A Symbolic Algorithm for Low Power Sequential Synthesis [*]

Balakrishna Kumthekar In-Ho Moon Fabio Somenzi

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

Abstract

We present an algorithm that restructures the state transition graph (STG) of a sequential circuit so as to reduce power dissipation. The STG is modified without changing the behavior of the circuit, by exploiting state equivalence. Rather than aiming primarily at reducing the number of states, our algorithm redirects transitions so that the new destination states are equivalent to the original ones, while the average activity of the circuit is decreased. The impact on area is also estimated to increase the accuracy of the power analysis. The STG and all other major data structures are stored as decision diagrams, and the algorithm does not enumerate explicitly the states or the transitions. (i.e., it is symbolic.) Therefore, it can deal with circuits that have millions of states. Once the STG has been restructured we apply symbolic factoring algorithms, based on Zero-suppressed BDDs, to convert the optimized graph into a multilevel circuit. We derive an efficient circuit from the BDDs of the STG by incorporating power constraints in the symbolic factoring algorithms.

1 Introduction

The progress in productivity for VLSI designers has resulted from advances in tools (both software and hardware) and methodology. Among the changes that have taken place over the last decade it is easy to point out the widespread use of Hardware Description Languages (HDLs) as one of the most significant. HDLs allow designers to work at a much higher level of abstraction than gate-level netlists or schematics, and therefore they require synthesis systems to translate behavioral specifications into structure. In this paper we study such translation when the behavior is given in the form of a state transition graph. The algorithms we propose in this paper are symbolic, relying on decision diagrams [5, 1, 12] for the manipulation of state transition graphs and circuits alike. We have developed a new graph restructuring algorithm that exploits the existence of equivalent states to decrease power dissipation, not necessarily by collapsing the equivalence classes, but by redirecting transitions in the graph. At the high level, we use a fairly abstract measure of power dissipation, namely the state bit transitions. As the refinement progresses, more detailed power models are brought into play.

[*]**Work supported by NSF/DARPA grant MIP-94–22268 and SRC contract 95-DJ-560.**

Another contribution of this paper is how the restructured state transition graph is translated into a circuit. For this task, traditional algorithm are quite inadequate. Translating a BDD that represents a finite state machine directly into a network of multiplexers (one for each BDD node) and then subjecting the network to algorithms like those in SIS [13] normally results in circuits that are both very large and very slow. We have extended Minato's symbolic synthesis algorithms in two ways: Improving the general efficiency, and incorporating power consumption considerations in the factoring algorithm. An accurate comparison of symbolic and traditional logic synthesis algorithms must include the observation that for small circuits that both kind of algorithms may handle, the traditional algorithms often produce better results. However, our work shows that the gap is closing.

In this paper we have omitted the basic definitions and notations regarding BDDs [5], ZDDs [12], finite state machines, Markov chains [7], and equivalent state computations [9] due to space constraints. The reader can consult the references for more information on these topics. The rest of the paper is organized as follows: Section 2 presents the restructuring algorithms. Section 3 covers the conversion of a state transition graph into a circuit. In Section 4 we discuss the symbolic synthesis algorithms. Experimental results are reported in Section 5 and conclusions and points to directions of future work are presented in Section 6.

2 Graph Restructuring

In this section we present a transformation that affects the structure of the state transition graph, without changing the behavior of the circuit. We illustrate here a simple example of such transformation. Consider the fragment of state transition graph shown in Figure 1. The solid arrows represent the actual transitions between states. The shaded oval represents an equivalence class. Since states B and C are equivalent, it is possible to change the graph so that the transition out of A goes to C (as shown by the edge with solid arrow). The choice between B and C can be guided by considerations of power, delay and area.

