Re-Encoding Sequential Circuits to Reduce Power Dissipation*

Gary D. Hachtel \textsuperscript{1} Mariano Hermida \textsuperscript{1} Abelardo Pardo \textsuperscript{1} Massimo Poncino \textsuperscript{2} Fabio Somenzi

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

Abstract

We present a fully implicit encoding algorithm for minimization of average power dissipation in sequential circuits, based on the reduction of the average number of bit changes per state transition.

We have studied two novel schemes for this purpose, one based on recursive weighted non-bipartite matching, and one on recursive mincut bi-partitioning. We employ ADDs (Algebraic Decision Diagrams) to compute the transition probabilities, to measure potential area saving, and in the encoding algorithms themselves.

Our experiments show the effectiveness of our method in reducing power dissipation for large sequential designs.

1 Introduction

The importance of low-power, high-throughput microelectronic systems is rapidly increasing. High power dissipation limits the developments of portable applications that demand intensive high-speed computation. Further, excessive power dissipation is a limiting factor in integrating more transistors on a single chip.

Power directed synthesis techniques can significantly reduce power dissipation [10, 6]. They can be viewed as straightforward modifications of conventional logic synthesis approaches, and are applicable to a very broad class of digital designs. All of these works exploit the estimated transition frequency (switching activity) in directing the synthesis [9], because in CMOS circuits there is little standby power consumption.

Regarding sequential synthesis for low power, previous work on encoding [10] follows the paradigm of conventional FSM encoding algorithms, such as [2, 4], where the states are referred to explicitly, and therefore cannot be applied to large practical state machines.

We employ the recently developed ADD (Algebraic Decision Diagram, [1]) technology to overcome these limitations. ADDs are a form of multi-terminal BDDs that support algebraic and arithmetic operations on their terminal nodes, which can hold objects drawn from an arbitrary set, e.g., real numbers. ADDs are the key to the Markov analysis which gives the edge weights which drive the re-encoding algorithm. ADDs have allowed us to solve the Chapman-Kolmogorov equations [10, 6] for realistic machines (million of states) that are not manageable by conventional sparse matrix techniques.

In the past we focused on power optimization of sequential logic by re-encoding an existing circuit. We try to find an encoding of the states such that the average 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 the output functions.

Our method initially computes the steady-state probabilities, and the state transition probabilities, as discussed in [7]. We then re-encode the circuit so as to minimize latch activity. We have studied two novel encoding strategies, one based on recursive weighted non-bipartite matching, and one on recursive mincut bi-partitioning, that combine the well-known Dolev-McCluskey method with the multi-terminal optimization techniques of the ADD-based algorithm [2] by building a weight matrix which is a convex combination of transition probabilities and area optimization potential. The next-state and output functions are then re-encoded by solving a boolean equation.

While our experimental results are preliminary, they demonstrate that our ADD technology is practically significant.

2 Matching Based Re-Encoding

Our first method is based on recursive maximum weighted matching, which we solve with an ADD extension of a BDD-based mincut/maxflow algorithm [8]; the levels of recursion correspond to encoding bits as in [3]. At each step of the recursion the states with the greatest transition probability are matched, and the half with the greatest match weight are grouped in the same partition, and receive the same code bit.

Figure 1 shows the re-encoding algorithm. Variable sets \( x \) and \( y \) encode rows and columns, respectively.

```
Encode \([G(x, y), N]\) \{
  i = 0;
  \(G'(x, y) = G(x, y)\);
  for (i = 0; i < N; i++) \{
    \(M'(x, y) = \text{Max Weight Matching}(G'(x, y))\);
    \(C_{\text{Add}}(x) = 2 \cdot (G(x, y) \cdot (x > y))\);
    \(Part(x, y) = \text{Pairs Of Matched Nodes}(M'(x, y))\);
    \(G'^{(i)}(x, y) = \text{Total Weight}(G'(x, y), Part(x, y))\);
  \}
```

Figure 1: High-Level Encoding Procedure.

The procedure Encode gets the matrix \( G \) and the number of latches of the circuit \( N \) as inputs.

Transition probabilities in \( G \) are non-zero only for states in the terminal SCCs of the transition graph. The reduction in the switching activity is obtained from the analysis of this (sometimes very small) subset of the actually reachable states. This is correct from the standpoint of average power dissipation, since transient states play a negligible role in the average power dissipation. Transient states are indeed relevant for area minimization, and they are taken into account for potential area savings.

