Computing the Maximum Power Cycles of a Sequential Circuit *

Srilatha Manne† Abelardo Pardo† R. Iris Bahar† Gary D. Hachtel† Fabio Somenzi† Enrico Macii‡ Massimo Poncino‡

† University of Colorado
Dept. of Electrical and Computer Engineering
Boulder, CO 80309

‡ Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

This paper studies the problem of estimating worst case power dissipation in a sequential circuit. We approach this problem by finding the maximum average weight cycles in a weighted directed graph. In order to handle practical sized examples, we use symbolic methods, based on Algebraic Decision Diagrams (ADDs), for computing the maximum average length cycles as well as the number of gate transitions in the circuit, which is necessary to construct the weighted directed graph.

1 Introduction

Designing integrated circuits for low power has become essential in today's electronics industry. Besides developing tools to automatically synthesize low-power devices, much effort is currently being spent in developing accurate power estimation tools to evaluate various circuit implementations. The easiest approach to power estimation is the one based on logic simulation. It is known that the average power dissipated by a CMOS circuit is directly related to the capacitive load of the network and to the number of voltage transitions at the output of the gates. This number, also called the switching activity factor of the network, can be computed by exhaustively simulating the circuit. This method requires the explicit enumeration of all the possible pairs of input vectors, which is impractical for devices of regular size and complexity. Taking computationally less expensive approaches is therefore needed; in particular, the use of symbolic techniques based on BDDs may be quite useful. Following this track, Devadas *et al.* [1] showed how to implicitly compute the pair (x^-, x) of consecutive primary input vectors which gives rise to the worst-case power dissipation, P, for a combinational circuit. They assumed that

$$\begin{aligned} P &= c \cdot V_{DD}^2 \cdot N(x^-, x) \\ N(x^-, x) &= \sum_g C_{Load}^g \cdot N^g(x^-, x), \end{aligned} \quad (1)$$

where $N^g(x^-, x)$ is the total number of transitions of gate g when primary input vector x^- is followed by vector x, and c is a constant scale factor. C_{Load}^g is determined by actual gate capacitance, fanout counts, and wire lengths.

This paper extends the above mentioned method to the case of sequential circuits by computing the maximum average power cycle, $\mathcal{C}_{Max}$, of the finite state machine associated with a given implementation of a sequential network. The average power dissipated in this cycle is the sum of the power dissipated in each transition, divided by the topological length of the cycle. This represents the maximum sustained power that the sequential circuit may dissipate.

Our approach features symbolic processing based on Algebraic Decision Diagrams (ADDs) [2, 3] throughout the computation. ADDs are used to compute and store the matrix $N(x^-, x)$ of the number of transitions in the whole circuit whenever input vectors x^- and x are applied consecutively. Furthermore, ADDs are used to represent the weighted edges of a dual graph G_D, derived from the state transition graph (STG) of the circuit, and used to determine the maximum power cycle.

Given directed STG $G = (S, E)$, the maximum power cycle computation can be shown as follows.

1. Compute the transition count functions $N^g(x^-, x)$ for all gates g in the circuit. Our method solves a problem quite similar to the one presented by Devadas *et al.* [1]. However, our procedure is novel in that it uses ADDs instead of timed boolean functions. Furthermore, the quantities computed are of more general interest.
2. Construct a dual graph $G_D = (S_D, E_D, (W_D, L_D))$ derived from G, which has a vertex $s \in S_D$ for each edge $(s, t) \in E$ of G. Each edge $(u, v) \in E_D$ is labeled with weight $W_D(u, v)$ equal to the power dissipation obtained from Equation 1, and length $L_D(u, v)$, initially set to 1.
3. Find the maximum average cycles of G_D. That is, the cycles of greatest average edge length using an exact algorithm [4] and iterative squaring [5].
4. Find a single maximum average cycle of G_D using a symbolic variant of the single-source shortest path technique given in [4].

The bulk of the paper will be devoted to symbolic ADD-based methods for computing transition counts (Section 2), constructing the dual graph (Section 3), and finding the maximum average cycles in this graph (Section 4). Section 5 is dedicated to symbolic processing considerations to improve execution times. Section 6 shows experimental results and Section 7 discusses conclusions and directions for future work.