To accomplish the restructuring transformation described in Figure 1, we first add and then remove edges, in such a way that the behavior of the machine is not affected. Suppose

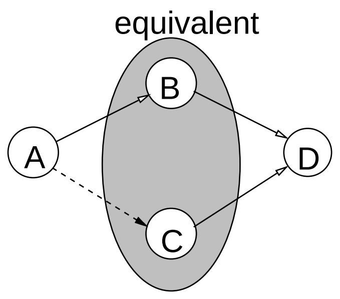

Figure 1: A fragment of state transition graph with equivalent states.

we are given a deterministic STG with transition relation T. For every state s_i, if there exists a transition to state s_j, then we add edges to every state s_k equivalent to s_j under the same input conditions. These new edges are called *ghost* edges. The resulting STG is no longer deterministic, but is clearly indistinguishable from the original one in terms of behavior. The transition relation of the resultant STG can be computed by the following simple formula:

$$T^a(x,w,y) = \exists_z E(y,z) \wedge T(x,w,z) \tag{1}$$

where $E(x,y)$ is the equivalence relation of the original FSM computed by the algorithm of [9].

We call T^a the *augmented* STG. Given an augmented STG we partition the synthesis problem into two phases: In the first phase we derive a new STG, which we call the *reduced* STG by removing undesirable edges from the augmented STG. In the second phase we synthesize a sequential circuit from the reduced STG. The second phase is the subject of Section 3. In the remainder of this section we describe the first phase, for which we propose and contrast several heuristic algorithms.

2.1 Priority Functions

As we can see from Figure 1 the problem is of selecting an edge which satisfies a cost function, from a possible choice of equivalent edges. Our purpose is to solve this problem for very large graphs whose size is beyond the possibility of traditional explicit algorithms. Hence, we need to avoid explicit enumeration of edges and make the selection in a single step. For this purpose we make use of priority functions [8] to select an edge according to a cost function.

Definition 1 *A priority function is a function $\Pi : |V|^3 \mapsto \{0,1\}$. The input to Π is a triplet of nodes (x,y,z). The first argument is the* bias*; the remaining two arguments are the nodes to be compared. For every choice of bias x, Π selects a total order of V and returns 1 if the second argument precedes the third in that ordering.*

In simple terms, $\Pi(x,y,z) = 1$ if y is a better choice than z, given x.

Example The priority function $\Pi_{RP}(x,y,z)$, called *relative proximity* is defined as:

$$\Pi_{RP}(x,y,z) = \begin{cases} 1, & \text{if } \|x-y\| < \|x-z\|; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

where $\|a-b\|$ is defined as

$$\|a-b\| = \sum_{i=0}^{n-1} |a_i - b_i| 2^{n-i-1} \tag{3}$$

$\Pi_{RP}(x,y,z) = 1$ if y is closer to x than z. This priority function will be used to build other priority functions in the following.

Given a cost criterion that the edges need to satisfy, it is possible to choose an appropriate priority function. Once a priority function Π is chosen, the following formula computes a *right-unique* edge relation, i.e., an edge relation in which there is at most one edge out of node encoded by x and which satisfies the cost criterion.

$$T^n(x,w,y) = T^a(x,w,y) \wedge \overline{\exists_z T^a(x,w,z) \wedge \Pi(x,z,y)} \tag{4}$$

In the above formula, the edge relation is denoted by $T^a(x,w,y)$. The term $T^a(x,w,z) \wedge \Pi(x,z,y)$ computes a triplet (x,y,z) (w is not involved in the computation as we choose an edge from a set of possible edges with identical labels). This triplet indicates that between the edges (x,y) and (x,z), the edge (x,z) is preferable to (x,y). After abstracting the nodes encoded by z, the result under the complement returns the edges not of highest priority (including edges that are not in the graph). The complement and the intersection finally return the edges of highest priority.

In the following sections we shall describe heuristics which embody different priority functions to reduce the average bit changes per state transition and the area of the final implementation.

2.2 Hamming Distance Heuristic

In this section we describe an algorithm that uses the Hamming distance as the basis to select edges from the augmented STG. The Hamming distance $HD(x,y)$ between a set of variables x and y is defined below.

$$HD(x,y) = \sum_{i=0}^{n-1} |x_i - y_i| \tag{5}$$

$HD(x,y)$ is represented by an ADD. The leaf values belong to the set $\{0,1,\ldots,n-1\}$.