At each iteration, we first find a maximum matching \( M' \) in the matrix \( G' \) (Max Weight Matching). Then, a "1" is arbitrarily assigned to the \( i \)-th bit of the new encoding of one member of each pair of matched nodes, and a "0" to the other. The node with higher index \( (x > y) \) is chosen as representative of each matched pair. The re-encoding is given as a 

\[ \text{trans-coding function}, \] which expresses the bit of the new encoding as a function of the bits of the old one.

After assigning to every state one of the bits of the encoding, we shrink the graph. That is, we find a new

---

*This work is supported in part by NSF/DARPA grant MIP-9115432 and SRC contract 92-DJ-206.

1Mariano Hermida is with the Universidad Politecnica De Madrid, Facultad Informatica, Madrid, Spain

2Massimo Poncino is also with the Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, ITAUX 10129.

weighted graph \( G^{k+1} \), in which the new nodes are the pairs of matched states. After the \( k \)-th matching stage of this recursion, a supemode corresponds to \( 2^k \) original states. Since every new node (state) gets either a "1" or a "0" in the next least significant bit of the new encoding, all the original nodes which formed the current (super) node get the same value for the \( k \)-th bit. Therefore, they differ in the bit corresponding to the iteration in which they were matched, and possibly in bits of preceding steps.

At each stage we must match all the states; hence the matching contains some degree of arbitrariness, since the matching of unreachable states is irrelevant. If no area saving weight is included, the same holds for states which are outside the terminal SCC of the transition graph. This degree of arbitrariness in the re-encoding can be used later in the resynthesis phase, for example, to simplify the logical expression or to try to minimize delay.

### 2.1 Weighted Non-Bipartite Matching

We now focus on finding a maximum weighted matching on a graph with an implicit algorithm. Our purpose is to solve this problem for very large graphs, whose size is beyond the possibility of traditional explicit algorithms.

The theoretical best solution is to use some implicit variant of Edmonds’s matching algorithm [5]. For general non-bipartite graphs, blossoms (odd cycles) significantly increase the complexity of the matching algorithm: It is difficult to deal with blossoms symbolically. On the other hand, we can trade-off accuracy for time and memory by targeting an approximate solution to the problem.

In the following we deviate from Edmonds’s algorithm in two ways. First, we get rid of odd cycles as soon as they appear (suppressing the edges which cause them). Second, after reducing the graph in this manner, we find an approximate maximum weighted matching.

We have developed a heuristic for quasi-maximum weighted matching that is amenable to symbolic implementations. It is based on (1) sorting the edges according to their weight, and (2) selecting a subset of the edges in decreasing order of weight. These edges are then used to find a first maximum unweighted matching. If this matching is not complete, we then add more edges to the selected set of edges and try to match nodes which are not yet match.

The process is continued until a complete matching is obtained.

The core of the matching is based on growing *alternating trees* from every unmatched node, in order to find an alternating path to another unmatched node. We build alternating paths rooted at exposed nodes till they meet again or expose a new node. However, to make our approach symbolic, we grow trees from every exposed node, and terminate the growth when they meet another tree. This is where our symbolization of Edmonds’s algorithm is similar to the symbolization of Dinitz's algorithm in [8].

Figure 2 shows the augmenting paths growing algorithm. The parameter \( E \) is the edge relation of the graph, and \( M \) is an initial matching; \( i \) represent the iteration index.

### 3 Mincut Based Re-Encoding

The alternative to maximum weighted matching is recursive mincut bi-partitioning. We extend the original work by Lin and Kernighan to the symbolic BDD/ADD based setting. Although our symbolic version gives up the linear complexity update of the Kernighan-Lin scheme, symbolic processing vastly extends capacity in cases amenable to BDD representations.

The idea is illustrated in Figure 3. Given an initial partition of the states into \( L \) and \( R \), migration groups \( MR \) (Move Right) and \( ML \) (Move Left) are computed as follows. \( MR \) is the set of states \( s \) for which the sum of the weights of edges going from \( s \) to \( R \) minus the sum of the weights of edges going from \( s \) to \( L \) is less than \( \epsilon \) (a user-defined threshold). \( ML \) is similarly defined. This process is iterated until no such states exist. The process may then be repeated with a smaller \( \epsilon \), if appropriate.

### Figure 2: High-Level Tree Matching Procedure

Once the partition on the global set of states is obtained, we assign one binary value to each block of the partition, corresponding to the most significant bit of the encoding, and we recur on the two halves. At the end, every node will have a distinct code. States having high mutual transition probabilities will get low-distance encodings. The algorithm is shown in the following figures. The high level procedure is shown in Figure 4. In the following, \( r \) and \( c \) variables encode row and column indices.