2 Symbolic Transition Count Computation

The procedure to obtain the number of transitions at the output of each gate will be similar to the one proposed in [1] but completely symbolic by using ADD technology. After this phase, a matrix $N(r, x^-, s, x)$ is obtained representing the number of gate transitions produced in the circuit when first setting the state lines and the inputs to the vector (r, x^-), let the circuit reach a steady state and then apply the new vector

*This work is supported in part by NSF/DARPA grant MIP-9115432 and SRC contract 94-DJ-560.

32nd ACM/IEEE Design Automation Conference ®

(s, x). Following the approach taken in [6] on the computation of arrival time ADDs, we first constructed ADDs which represented all the transition times for each gate. Consider a combinational two-input circuit with gate g as the active gate in the process of computing the transition counts. Figure 1 shows an ADD fragment representing the transitions of gate g for primary input pairs $(10, 00)$ and $(10, 01)$. Given the delay of the gate, the output behavior is represented by a voltage waveform with an arbitrary number of transitions. Our strategy is to store the time interval between transitions as terminal values of the transition count ADDs. In the figure, the edges marked with circles denote the negative phase of the ADD node variable. Suppose UB is some upper bound on the number of possible transitions that may occur at any gate. We append $q = \log_2 UB$ new logic variables $h_1, \ldots, h_q$ to (x^-, x), to order the transitions of gate g. The bottom part of the ADD is just a binary representation of a linear list of numbers.

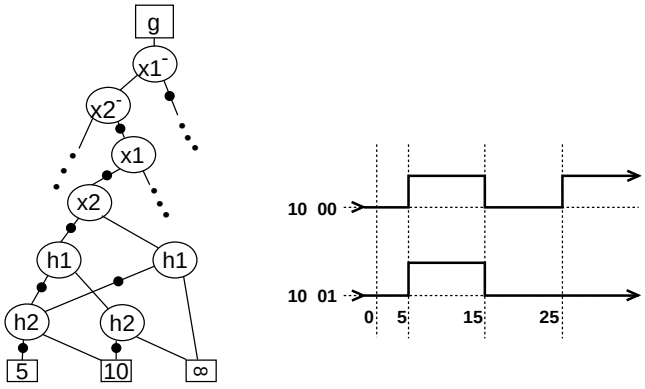


Figure 1: ADD Representation of Transition Times.

The top part of the ADD (in the figure encoded by the variables x_1^-, x_2^-, x_1, x_2) represents the previous and current input vector pair for the gate g, while the bottom part of the ADD (encoded by the variables h_1, h_2) represents the time intervals for the transitions in the voltage waveform seen at the output of the gate. The figure shows how recombination is more likely to occur when storing only the time difference between output transitions instead of the absolute times. In the current implementation a unit delay timing model is assumed, however, our method can easily be extended for non-unit delay.

The basic idea of the ADD-based symbolic transition simulation is as follows: We first construct BDDs giving the functional boolean values, $y_g(x)$, for each gate g. Next, we process each gate in topological order from primary inputs to outputs. For each gate g, we proceed by recursive descent through the ADDs for the transition times of the fanin gates f_j of gate g. For each f_j we extract a list of events represented by the bottom part of the ADD (variables $h_1, \ldots, h_q$). We then traverse these event lists and with the delay and the boolean equation of g determine the transition events at the output. Thus, an event list for g is obtained, and an ADD constructed for the event list. When we are finished using a given transition count ADD for simulating its fanout gates, the ADD may be deleted (after storing its minterm count). When the last gate has been processed, we have the transition count ADD for $N(r, x^-, s, x)$.

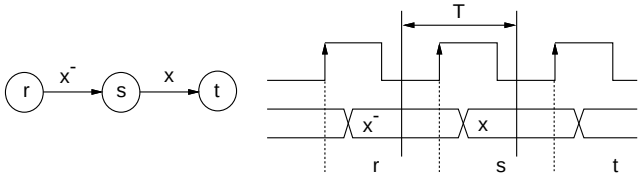


Figure 2: $N(r, x^-, s, x)$ Measures the Transitions in T.