We try to select a destination state such that the number of bit changes per state transition is minimized. By doing this, besides minimizing the toggling of the latches, we possibly reduce the switching activity in the combinational logic that implements the next state and output functions.

In order to select a destination state, such that the number of bit transitions from source to destination is minimized, we make use of the following function. It is defined as:

$$H(x,y,z) = \begin{cases} 1, & \text{if } HD(x,y) < HD(x,z); \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

$H(x,y,z)$ is not a priority function: For distinct encodings x, y, and z, $HD(x,y)$ can be equal to $HD(x,z)$; hence the application of H in Equation (4), does not guarantee a right-unique edge relation and the resultant STG will still be nondeterministic. To break the tie, we use the relative proximity function defined in Equation (2). The resultant priority function $\Pi_H(x,y,z)$ is defined as:

$$\Pi_H(x,y,z) = H(x,y,z) \vee (\neg H(x,z,y) \wedge \Pi_{RP}(x,y,z)) \quad (7)$$

The new edge relation, called the reduced STG is computed by substituting Π_H for Π in Equation (4). It is also possible to avoid the use of a tie breaker by synthesizing directly from a nondeterministic STG. This approach is outlined in Section 3.

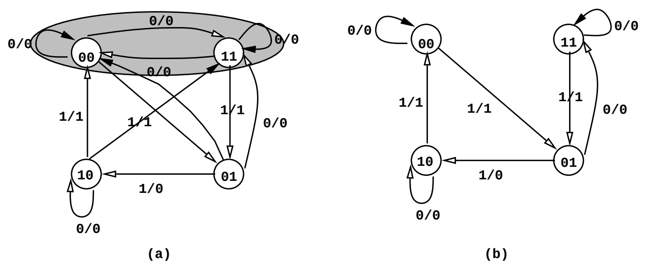

Figure 2: Example to illustrate Hamming distance heuristic.

Example. Figure 2 shows how the priority function selects only those edges that reduce the number of bit transitions. Figure 2(a) shows the augmented STG and Figure 2(b) the reduced STG. The states in the shaded area are equivalent states, and the edges with solid arrows are the ghost edges. The reduced STG now has fewer bit changes per state transition than the original STG.

2.3 Fanin-Oriented Heuristic

In the previous method, the potential area savings in the form of reduction in the number of states is not inherently taken into consideration. Potential area savings could occur, along with reduction in average bit change, in the case when all the incoming edges into an equivalence class get mapped to a single state. In this section, we propose an extension to the method described in Section 2.2 so as to include the potential area savings into the cost function.

In this method we take into account the absolute transition probabilities along with the Hamming distance criterion to make a better choice. The steady state probabilities $SS(x)$ are computed via markovian analysis along the lines of [7]. The steady state probability $SS(x)$ gives us the probability of occupation of the state encoded by x.

Each edge (x,y) of the augmented STG is annotated with the product of $N_b - HD(x,y)$ and the absolute transition probability. N_b denotes the number of bits encoding a state. The absolute transition probability between state s_i and s_j is given by the product of the probability of state occupation of s_i and the conditional transition probability between s_i and s_j. The one-step conditional transition probability $P^a(x,y)$ of the *augmented* STG is computed according to [7].

This weighted augmented STG is represented by an ADD. It is computed as below:

$$WT^a(x,y) = P^a(x,y) \cdot (N_b - HD(x,y)) \cdot SS(x). \quad (8)$$

Given the weighted matrix (relation) $WT^a(x,y)$ we choose a representative state for each equivalence class such that the sum of weights of the edges into that state is maximum. The maximum value gives preference to those edges that have the maximum probability of being taken and would result in minimum average bit change (maximum $N_b - HD(x,y)$). Since we consider only those transitions that have the states from an equivalence class as the destination state, this method is called the *fanin-oriented* method.