The procedure RecursiveMincut gets the current weight (sub)matrix \( A(r,c) \), the partition \( (L,R) \), the coding functions \( Codes \), and the current index of the state bit \( n \) as inputs. Initially, \( A \) is the transition probability matrix, \( L \) and \( R \) are the tautologies, \( Codes \) is the zero function, and \( n \) the number of state bits. \( Codes \) is a trans-coding function, which will give each bit of the new encoding as a function of the old coding bits.

Line 1 shows the termination condition. When the current subspace consists of two states only, we do not attempt further moves, and immediately update the LSB of the new encoding. In Line 2, an initial partition \( (L,R) \) is computed by split-set. We cannot simply split on the top variable since \( L \) and \( R \) must he have the same size. This is because we will eventually assign a 1 in the \( i \)-th code bit to all states of one of the two partitions, and there must be the same number of states with a given code bit set to 1 and to 0. The effectiveness of a partitioning algorithm depends on
the initial partition. An alternative for the initial guess is the partition produced by a single run of the symbolic matching algorithm described in Section 2.

In Line 3 we compute the cut-size of the current partition. In Lines 4 and 5, the migration groups \( MR \) and \( ML \) are computed, as detailed in Figure 5. The blocks of the initial partition are then updated.

In general, the migration procedures compute sets of different size. Since the partition must be balanced, the procedure correction_set (Line 6), computes a set of states \( CS(\pi) \) whose size equals the difference \( |ML| - |MR| \).

Then, we update the partition according to the migration groups and the correction set to get the new \( L \) and \( R \). At this point the new cut-size is computed and compared to the previous one. If the new one is larger, we restore the initial partition. This case may arise since we force to move equally sized sets of states.

In Line 9, we update the \( i \)-th code bit function with the \( L \) function, obtained by the migration procedures. Notice that the coding functions are built by accumulating partial results at the same level of recursion. We then recur on \( L \) (Line 10). The set \( L(\pi; R(\pi; L(\pi), C(\pi), C(\pi), C(\pi), C(\pi), C(\pi), C(\pi)) \) represents the subgraph consisting of nodes and edges in \( L \). In Line 11 we repeat the same operation for states in \( R \).

In the group migration algorithm, \( \text{mindepth} \) is a parameter which prevents the move of too large sets of states. In the early stages of the recursion, it is unlikely that moves of large sets of nodes are profitable. Since we want to reduce the "expensive" operations, we allow them only at depth greater than \( \text{mindepth} \). Clearly, \( \text{mindepth} \leq n \).

The procedures receives the sets \( L \) and \( R \), the weight matrix \( A \), a function \( p \) which represents the cofactor cube, and the maximum depth \( n \) of the recursion.

Figure 4: High-Level Procedure.

4 Area Minimization

If we use only the transition probabilities matrix as the underlying graph of the algorithm, we may get an encoding that, though optimal for minimizing the transition on the state lines, may cause an increase in the circuit area. This increase in area, besides being undesirable by itself, could eventually mask the saving in power, since the extra gates will dissipate some power. Therefore, we need a mechanism which allows to control the area build up. Since area is not the main concern of our algorithm, we can accept a relatively simple measure, which is easily amenable to a symbolic formulation, like the one used in MUSTANG [2].

In MUSTANG, the attractive force between states is related to the ability of extracting common cubes from either the next state or output functions, according to the two variants (fanout-oriented and fanin-oriented) of the algorithm. In the fanout-oriented algorithm, states that have a common fanout state are attracted; in the fanin-oriented, states that have a common fanin state are attracted. Both algorithms build a weight matrix, which expresses the attraction between pairs of states. We compute the weight matrices symbolically, by representing them as ADDs. In the following description, \( T(s, x, t) \) represents the transition relation of the STG.

Fanout-oriented algorithm. First we build a matrix \( S(s, t) \); entry \( s_i \) gives the number of input patterns that label the transition between state \( s_i \) and state \( s_j \). (10-1 yields a value of 4, because it represents four patterns.) \( S \) is obtained as:

\[
S(s, t) = \sum_{i=1}^{n} T(s, i, t)
\]

Then we build a second matrix \( Z(s, o) \), having one row for each state, and one column for each output. Entry \( z_{jo} \) gives the number of input patterns that, on the arc from state \( s_i \), assert output \( o_j \). To compute \( Z \), we need the output functions \( \lambda(x, s) \). The variables denoted with \( o \) encode primary outputs by randomly selecting a binary code for each output variable. The \( i \)-th column of \( Z \) is given by:

\[
Z(s, o) = \sum_{i=1}^{n} \lambda(x, s, o)
\]

Then, the weight matrix is obtained as:

\[
W(s, s) = \frac{1}{2} \cdot (S(s, s) \times S^T(s, s)) + (Z(s, o) \times Z^T(o, o))
\]
$N_b$ is the number of encoding bits. The factor $\frac{N_b}{N}$ says that the number of occurrences of the common cube depends on the number of 1's in the next state code. In the end, $W$ is a square matrix with as many rows and columns as the number of states.