This approach symbolically explores all possible combinations in the input and state signals of the circuit; however, in the case of sequential circuits, we need to account for the fact that not all states are reachable. More precisely once r and x^- are specified, s is determined by the transition relation of the given FSM. Figure 2 shows how each combination is composed of a state combination and a primary input combination. Our approach to account only for legal transitions is to carry along BDDs for $R(r)$, $R(s)$, and $T(r, x, s)$. Here $R(r)$ represents the set of reachable states, and $T(r, x, s)$ is the transition relation. As the recursive ADD cofactoring proceeds in the above algorithm for ADD processing, we cofactor the BDDs for $R(r)$, $R(s)$, and $T(r, x, s)$. When any of these three BDDs evaluates to 0, we no longer proceed in the computation since the set of input and state variables is not part of the behavior of the system.

3 Deriving the Dual Graph

Suppose we are given an STG $G = (S, E)$, with set of initial states S^0, and transition relation $T(r, x, s) = \{0, 1\}$ defined as $T(r, x, s) = 1$ if the circuit goes from state r to state s with input combination x, and suppose that only the states in the set denoted by $R(s) \subseteq S$ are reachable from the states in S^0. Suppose we are also given a transition count function $N(r, x^-, s, x)$, meaning that if we go from state r and inputs x^- to state s with inputs x, the circuit goes through $N(r, x^-, s, x)$ gate transitions. As previously stated, we seek to find the maximum average power cycle of the circuit, that is, the sequence of states/inputs that causes the greatest average number of transitions.

Toward this end we define a weighted dual graph $G_D = (V_D, E_D, (W_D, L_D))$, which has a vertex $v \in V_D$ for each edge $(r, x, s) \in E$ of G. (Note that s is uniquely determined by r and x, and therefore may be omitted in the description of a vertex.) There is an edge $(u, v) \in E_D$ if there are successive transitions $T(r, x^-, s)$ and $T(s, x, t)$ for some state t in G, where $u = (r, x^-)$ and $v = (s, x)$. Note that each edge (u, v) of G_D is uniquely associated with a transition count $W_D(u, v) = N(r, x^-, s, x)$, since each such edge has its own triple (r, x^-, x). It follows that the dual graph has the property that the path of maximum average length in G_D is in one-to-one correspondence with the maximum average power cycle in G. The the dual graph G_D is large: It has $2^q 2^i$ vertices and $2^q 2^{2i}$ edges, where q is the number of latches and i is the number of primary inputs. It is therefore advantageous to manipulate a symbolic, rather than explicit,

representation of G_D.

Given S^0, $T(r,x,s)$, $R(r)$, and $N(r,x^-,s,x)$, the dual graph can be built as follows. We first compute the set of all reachable vertices V_D:

$$V_D(u) = V_D(r,x^-) = R(r) \cdot \exists_s T(r,x^-,s). \tag{2}$$

The edges of the dual graph are labeled by the functions

$$\begin{array}{rcl} L^1_D(u,v) & = & R(r) \cdot T(r,x^-,s) \cdot V_D(s,x), \\ W^1_D(u,v) & = & N(r,x^-,s,x) \cdot L^1_D(r,x^-,s,x). \end{array} \tag{3}$$

(Multiplying by $V_D(s,x)$ in the computation of $L^1_D(u,v)$ is actually redundant, as long as the next state function is complete.) In practice we store a pair of constants W, L at each terminal node of the single ADD representing this pair of functions. However, it will be convenient to refer in the sequel to separate functions $W_D(u,v)$ and $L_D(u,v)$.

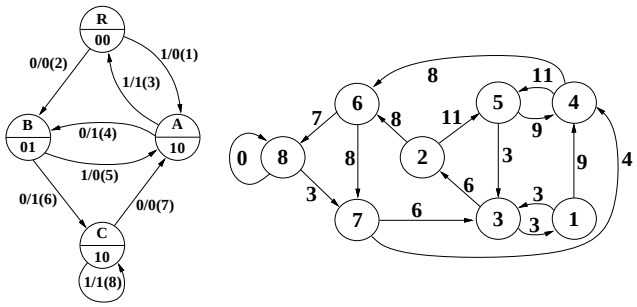


Figure 3: Construction of the Dual Graph.