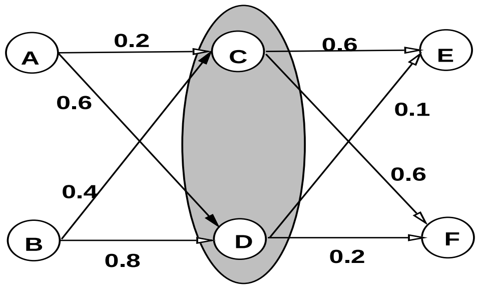

Figure 3: Fragment of augmented STG showing the edge weights.

Figure 3 shows a fragment of the augmented STG. The equivalent states are shown in the shaded area. As can be seen from the figure, the sum of the weights of the edges with C as the destination state is less than that of the edges with D as the destination. Hence the optimal choice would then be state D.

Selecting a Representative using Priority Functions. Initially an arbitrary choice of a *nominal representative* for each equivalence class is made, along the lines of [9] by using the initial state as the reference vertex. Let the projection function that maps each state to its nominal representative be denoted by $\Psi(x,y)$. The x variables encode the nominal representative and the y variables encode the state. The weight associated with each state of an equivalence class is computed as follows:

$$W^L(x,y) = \exists_v^+ WT^a(v,y) \cdot \Psi(x,y) \quad (9)$$

$W^L(x,y)$ can be considered as a weighted edge relation. The x variables encode the nominal representative and y the members of the equivalence class. The weight $w = W^L(x,y)$, on each edge (x,y) is the sum of all the weighted edges that have y as its destination state in $WT^a(x,y)$. In other words, w represents the *priority* value for the state y. We now select an *optimal* representative with the highest weight among the members of an equivalence class. To accomplish this, we use the relative proximity function defined in Equation (2) and a second function $FI(x,y,z)$ defined as:

$$FI(x,y,z) = \begin{cases} 1, & \text{if } W^L(x,y) > W^L(x,z); \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

The resultant priority function $\Pi_{FI}(x,y,z)$ is defined similarly to $\Pi_H(x,y,z)$:

$$\Pi_{FI}(x,y,z) = FI(x,y,z) \vee (\neg FI(x,z,y) \wedge \Pi_{RP}(x,y,z)). \tag{11}$$

The pair (x,y) of nominal and optimal representative for each class is computed as follows:

$$R(x,y) = \Psi(x,y) \wedge \overline{\exists_z \Psi(x,z) \wedge \Pi_{FI}(x,z,y)}. \tag{12}$$

where, x encodes the nominal representative and y the corresponding optimal representative. The new projection function is then computed as follows:

$$\Psi^n(x,y) = \exists_z \Psi(z,y) \wedge R(z,x). \tag{13}$$

Once the optimal representative for each equivalence class is found, the original transition relation $T(x,w,y)$, the output functions λ_i and the initial state S^0 are modified using the new projection function as follows:

$$\begin{aligned} T^n(x,w,y) &= \exists_{uv}(\Psi^n(x,u) \wedge T(u,w,v) \wedge \Psi^n(y,v)) && (14)\\ \lambda_i^n(x,w) &= \exists_u(\Psi^n(x,u) \wedge \lambda_i(u,w)) && (15)\\ S^{0n}(x) &= \exists_u(\Psi^n(x,u) \wedge S^0(u))) && (16) \end{aligned}$$

2.4 Fanin-Fanout Oriented Heuristic

A more sophisticated approach performs a less greedy selection. For instance, in Figure 3 the choice of state D results in higher average bit change because of the transitions that emerge from state D. To choose an optimal representative for an equivalence class, it is sometimes beneficial to consider both the transitions into and from the equivalent states. We call such a method *fanin-fanout* method.

In addition to the weighted matrix $W^L(x,y)$ as described in Section 2.3, we define $W^R(x,y)$ to take into consideration the contribution of the transitions that originate from states in the equivalence classes, to the average bit change. The process is very similar to that described in Section 2.3, except for $W^R(x,y)$ which is computed as follows:

$$W^R(x,y) = \exists_v^+ WT^a(y,v) \cdot \Psi(x,y) \tag{17}$$

The complete weight matrix $W(x,y)$ is the average of $W^L(x,y)$ and $W^R(x,y)$. Equations (10) through (16) still hold with $W^L(x,y)$ replaced by $W(x,y)$.

