1
x4
x5
y2
y
x6
y3
x7
S(y1) = (x1, x2, x3), S(y2) = (x4, x5, x6)
S(y3) = (x5, x6, x7)
S(y) = (x1, x2, x3. x4. x5, x6, x7)
S(y1) = (x1, x2, x3)
S(y) = (y1, x4, x5, x6, x7)

Figure 6: An example of partitioning with support sets

Once a partition is created, the support sets of some of the nodes in the graph is updated. A breadth-first search based technique is used to accomplish this. Only some of the nodes reachable from the partitioned node go through the update process. Since the next iteration of computation needs only the minimally infeasible nodes created as a result of some of the partitions formed in the current iteration, the update stops at a node if it becomes an infeasible node. The updating process also stops at a node if it has already been partitioned before. This is because the prior partitioning at that node has isolated node's support set from the support set of all nodes reachable from that node.

The circuit partitioning procedure stops when there are no minimally infeasible nodes left in the graph. The circuit is now divided into partitions such that each partition depends on at most k variables. The overall complexity of the partitioning algorithm is linear for bounded fanout circuits.

5.2 Overall Algorithm Outline:

The partitioning algorithm, described in Section 5.1, works in stages. At each stage, the set of currently minimally infeasible nodes result in a set of partitions. These sets of partitions are stored in the order of their formation:

$$S_P = (S_{P_0}, S_{P_1}, ..., S_{P_{m-1}})$$

where S_{P_0} is the set of partitions whose input nodes are primary input nodes and $S_{P_{m-1}}$ is the final set of partitions. The computation of transition density values can proceed in the order of the elements in this set. This is because the nodes in the partition set S_{P_j} do not require the density values for any of the nodes in the partition sets $(S_{P_{j+1}}, ..., S_{P_{m-1}})$. Within each set S_{P_i}, the partitions can be computed in an arbitrary order. Within each partition, the computation of density values is carried out in a bottom-up fashion, as described in Section IV.

Since each partition, within a partition set ready for computation, can be considered in an arbitrary order, this suggests an easy adaptation of the algorithm to parallel methods. This is further aided by the fact that each partition depends on approximately k variables. The amount of work needed to compute the transition densities will be comparable for each partition. This will allow an uniform distribution of load on each processor.

VI. EXPERIMENTAL RESULTS

The algorithms described in this paper have been implemented in Common LISP running on a SPARC-Station 10. The program, called CAM, allows a user to do the trade-off between speed and accuracy based on the maximum partition size.

The efficiency of DIFFERENCE operation-based computation of Boolean difference probabilities allows fewer and smaller OBDDs to be created. Over a set of 50 small combinational examples, with number of primary inputs up to 20, this method required 40% fewer nodes than the straightforward method of computing Boolean difference probabilities. These gains, in general, are larger for larger examples. It also speeds up the computation. The OBDD tool used here did not have a facility for ordering variables, like [6]. However, the basic idea behind the circuit partitioning approach is to keep the number of inputs in each partition small enough and achieve reasonably accurate results which can be used in solving power optimization problems.

ISCAS85 benchmark combinational circuits have been used for experimentation. Table 1 shows the lower bound on percentage of nodes having 100% accurate value as a result of choosing a partition of a given size. The size of the partition is given in terms of maximum number of input variables for a partition. The second column contains the size (number of inputs) of the largest gate in these circuits. This determines the smallest partition size considered during the experimentation.

The numbers, in Table 1, give the lower bound on percentage of nodes with accurate value. In reality, more nodes can have accurate values. This is due to the fact that not every node is affected by reconvergent fan-out. As more and more nodes are computed accurately, the percentage of nodes with smaller inaccuracies increases as well. The numbers in Table 1 indicates that much more accurate results can be obtained, compared to the lowest level partition, by using fairly small partitions i.e., the partition size of k. This appears to be good in most cases, and is excellent in the case of c1908, c3540, c5315, and c7552. The improvements for c6288 and c499 are relatively poor. By using partitions such that each partition depends on less than 20 variables, true for the $k + 10$ case, a fairly high percentage of nodes can be precisely estimated.

Table 1: % NODES WITH 100% ACCURACY

		Partition Size			
ckt	k	llp	k	k + 5	k+10
c432	9	11.25	28.12	28.12	58.74
c499	5	19.80	27.72	35.64	59.40
c880	4	14.10	28.45	42.55	51.95
c1355	5	7.32	36.63	48.35	61.53
c1908	8	24.54	62.72	72.84	79.43
c2670	5	13.24	30.34	61.11	74.76
c3540	8	9.04	54.76	60.75	64.95
c5315	9	15.50	70.95	85.00	88.94
c6288	2	10.59	13.08	25.37	43.09
c7552	5	8.68	60.30	75.88	86.16

Table 2 contains the value of average transition density for these circuits, computed for various partition sizes. Each primary input was assigned a density value of 2.0 eps for all the benchmark examples. Column 1 contains the result of logic simulation using an average of 1000 transitions per input node [1].