Here the initial topological edge relation ADD $L^1_D(u,v)$ has just two terminal nodes α and 1, where $\alpha = -\infty$ is the appropriate idempotent background value for longest path computations [2]. Similarly, when $T(x,s,t) = 0$, we have $W^1_D(u,v) = \alpha$, else $W^1_D(u,v) = N(r,x^-,s,x)$.

Note that in computation, we convert the BDD $T(s,x,t)$ into an ADD, so the indicated multiplication represents a term-wise ADD apply operation [2]. Note also that $N(u,v) = 0$, always. Figure 3 depicts a STG and its correspondent dual graph. The labels in parentheses in the edges of the original STG correspond to the labels of the nodes in the dual graph. The labels in the edges of the dual graph represent the number of transitions in the circuit.

4 The Longest Average Length Cycle

The output of the dual graph calculation is a directed graph $G_D = (V_D, E_D, W_D)$, where $W_D(u,v)$ is the transition count weight associated with edge $(u,v) \in E$. If an edge doesn't exist between two nodes, the weight of the edge weight defaults to a background value of $\alpha = -\infty$. The length and the weight of a path between 2 vertices (u_1, u_i) is defined to be the sum of the lengths and weights of the edges traversed to reach u_i from u_1. In other words, for some path p from (u_1, u_i), which contains edges $e_1, e_2, \ldots, e_{(i-1)}$, $W(p) = \sum_{j=1}^{i-1} W_D(e_i)$ and $\textsc{avg}(p) = \frac{W(p)}{i-1}$. If a path p starts and ends at the same node, it is defined to be a cycle, and a cycle of maximum average weight is

$$C_{\mathrm{Max}} = \textsc{max}_{\mathcal{C} \in G_D} \textsc{avg}(W(\mathcal{C})), \tag{4}$$

for every cycle $\mathcal{C}$ in G_D.

The maximum average cycle, C_{Max} is the cycle in G_D which, over the lifetime of the machine, dissipates the most power. G_D is assumed to consist of a single, strongly connected component, although the algorithm below can be used for graphs with multiple strongly connected components. As stated in [4], each strongly connected component can be evaluated separately for C_{Max}, and the maximum of the different C_{Max} values is the true maximum average cycle. We now describe two approaches for computing or estimating C_{Max}.

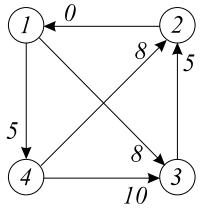
Karp's Algorithm. In [4], Karp proposed an algorithm for calculating the minimum average edge weight cycle in a weighted digraph. This algorithm is easily modified to calculate the maximum edge weight cycle to produce C_{Max}. The algorithm is simply stated as follows. Let s be an arbitrary source vertex, and let $F_k(v)$ be the weight of the longest weighted path from v_0 to v. We define $F_0(v) = \alpha, \quad \forall v \neq s$ and $F_0(s)$=0. Then we let

$$F_k(v) = \max_{(u,v) \in E} (F_{k-1}(u) + W_D(u,v)). \tag{5}$$

Karp has proved that

$$C_{\mathrm{Max}} = \max_{v \in V_D} \min_{0 \le k \le n-1} \frac{F_n(v) - F_k(v)}{n-k}, \tag{6}$$

where $n = |V|$. Once the maximizing v and k in (6) has been determined, an actual cycle yielding the maximum average weight can be determined by (1) finding a length-n maximum weight path $\pi(s,v)$ from s to v, and (2) extracting a terminal cycle of length $n-k$ occurring within $\pi(s,v)$. An example graph is given in Figure 4 The digraph at the left of the



	F_0	F_1	F_2	F_3	F_4
1	0	α	α	13	20
2	α	α	13	20	α
3	α	8	15	α	21
4	α	5	α	α	18

Figure 4: Maximum Average Length Cycle Computation.

figure has $n = |V| = 4$ vertices and $|E| = 6$ weighted edges, with $W_D(2,1) = 0$, corresponding to a state-transition in the original state diagram which causes no gate transitions. The column vector iterates $F_k(v)$, $v = 0, 1, \ldots, n$ are shown at the right. Equation 5 then gives

$$\begin{array}{rcl} C_{\mathrm{Max}} & = & \max_{v \in V_D} \min_{0 \le k \le n-1} \begin{bmatrix} 5 & \infty & \infty & 7 \\ 0 & 0 & -\infty & -\infty \\ \infty & \frac{13}{3} & 3 & \infty \\ \infty & \infty & \frac{13}{3} & \infty \end{bmatrix} \\ & = & \max_{v \in V_D} \begin{bmatrix} 5 \\ \infty \\ 3 \\ \frac{13}{3} \end{bmatrix} = 5 \end{array} \tag{7}$$

