A Semi-Gray Encoding Algorithm for Low-Power State Assignment*

Chunhong Chen, Jiang Zhao, and Majid Ahmadi
Department of Electrical and Computer Engineering
University of Windsor, Ontario, Canada

ABSTRACT

In sequential circuit synthesis, state encoding is to assign binary codes to given symbolic states such that a specific objective function such as area or power dissipation can be minimized in the final implementation. For low power design, reducing the switching activity of state bits is of special interest. In this paper we present a near-optimal semi-Gray encoding technique to minimize the switching activity for any given state transition graphs. Experiments with standard benchmarks show that there is a significant improvement over previous algorithms in terms of both speed and result quality.

1. INTRODUCTION

Sequential circuit design involves the state encoding (or state assignment) of finite state machines (FSMs) and the logic synthesis of combinational components. State encoding uses specific binary codes to represent different symbolic states in FSMs such that some objectives are optimized. Since dynamic power dissipation is proportional to the average number of signal transitions, the switching activity on state variables is a typical cost function for low power. However, this function does not account for the power dissipation due to the resulting combinational logic which is affected by different encoding strategy, depending upon the specific implementation (two-level or multilevel). In general, a set of constraints on the relationship between codes for different states can be derived, and these constraints should be satisfied for producing good results (i.e., small area or low power dissipation).

Earlier attempts at state encoding optimize the area of two- and multi-level implementations [1-3, 5, 6, 10]. Recently-developed approaches have targeted toward low power design [4, 7-9]. A typical example of low power technique is the so-called column-based algorithm [7] which takes one signal state variable at a time and tries to assign its value for each state such that the switching activity can be minimized for the complete assignment. This is done iteratively for each state variable until the codes have been completely specified. The idea is to give states that are associated with high transition probability the same value for most state variables. While minimizing the switching activity by itself does not guarantee the minimum power dissipation, experiments have shown that there is a strong correlation between the switching activity metric and the actual power dissipation [7]. A good survey of state encoding can be found in [10].

In this paper we describe a novel semi-Gray encoding technique to minimize the switching activity associated with the state registers in a given FSM. We show the effectiveness of our algorithm by comparing with optimal results for small problems and significantly improving the results from [7] on benchmarks in terms of both switching activity and computation time. The rest of the paper is organized as follows. Section 2 introduces the state encoding model for low power dissipation. In Section 3 we propose a semi-Gray encoding algorithm together with a simple testing example. The experimental results and discussions are given in Section 4, and Section 5 concludes the paper.

* This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant 249499-02.

2. STATE ENCODING MODEL

An FSM with n_s states can be described by a state transition graph (STG), $G = (V, E)$ where each vertex $s_i \in V$ represents a state and each directed edge $e_{i,j} \in E$ represents the transition from one state (s_i) to another (s_j). The conditional probability $t_{i,j}$ for edge $e_{i,j}$ is the probability that the next state is s_j given the present state being s_i [11]. The total transition probability $P_{i,j}$ between s_i and s_j is the probability that a transition from s_i to s_j occurs, and is thus given by $P_{i,j} = t_{i,j}\, p_i$, where p_i is the state probability (i.e., the probability that the machine is in a given state s_i). If all multiple directed edges between any two states are collapsed into a single undirected edge with weight $w_{i,j}$ equal to the sum of the directed edges' total transition probabilities, a weighted undirected graph $G_w = (V, E_w)$ can be obtained. The power metric of state encoding is to minimize the switching activity:

$$SW = \sum_{s_i, s_j \in V} w_{i,j} H(s_i, s_j) \qquad (1)$$
$$s.t. \quad H(s_i, s_j) \geq 1, \quad s_i \neq s_j$$

where $H(s_i, s_j)$ is the *hamming* distance between the codes for two states s_i and s_j in the undirected graph G_w. $H(s_i, s_j) \geq 1$ means that no two states can be assigned to the same code.

Note that for n_s states, the minimum number of state variables required is $\lceil log_2\, n_s \rceil$. While the optimal length of binary codes assigned to states could be larger, most state encoding algorithms produce the best results with minimum-length codes. For small FSMs with a few state variables, an optimal solution can be achieved using an exact integer linear programming approach [12]. For large problems, heuristics are required in general to get near-optimal results.

3. SEMI-GRAY ENCODING

Consider a weighted undirected graph G_w as defined above. Ideally, if each state transition brings about only one single state variable change, we will achieve the optimal (minimum) switching activity which is the sum of weights on all edges in the graph. A typical instance is a counter with Gray encoding which guarantees that the transition between two successive numbers results in only one bit switching. For general graphs, the two states connected by a high-weight edge should be assigned to adjacent codes. Intuitively, when all states are sorted out in decreasing order of

0-7803-7761-3/03/$17.00 ©2003 IEEE

weight of connecting edges, one can divide these states into groups, followed by Gray-encoding group by group. More generally, this is done in the following two phases: (a) grouping, and (b) encoding.