3 Converting the State Transition Graph into a Circuit

After the STG has been restructured as described in Section 2, a new circuit must be derived from it. Direct conversion of the BDD to a circuit by translating each node of the BDD into a multiplexer leads to a circuit that is typically large and slow. Optimization of such a circuit with a tool like SIS [13] is normally time consuming and ineffective, because of the poor quality of the starting point.

We derive an efficient circuit from the BDDs of the STG by the symbolic techniques based on ZDDs [10, 11], which we have extended to deal with low power consumption. The inputs to the conversion procedure are the restructured STG, $T^n(x,w,y)$, the output functions, $\{\lambda_i^n(x,w)\}$, and the set of reachable states of T^n, $R(x)$. If the circuit does not have a prescribed set of initial states, then $R(x) = 1$.

Initially, the next state functions are extracted from T^n. For a deterministic T^n—one that associates exactly one next state to each combination of present state and primary inputs—the i-th next state function $\delta_i^n(x,w)$ can be computed as follows. First the lower bound is found by the following formula:

$$\delta_i^{ln}(x,w) = (\exists_y [T^n(x,w,y)_{y_i}]) \wedge R(x). \tag{18}$$

Similarly the upper bound is given by the formula:

$$\delta_i^{un}(x,w) = \delta_i^{ln}(x,w) + R(x)'. \tag{19}$$

If T^n is nondeterministic, it is still possible to extract the next state functions from it, though the result is not unique. To do so, one solves the boolean equation $T^n(x,w,y) = 1$ for the y variables to find the most general solution [4]. Then one extracts a particular solution by assigning appropriate values to the parameters in the general solution [6]. This approach can be used in the case of the Hamming distance heuristic (Section 2.2). The *fanin* and *fanin-fanout* methods produce a deterministic STG; hence, we do not need this more general approach.

4 Symbolic Factorization using ZDDs

The factorization procedure inputs BDDs of the next states and output functions and don't cares. We use Minato's ISOP (Irredundant Sum Of Products) algorithm [10] to extract a ZDD from the BDD to represent an irredundant cover of the function.

4.1 Finding Good Divisors

The quality of factorization depends on the ability to find good divisors. There are several ways to find a divisor for a function to be factored. We have four methods to get divisors: least occurrence divisor, most occurrence divisor, level-0 kernel divisor, and random divisor. Least occurrence divisor finds a literal that occurs least frequently in a function, among those that occur more than once. Most occurrence divisor finds a literal which occurs most frequently in a function. Once the literal is chosen, we divide the function by the literal to get a quotient, and we recur on the quotient. Level-0 kernel divisor is slightly different from most occurrence divisor in that we make the quotient cube-free before we recur. A random divisor can be found quickly by tracing through a ZDD graph. As soon as we find a ZDD node that has two or more predecessors, we simply use the literal of this node as a divisor, and we recur. This can be done by depth first search. When a node is visited on a ZDD graph, if the node is already marked, we stop and return this node as a divisor. Otherwise, we mark the node and recur. In Figure 4, we visit the nodes a-d-e-f-b-d. Since the node d is already marked, d is chosen as a divisor.

When we have a tie, we try to find a better divisor by considering how many output functions contain this variable

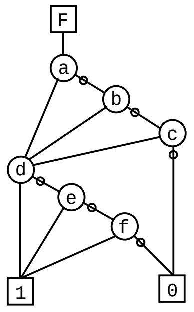

Figure 4: An example of a ZDD cover.

and what the input probability is. If the transition probability of an input is known, the input with highest transition probability is chosen. This would ensure that an high activity input will be closer to the output and hence reduce the switching activity in the combinational part.

Among the factoring methods, no one is consistently superior to the others. Therefore, our program offers all of them as options.

4.2 Maximizing Sharing

The more nodes we share, the fewer literals we get. In order to maximize sharing of common nodes, we do the following.