Here the row maximizer was vertex 1, and the minimizer in row 1 was $k = n = 4$. A longest path search from vertex 1 to vertex 4 identifies the MPC as the Hamiltonian cycle $C_{\text{Max}} = (1, 2, 3, 4, 1)$ with maximum average length 5, given by a path of weight 20 and topological length 4, denoted in the sequel as 20/4.

Because Karp's algorithm does $O(|E|)$ length update operations for each of the $|V|$ vertices in the graph, it has a worst case complexity of $O(|V| \times |E|)$. Unfortunately, although symbolic computations can overcome (in principle) the $O(E)$ factor through ADD recombination, $|V|$ iterations must be done to get an exact answer, so Karp's algorithm grows too expensive for combinatorically large graphs. Note there is a non-background value in row v, column k if and only if there is a path of length k and weight $F_k(v)$ from s to v. If n is large and other are many small cycles, we expect the path length matrix to be asymptotically full. Note also (row 3 in the example) that not all the row minimums correspond to actual cycles or even to actual paths. This adds some complication to obtaining a lower bound in the case in which n is too large to complete the whole algorithm.

Iterative Squaring. To get around the requirement of n iterations, an iterative squaring algorithm was developed. Although this algorithm has been already tried in several contexts by others, sometimes with limited success, it at least has the virtue of looking at long cycles quickly. In this algorithm we triangulate each edge by computing the longest average length 2-path in parallel to the edge, storing both the total weight and topological length of the 2-path. The longest average weight 2 path replaces the original edge if its average weight exceeds the weight of the original edge.

For the example of Figure 4, 2 steps of iterative squaring is ordinarily though to suffice, and we give the results for the two steps:

$$\frac{W_D^1}{L_D^1} = \begin{matrix} \alpha & \frac{13}{2} & 8 & 5 \\ 0 & \alpha & \frac{8}{2} & \frac{5}{2} \\ \frac{5}{2} & 5 & \alpha & \alpha \\ \frac{8}{2} & 8 & 10 & \alpha \end{matrix} \quad \frac{W_D^2}{L_D^2} = \begin{matrix} \frac{13}{3} & \frac{13}{2} & 8 & 5 \\ \frac{13}{4} & \frac{13}{3} & \frac{15}{3} & \frac{5}{2} \\ \frac{5}{2} & 5 & \frac{13}{3} & \frac{10}{3} \\ \frac{15}{3} & 8 & 10 & \frac{13}{3} \end{matrix} \tag{8}$$

It can be observed that after 2 steps of iterative squaring the inferior $(1, 3, 2, 1)$ cycle with average 13/3 is on the diagonal instead of the expected Hamiltonian cycle $20/5 > 13/3$. This disproves our former conjecture that after k steps of iterative squaring, $A_{u,v}^k = W_{u,v}^k / L_{u,v}^k$ gives the maximum average length of a path from u to v of topological length $L_{u,v}^k \leq 2^k$.

However it should be noted that if iterative squaring is continued beyond $\log_2(n)$ steps, the off-diagonal elements often continue to grow or shrink until they reach the value of the true MPC, because the paths from u to v continue to be "pumped" through the maximum average cycle. In the present case, we note that a third iteration will produce the correct answer of $20/4 = 5$ on each of the diagonal elements. We therefore continue the iterations until the diagonal elements reach a fixed point.

¿From a computational standpoint the iterative squaring process is analogous to matrix multiplication with weighted sum replacing the $\times$ operation, and max replacing the $+$ operation. Thus iterative squaring explores cycles of all possible lengths in $\lceil(\log_2 n)\rceil$ iterations, where n is the number of states in the reached set of the dual graph. Unfortunately, as the above example demonstrates, average path lengths aggregate in an unfavorable way. In fact,

$$(a/b) > (c/d) \not\Rightarrow (a+e)/(b+f) \geq (c+e)/(d+f). \tag{9}$$

In our example, we saw that although $(5 + 10)/(1 + 1) < (8/1)$, nevertheless $(20/3) = (5 + 10 + 5)/(1 + 1 + 1) > (8 + 5)/(1 + 1) = (13/2)$.