In a similar fashion we compute the weight matrix $W(t, s)$ for the fanin-oriented algorithm. The two variants may be used either separately or together. In the second case, we get a weight matrix whose elements are the average of the weight values of $W$ and $\overline{W}$. The overall weight matrix is an input to the encoding algorithm, together with the transition probability matrix.

5 Experimental Results

We have applied our encoding algorithm to examples taken from the ISCAS'89 and MCNC benchmarks. Table 1 shows the results obtained on a DECstation 5000/200 with 80 MB of memory. The columns labeled Bit Changes give the average number of bit changes in the encoding, for both the original circuit (column Orig) and the re-encoded one (column Re-enc). This quantity corresponds to the cost of the encoding $C = \sum_{i,j} w_{ij} - d(e_i, e_j)$, where $w_{ij}$ is the value of the weight matrix, and $d(e_i, e_j)$ is the Hamming distance between the codewords $e_i$ and $e_j$. The values in Table 1 are obtained with the matching algorithm, and show that this cost function is significantly reduced in most examples.

Table 1: Encoding Results.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Latches</th>
<th>States</th>
<th>Bit Changes</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>277</td>
<td>3</td>
<td>8</td>
<td>0.89</td>
<td>0.3</td>
</tr>
<tr>
<td>keyb</td>
<td>5</td>
<td>37</td>
<td>0.83</td>
<td>0.65</td>
</tr>
<tr>
<td>1lk</td>
<td>5</td>
<td>37</td>
<td>1.53</td>
<td>1.21</td>
</tr>
<tr>
<td>s77</td>
<td>5</td>
<td>37</td>
<td>1.26</td>
<td>0.91</td>
</tr>
<tr>
<td>t720</td>
<td>10</td>
<td>37</td>
<td>0.78</td>
<td>0.94</td>
</tr>
<tr>
<td>11888</td>
<td>6</td>
<td>64</td>
<td>0.92</td>
<td>0.86</td>
</tr>
<tr>
<td>1396</td>
<td>6</td>
<td>64</td>
<td>1.03</td>
<td>1.6</td>
</tr>
<tr>
<td>181</td>
<td>5</td>
<td>132</td>
<td>1.28</td>
<td>2.88</td>
</tr>
<tr>
<td>120</td>
<td>8</td>
<td>292</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>1074</td>
<td>1.98</td>
<td>1.72</td>
</tr>
<tr>
<td>1278</td>
<td>14</td>
<td>1074</td>
<td>1.96</td>
<td>1.80</td>
</tr>
<tr>
<td>1240</td>
<td>9</td>
<td>65126</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>4010</td>
<td>21</td>
<td>266</td>
<td>1.59</td>
<td>1.37</td>
</tr>
</tbody>
</table>

Table 2: Estimated Power Results.

Power roughly scales with area, but for cm18, s1488, and scf, despite an increase in area, the resulting power is lower. The largest circuits of Table 1 are missing from Table 2 because the experimental procedure has two major bottlenecks. First, computing the next-state and output functions of the re-encoded circuit is rather expensive, in terms of memory. Moreover, the blif files generated from the BDDs are, in some cases, too large to be processed by sis. Second, the power estimator of [6] is very memory intensive. Therefore, although we have been able to generate encodings for some large circuits, we could not evaluate the dissipated power.

6 Conclusions and Future Work

We have described two symbolic methods for re-encoding a circuit to reduce the dissipated power; they have proven effective for designs that are too large for traditional encoding techniques. Although the experimental results are promising, the synthesis procedure still needs further refinements in both the circuit transformation and the power estimation phases.

The matching algorithm gives more accurate results, while the mincut approach takes less memory, and therefore can be applied to larger circuits.

We are currently investigating the possibility to account for unreachable states already in the encoding processes. This implies executing the algorithms on a subgraph that contains only the reachable states. The cost functions obtained in this case would be incompletely specified, and the problem of assigning a distinct code word to each state can be addressed in the transformation phase.

This technique may especially benefit the mincut approach, where we address nodes. In the matching approach, where we match edges, the impact may be more limited.

References