First, we order the output functions according to the size of their supports. We factor the functions with the smallest support first.

Second, we use algebraic re-substitution [3]. After we divide a function by a divisor, division of the remainder by the complement of the divisor is attempted.

Third, we keep all divisors, quotients, and remainders that are already generated by previous divisions in a list, sorted by the size of their support. Whenever we find a divisor for a function, we extract a divisor from the function itself, and another divisor from the list. We then use the one with more literals.

Fourth, we use the complement of each divisor. If the complement of a divisor can divide another function, we can increase sharing of nodes. Whenever we put divisors and quotients in the divisor list, we insert their complements as well.

Fifth, we use kernel-by-kernel division. After we finish factoring all output functions, we have a lot of level-0 kernels that cannot be factored further. However, there is still a chances to share nodes. A level-0 kernel may be divided by another level-0 kernel, and a cube of a kernel may divide other level-0 kernels. This post-processing is very important to reduce the number of literals in factorization. As an example, consider the following two level-0 kernels.

$$\begin{aligned} F &= a + bcd \\ G &= b' + c' + d' \end{aligned}$$

Even though both F and G cannot be factored further, F can be divided by the complement of G. Therefore, we can save two literals by letting $F = a + G'$.

5 Experimental Results

All the algorithms that we have described in this paper have been implemented in VIS [2]. From our experiments we have observed that neither of the previously described restructuring algorithms, *viz.*, *Hamming distance* based heuristic, *fanin*, and *fanin-fanout* methods was consistently better than the others. So, the results that we have reported here correspond to the best run among the three restructuring options. In Table 2 we report the results of the restructuring process. The algorithms were run on many ISCAS '89 benchmarks. The experiments were conducted on an UltraSparc 167MHz with 192MB of memory. To get an estimate of the power dissipated we have used the power estimator in SIS [13] package. No further optimizations are performed in SIS after symbolic synthesis. Library delay model was used for all the experiments. The circuits were mapped with the **map -AFG -n 1** command using the **lib2** and **lib2_latch** standard libraries. Since performance is an important factor, we have mapped the circuits to reduce delay, rather than area.

Circuit	Minato		MIS		Ours	
	# lit	Time	# lit	Time	# lit	Time
xor8	28	0.3	28	38.3	28	0.02
xor16	60	0.7	—	—	60	2.37
9sym	117	1.8	83	29.8	76	0.14
vg2	102	1.7	97	33.9	93	0.40
alu4	1148	64.5	1319	3751.6	1375	5.25
apex1	2521	209.6	2863	10945.1	1784	7.72
apex2	253	29.5	—	—	336	3.00
apex3	2221	158.2	2132	1926.6	1687	11.92
apex4	3473	462.4	3509	1345.9	1979	21.93
apex5	1185	58.7	1206	156.9	1070	3.90
c432	1510	692.3	—	—	2454	36.52

Table 1: Results of symbolic factorization.

In Table 2 column *Equiv* states the number of equivalence classes. The average bit changes are shown in column *ABC*. From the experiments it can be seen that most of the circuits have many equivalent states. Circuits *tlc, s344, s641* show substantial decreases in the average bit change after restructuring. Though there is no substantial decrease in the average bit change for other circuits, the power dissipation, nevertheless decreases. This is due in part to a decrease in area. In some cases like *s832*, *s1494*, *s386* and *s510*, even though restructuring did not decrease the average bit change, restructuring did produce reduced STGs with fewer states resulting in lower power dissipation and lower area. In the case of *s298* the area as well as the power increased after restructuring. This is due in part to the failure of the factoring heuristics on this specific example. The time taken to complete the restructuring and the symbolic synthesis process for all the examples is very small. The time reported is the time taken in CPU seconds for the symbolic synthesis of original circuits, restructuring and the synthesis of the reduced FSMs.