Another drawback of iterative squaring is that each iteration requires a matrix multiplication procedure which has a complexity of $O(n^3)$. Therefore, worst case running time, discounting recombination in the symbolic algorithm, for calculating C_{Max} is $O(\lceil(\log_2 n)\rceil \times n^3)$, compared to $O(n|E|) = O(n^3)$ for Karp's method.

Algorithm Comparison. Karp's algorithm has the unequivocal advantage of guaranteeing computation of the exact maximum average cycle for the dual graph specified. In contrast, as far as we know, iterative squaring only gives a lower bound.

However, iterative squaring has the advantage of a clearer interpretation of intermediate results, since we do not yet know quite what to do in cases where we cannot complete all n Karp steps. Further, iterative squaring has the relative advantage of early checking of long paths, and the backtracking algorithm for finding the actual longest average cycles is more straightforward. Further, although Karp's algorithm is cheaper per iteration by worst case complexity measure, the implementation of iterative squaring as matrix multiplication using symbolic algorithms, can be arbitrarily cheaper than n^3, for circuits which lead to high recombination in the ADDs for the dual graph.

5 Binning

The dual graph for many of the circuits has a large transition matrix of 2^{28} elements or more. Even with a gigabyte of memory, it cannot be represented using full matrix methods. The ADD representation of matrices, due to recombination, produces compact structures which require less memory but more processing time. Therefore, a two-tier binning method was developed to increase recombination and reduce execution times without significantly jeopardizing accuracy.

Recombination in ADDs occurs when there are adjacent terms in the matrix with the same value. The binning strategy attempts to increase recombination by reducing the number of unique constants in the matrix. The two-tier approach takes advantage of the fact that, in general, a maximum average power cycle contains large weighted edges. Therefore, our binning algorithm introduces a large error on small weighted edges and a small error on the larger weighted edges. For example, in Figure 5 all edge weights under the break point value (B_r) of 100 will be binned into values of 0, 50 or 100, depending on which bin has the smallest value exceeding the current edge weight. Subsequently, all edge weights greater than B_r will be allocated to bins of value 100, 105, 110 ... 135. E_M and E_m are the errors introduced in the MaxBin and the MinBin Range, respectively. For the example in Figure 5, E_M=49 and E_m=4. In the current implementation, binning is performed on the weighted, adjacency matrix used in Karp to produce the paths $F_k(v)$.

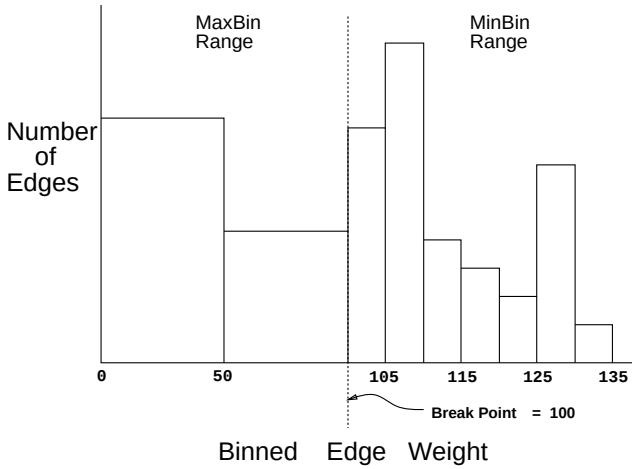


Figure 5: Two-Tier Binning Strategy.

The worst case error for the final, binned solution S to the C_{max} problem is as follows.

$$\mathrm{E}_S = \frac{n \cdot \mathrm{E}_m + (\mathrm{S}_l - n) \cdot \mathrm{E}_M}{\mathrm{S}_l}, \ \mathrm{n} = \left\lceil \frac{\mathrm{S}_w - \mathrm{S}_l \cdot \mathrm{B}_r}{\mathrm{M}_w - \mathrm{B}_r} \right\rceil, \quad (10)$$

where E_S is the error for the max power cycle and S_l and S_w are the topological length and weight of the C_{max} solution. n is the minimum number of edges in the MinBin Range, i.e., the number of edges with the minimum error. Ideally, n should be equal to S_l to minimize the error in the final solution.