For a circuit with n primary inputs, the number of distinct state transitions, where each state transition can have a different contribution to the power dissipation of the circuit, is $2^n * (2^n - 1)$. The number of primary inputs for these circuits range from 32(c6288) to 233(c2670). It is quite likely that only a small fraction of the possible distinct state-transitions were utilized during the simulation process. Hence, a comparison should be made with this fact in mind.

These results point out an interesting fact about the average transition density values using the lowest level partition. In some cases, like c7552, the average value obtained using the lowest level partition is closer to the value obtained using $k+10$ size partition than the value obtained using an intermediate partition size. This can be explained as follows: At the lowest level partition, density values for a large number of nodes in the circuit are being either underestimated or overestimated. Some of these errors may cancel each other when an average value is considered. The average value may look more accurate, there is no reason to believe that such is the case with values at the individual nodes in the circuit.

Table 2: AVERAGE DENSITY VALUES

		Partition Size			
ckt	sim	llp	k	k + 5	k+10
c432	3.39	3.46	3.19	3.19	3.00
c499	8.57	11.36	11.36	11.36	11.36
c880	3.25	2.78	3.48	3.40	3.39
c1355	6.18	4.19	6.87	6.87	6.87
c1908	5.01	2.97	4.38	4.90	5.16
c2670	4.00	3.50	3.45	3.52	3.56
c3540	4.49	4.47	4.62	4.62	4.83
c5315	4.79	3.52	3.66	3.88	3.89
c6288	34.2	25.10	23.34	21.84	20.88
c7552	5.08	3.85	4.14	3.84	3.72

Table 3 shows a comparison of the CPU times for these methods. The run-times, obtained on a SPARC-station 10, are given in seconds. Row 2 contains the run-times for the lowest level partition and Row 3 contains the run-times when the partition size is k. The run-times increase as as the size of the partition increase. However, the real problem with large partition comes from the memory usage. So far as the run-times are concerned, much larger OBDDs can be created very quickly using methods such as [9].

Table 3: CPU TIME COMPARISONS

ckt	c432	c499	c880	c1355	c1908
llp	0.28	0.33	0.61	0.85	1.24
k	21.1	12.6	9.7	13.8	15.6

Table 3: CONTINUED ...

ckt	c2670	c3540	c5315	c6288	c7552
llp	1.79	2.25	3.33	3.59	4.76
k	21.6	44.9	56.7	9.9	26.7

VII. SUMMARY AND FUTURE WORK

To summarize, we have described a new recursive approach for computing the Boolean difference probabilities using OBDDs. An efficient circuit partitioning algorithm, with the goal of maximizing the number correlated nodes within each partition, has been developed. This allows more accurate measurements compared to a randomly selected set of partitions.

The efficiency of this system can be further improved by using an efficient variable ordering algorithm for OBDDs. At present, this method is limited to combinational circuits only and does not consider glitches.

The algorithms presented here have been incorporated in an improved simulator for circuit activity measurement. Some results obtained on the ISCAS85 benchmark circuits establish the feasibility and efficiency of the approach.

Acknowledgment

The author would like to thank Dr. Brock Barton at Texas Instruments for suggesting very significant improvements to the work presented here. He is also thankful to Prof. Farid Najm at UIUC for answering several questions on the original work. Finally, thanks are due to Dr. Paul Stanford at Custom Computer Languages and Ms Aditi Dubey at The University of Michigan for reviewing the original manuscript.

References

[1] F. Najm, "Transition Density, A New Measure of Activity in Digital Circuits," in IEEE Trans. Computer-Aided Design, Feb. 1993, pp. 310-323.

[2] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," in IEEE Trans. Computers, Aug. 1986, vol. 35, pp. 677-691.

[3] S. M. Kang, "Accurate Simulation of Power Dissipation in VLSI Circuits," in IEEE J. of Soild-State Circuits, Oct. 1986, vol. 35, pp. 889-891.

[4] G. Y. Yacoub and W. H. Ku, "An Accurate Simulation Technique for Short-Circuit Power Dissipation Based on Current Component Isolation," in IEEE Int. Sym. Circuits and Systems 1989, vol. 35, pp. 1157-1161.

[5] A. Chandrakasan, T. Sheng, and R. W. Brodersen, "Low Power CMOS Digital Design," in IEEE J. of Solid-State Circuits, Apr. 1992, pp. 473-484.

[6] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment," in Proc. ICCAD, Nov. 1988, pp. 6-9.

[7] R. Burch, F. Najm, P. Yang, and T. Trick, "A Monte Carlo Approach for Power Estimation," in IEEE Trans. on VLSI Syst., Mar. 1993, pp. 63-71.

[8] M. A. Cirit, "Estimating Dynamic Power Consumption of CMOS circuits," in Proc. ICCAD, Nov. 1987, pp. 534-537.

[9] K. S. Brace, R. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," in 27th Design Auto. Conf., June 1990, pp. 40-45.

[10] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits," in 29th Design Auto. Conf., June 1992, pp. 253-259.