The results show that our symbolic synthesis approach

Ckt.					Original				Restructured				
	in	out	FF	Equiv	lit	μW	area	ABC	lit	μW	area	ABC	time
s832	19	18	5	25	371	749.5	471424	0.82	331	678.8	413424	0.82	0
s1494	8	19	6	49	777	1915.7	967904	1.96	686	1910.9	860720	1.94	0
s386	7	7	6	15	163	320.5	217616	1.04	118	291.2	166112	1.04	0
s510	19	7	6	59	287	783.2	390244	1.51	284	766.7	373520	1.51	8
tlc	3	5	10	254	101	211.5	155904	2.0	73	181.5	127136	1.6	1
s298	3	6	14	8061	136	253.0	219008	1.97	159	360.8	225504	1.96	7
s344	9	11	15	18608	228	647.2	308096	4.22	246	568.6	319232	3.68	19
s641	37	23	19	294912	633	1562.2	773024	1.97	451	1197.8	579072	0.89	474
s400	3	6	21	608448	259	302.9	387904	2.0	206	302	286288	2.0	21
s526	3	6	21	1432190	239	294.4	339184	2.0	212	280.4	296960	2.0	141

Table 2: Symbolic restructuring and synthesis.

is quite effective in most cases. Refinement of the synthesis process is still under development and we hope to have even better results soon.

The results of our symbolic factorization are shown in Table 1. The results reported in this table are for combinational synthesis. The results of Minato and MIS come from [11]. Those experiments were run on SPARCstation 2. Even assuming a conservative factor of 10 between the speed of the two computers, our implementation is faster, especially on the larger examples. In terms of literals, we get mostly better results than both Minato and MIS. There is still some room for improvement. The strategy for choosing divisors is straightforward and we use only the algebraic division and sequential factorization for multiple output functions. If we adapt a backtracking strategy, boolean division, and simultaneous factorization in our algorithm, we may get better results.

6 Conclusions

In this paper we have presented algorithms for the restructuring of state transition graphs without changing the behavior of the circuit. From experimental results we have found that the restructuring works well for circuits with large number of equivalent states. Currently, we are in the process of synthesizing circuits described in Verilog. Behavioral descriptions often have large redundancies. We plan to integrate restructuring algorithms with symbolic re-encoding of sequential circuits to take advantage of these redundancies. With these two methodologies in place we hope the synthesis process to improve further.

7 Acknowledgments

We would like to thank Srilatha Manne for the interesting discussions on this topic.

References

[1] 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*, pages 188–191, Santa Clara, CA, November 1993.

[2] R. K. Brayton et al. VIS: A system for verification and synthesis. Technical Report UCB/ERL M95/104, Electronics Research Lab, Univ. of California, December 1995.

[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A multiple-level interactive logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(6):1062–1081, November 1987.

[4] F. M. Brown. *Boolean Reasoning: The Logic of Boolean Equations.* Kluwer, Boston, 1990.

[5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[6] M. Fujita, Y. Tamiya, Y. Kukimoto, and K.-C. Chen. Application of boolean unification to combinational logic synthesis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 510–513, Santa Clara, CA, November 1991.

[7] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE Transactions on Computer-Aided Design*, 15(12):1479–1493, December 1996.

[8] G. D. Hachtel and F. Somenzi. A symbolic algorithm for maximum flow in 0-1 networks. In *Proceedings of the International Conference on Computer-Aided Design*, pages 403–406, Santa Clara, CA, November 1993.

[9] B. Lin, H. Touati, and A. R. Newton. Don't care minimization of multi-level sequential logic networks. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 414–417, Santa Clara, CA, November 1990.

[10] S.-I. Minato. Fast generation of irredundant sums-of-products forms from binary decision diagrams. In *SASIMI '92*, pages 64–73, Kyoto, Japan, April 1992.

[11] S.-I. Minato. Fast weak-division method for implicit cube representation. In *SASIMI '93*, pages 423–432, Nara, Japan, October 1993.

[12] S.-I. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the Design Automation Conference*, pages 272–277, Dallas, TX, June 1993.

[13] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. In *Proceedings of the International Conference on Computer Design*, pages 328–333, Cambridge, MA, October 1992.