Based on the E_S calculated using Equation 10, a range R_{max} is produced for the solution to the C_{max} calculation. It can be proven that the true Maximum Power Cycle C_{max} is within the range R_{max}. Since we are binning up the edge values in both ranges, the solution S produced by the binned C_{max} calculation is an upper bound on C_{max}. The Minimum bound on C_{max} is $\mathrm{S}_{min} = \mathrm{S} - \mathrm{E}_S$.

6 Experimental Results

The experimental results presented in this sections were obtained using a DEC 7000 Model 610 AXP with 1GB of memory. Table 1 shows the computational time spent in the logic simulation and power estimation of various sequential circuits. Column labeled *Gates* shows the implementation of the circuit in terms of unit delay gates. Execution time is in CPU seconds. The algorithm is able to handle any number of inputs per gate, as well as non-unit delay models. Label S_D is used for the column describing the number of states in the dual graph G_D. The column labeled *Depth* refers to the depth of the circuit. The maximum number of transitions in a certain gate is a function of circuit depth; hence this parameter is used to compute the number of ADD variables necessary to encode the list of transitions. Note that simulation time is not necessarily proportional to the number of gates. For example, the time required to run *s208* is significantly less than the time required to run *mm3*, while the number of gates is approximately the same. This is true even though *s208* has almost twice the number of primary inputs as *mm3*.

Ckt	*PI/FF*	*Gates*	*Depth*	*Time*	S_D
arbiter4	4/8	37	10	14.57	1024
arbiter6	6/12	57	16	628.72	24576
mm2	5/6	98	20	6.29	768
mm3	6/9	143	23	363.96	8192
ex1	9/5	462	8	594.43	10240
s208	11/8	100	16	48.91	34816
s27	4/3	12	5	0.39	96
s298	3/14	141	10	67.43	1744
s386	7/6	203	10	311.40	1664
dk16	2/5	458	8	18.02	108

Table 1: Computation of matrix $N(r, x^-, s, x)$.

Tables 2, 3, 4 show the results for exact, upper bound, and lower bound computations, respectively, for the circuits described in Table 1. For all the tables, W_m refers to the maximum edge weight in the the dual Graph G_D. This is the value expressed as the combinational solution to the MPC problem. Note that in all cases, the final solution is significantly smaller than the combinational solution. Therefore, the combinational solution provides an extremely pessimistic approximation to the MPC problem. In Table 4, the column labeled I_c/I_m refers to the number of iterations completed versus the maximum number of iterations possible using the iterative squaring algorithm. Hence, the solution produced by iterative squaring only considers cycles up to topological length 2^{I_c}. Finally, Table 3 provides a bounded solution to the C_{max} problem by using binning. B_m, B_M, and B_r refer to the minimum bin value, maximum bin value and break point. and E_S is the maximum error produced by binning. S_{min} and S are the lower and upper bound, respectively, for the final MPC solution. All $Time$ values are expressed in CPU seconds.

Table 2 provides exact solutions to some problems, but due to the large size of G_D and the complexity of Karp's algorithm, the size of the problems solved is relatively small. Binning reduces the execution time for the larger problems so that they may be completed given the memory resources. For example, s298 produces an exact solution in 1527 CPU seconds, while the binned version completed in 606 CPU seconds with an error of 18. Sometimes binning can introduce a very large error such as that produced for mm3. For these cases, the lower bound solution found using iterative squaring may be used to shrink the solution bounds. The lower bound found for mm3 is 149, which, when combined with the binned solution, produces a final, bounded solution range of [149, 253.33].

7 Conclusions

We have presented a new ADD-based algorithm for transition simulation. The number of transitions obtained are used to label a dual graph derived from the original STG of the circuit. We have presented also two symbolic algorithms to solve the longest average length cycle. One of them is the symbolic version of the algorithm proposed by Karp, and,

Name	$B_m/B_M/B_r/W_M$	*Sol*	E_S	S_{min} : S	*Time*
s27	1/5/15/21	30/2	4	11 : 15	2.65
dk16	5/50/150/295	1537/7	23.29	196.29 : 219.57	2.86
arbiter4	1/8/16/34	86/4	3.5	18 : 21.5	88.46
arbiter6	5/40/40/52	255/6	33.17	9.33 : 42.5	11129.03
mm3	20/200/200/379	760/3	139	114.33 : 253.33	4928.83
s298	5/25/50/114	690/10	18	51 : 69	606.75
s386	5/50/150/215	350/2	26.5	148.5 : 175	3192.18

Table 3: Upper Bound Results.

Name	W_M	*Sol*	*Time*
arbiter4	34	86/4=21.5	118.55
dk16	295	2346/11=213.27	3.12
mm2	175	274/3=91.33	143.59
s27	21	27/2=13.5	2.66
s298	114	632/10=63.2	1527.01

Table 2: Exact C_{max} Results.

Name	W_M	I_c/I_m	*Sol*	*Time*
s27	21	4/7	27/2=13.5	10.59
dk16	295	7/7	2346/11=213.27	108.11
s208	100	1/16	48/2=24	3093.44
s298	114	11/11	334/10=33.4	146.98
s386	215	2/11	248/2=124	1527.25
arbiter4	34	4/10	78/4=19.5	663.25
mm3	379	1/13	298/2=149	143.90

Table 4: Lower Bound Results.

when it can complete $n = |V|$ iterations, provides an exact solution to the problem. The second one, also symbolic, is based on an iterative squaring scheme and provides a sub-optimal lower bound on the true solution. We have explored the tradeoff between execution time and accuracy of the solution by means of a binning strategy in the values labeling the edges of the graph. Our experiments have succeeded in cases beyond the ability of conventional full-matrix or sparse matrix methods. For example, circuit s208 has a dual graph with 34816 vertices and more than 71 million edges. Since the computation needs two matrices to be stored at the same time, the memory requirements exceed 1 gigabyte. In this particular example, our algorithms were able to provide certain bounds for the solution.

Currently there are several lines of research that are being explored. In order to improve the memory requirements we are considering two options. On one hand we are measuring the effect of dynamic reordering of the variables in the ADDs in the memory requirements. Furthermore we have some preliminary experiments using Edge Value BDDs [7] that shows a reduction in the data size with respect to ADDs. With regard to time efficiency, we are exploring different ways to provide upper bounds in the computation. One approach consists of replacing all edge weights on edges leaving a given node with the largest of them. A second approach consists of partitioning the circuit as in [8, 9], which involves tearing some primary input or present variables.

Acknowledgements

We thank Robert Brayton and Dirk Grunwald for their support in the experiments and Randall Bryant for providing a counter example for the iterative squaring algorithm.

References

[1] S. Devadas, K. Keutzer, and J. White, "Estimation of power dissipation in CMOS combinational circuits using boolean function manipulation," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 373–383, Mar. 1992.

[2] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," in *Proceedings of the International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 188–191, Nov. 1993.

[3] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," *Journal of Formal Methods in Systems Design*, 1994. to appear.

[4] R. M. Karp, "A characterization of the minimum cycle mean of a digraph," *Discrete Mathematics*, vol. 23, pp. 309–311, 1978.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *An Introduction to Algorithms*. New York: McGraw-Hill, 1990.

[6] R. I. Bahar, H. Cho, G. D. Hachtel, E. Macii, and F. Somenzi, "Timing analysis of combinational circuits using ADD's," in *Proceedings of the European Conference on Design Automation*, (Paris, France), pp. 625–629, Feb. 1994.

[7] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula, "FGILP: An integer linear program solver based on function graphs," in *Proceedings of the International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 685–689, Nov. 1993.

[8] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, and F. Somenzi, "A state space decomposition algorithm for approximate FSM traversal," in *Proceedings of the European Conference on Design Automation*, (Paris, France), pp. 137–141, Feb. 1994.

[9] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, and F. Somenzi, "A structural approach to state space decomposition for approximate reachability analysis," in *Proceedings of the International Conference on Computer Design*, (Cambridge, MA), pp. 236–239, Oct. 1994.