3.1 Grouping

Given a set of n_s states $V = \{s_1, s_2, \ldots, s_{ns}\}$ and an integer $l = \lceil log_2 n_s \rceil$, a state encoding of V is a one-to-one mapping $V \rightarrow \{0, 1\}^l$, where l is the number of bits in binary codes. Grouping is to divide binary codes into groups (denoted by $C_1, C_2, \ldots, C_g$) with k Gray codes each, and to divide V into groups (denoted by $S_1, S_2, \ldots, S_g$) with at most k states each, where k is the power of 2 (typically k is selected as 4 in this paper). The number of groups is given by $g = \frac{2^{\lceil \log_2 n_s \rceil}}{k}$. Thus, the binary encoding problem becomes a one-to-one mapping from a set of state groups to a set of code groups. Within each group of k elements, an exact solution to encoding can be found efficiently as long as the value of k is small enough. To take full advantage of freedom in encoding for this case, n_s states are evenly distributed among all groups. Assuming that there are g_1 groups with $\lceil n_s/g \rceil$ states each, and g_2 groups with $\lfloor n_s/g \rfloor$ states each, one can determine g_1 and g_2 using the equations:

$$g_1\lceil n_s/g \rceil + g_2\lfloor n_s/g \rfloor = n_s \qquad (2)$$
$$g_1 + g_2 = g$$

For instance, for a problem with $n_s = 10$ and $k = 4$, we have $g = 4$, $g_1 = 3$, and $g_2 = 2$. The four groups of Gray codes are: C_1 = ({0000}, {0001}, {0011}, {0010}), C_2 = ({0100}, {0101}, {0111}, {0110}), C_3 = ({1100}, {1101}, {1111}, {1110}), and C_4 = ({1000}, {1001}, {1011}, {1010}). The four groups of states are: $S_1 = (s_1, s_2, s_3)$, $S_2 = (s_4, s_5, s_6)$, $S_3 = (s_7, s_8)$, and $S_4 = (s_9, s_{10})$. In the next subsection, we will discuss how to do the group mapping from S_i to C_j ($i, j = 1, 2, \ldots, g$).

Without loss of generality, we assume all given states have been in decreasing order of weights of their associated edges. If this is not the case, they need to be sorted before grouping. The pseudo-code description of the above grouping is given below (the grouping for Gray codes is straightforward and hence not shown):

```
grouping {
  sort edges by weight in decreasing order;
  S_m ← ∅ , m = 1, 2, ..., g ;
  m ← 1;
  if (⌈n_s/g⌉ and ⌊n_s/g⌋ are equal)   {g_1 ← g}
    else {calculate g_1 and g_2 using Eq. (2)}
  foreach edge {s_i, s_j} {
    if (s_i is unselected)   {S_m ← S_m ∪ s_i};
    if (|S_m| = ⌈n_s/g⌉) or (|S_m| = ⌊n_s/g⌋ and m > g_1)
      {m ← m + 1};
    perform the above operation on s_j ;
  }
}
```

In the above procedure, the edge sorting takes $O(|E_w| \cdot \log|E_w|)$ time, and the grouping itself needs $O(|V|)$ time. Since we have $|E_w| > |V|$ for most graphs, the complexity of the grouping process is $O(|E_w| \cdot \log|E_w|)$, where $|E_w|$ is the number of edges in the weighted undirected graph G_w.

3.2 Encoding

In this phase, we do the actual encoding group by group. We start with the first state group S_1 where there are high transition probabilities between the states. The tentative group mapping is $S_1 \rightarrow C_j$, $j = 1, 2, \ldots, g$. Within the group, we obtain an exact encoding for the states in S_1 by evaluating the performance of all k^2 (at most) possible solutions. The best result from all g tentative mappings defines the final codes for the states in S_1. Then we move on to the next state group. This process continues until all states have been encoded. In order to take into account the effects due to the interconnected edges between the states in different groups, the performance of a solution is measured by a *partial switching activity*, $SW_{partial}$, which is defined to be:

$$SW_{partial}(S_{encoded}) = \sum_{s_i, s_j \in S_{encoded}} w_{i,j} H(s_i, s_j) \qquad (3)$$

where $S_{encoded}$ is the set of all states which have been encoded. More formally, the encoding procedure is described below:

```
encoding {
S_encoded ← ∅; C ← (C_1, C_2, ..., C_g);
  for i = 1, 2, ..., g {
    for all C_j ∈ C {
      SW_min ← SW_partial(S_encoded ∪ S_i);
      do the tentative mapping: S_i → C_j;
      if (SW_partial(S_encoded ∪ S_i) < SW_min) {
        SW_min ← SW_partial(S_encoded ∪ S_i);
        C_min ← C_j;
      }
    }
    C ← C \ C_min;  S_encoded ← S_encoded ∪ S_i ;
  }
}
```

Note that the above algorithm is very fast. The state encoding within each group takes $O(k!|E_w|)$ time which is proportional to $O(|E_w|)$ for a small fixed value of k. The tentative group mapping needs $O(g^2)$. Since $g \propto n_s/k$, the timing complexity of the whole algorithm is $O(n_s^2|E_w|)$, where n_s and $|E_w|$ are the number of states and the number of undirected edges, respectively, in the given state transition graph.

It is interesting to look at the two extreme cases of the proposed algorithm when $k = 1$ or $k = n_s$. If $k = 1$, the algorithm is reduced to a full Gray-encoding where the states with high transition probability are assigned to adjacent codes as much as possible. While this is a very fast encoding process, it has little capability of accounting for the interaction among the states in a complicated graph. The other extreme case when $k = n_s$ corresponds to a full exploration of solution space. It leads to an optimal solution for small graphs, but is computationally impractical for large ones. The algorithm selects the value of k to be somewhere in between, making a tradeoff between the CPU time and result quality. Our experiments on all tested benchmarks show that $k = 4$ produced the results very close to the optima.

3.3 Example

To illustrate the procedure of the proposed algorithm, we use a simple example shown in Fig. 1 where there are five states (i.e., $n_s = 5$) and three state variables are required. The grouping phase of our algorithm divides the states into two groups: (s_1, s_2, s_4) and (s_3,

s_5). The encoding phase generates the solution which is summarized in Table 1. For comparison, this table also shows the results using column-based algorithm [7] and an optimal solution. It can be seen that our algorithm achieves the switching activity of 1.35, compared to 1.65 given by the column-based algorithm. The optimum is 1.20 as shown in the last column of the table.

4. EXPERIMENTS

We have implemented the proposed semi-Gray encoding algorithm and tested its performance on the FSMs from MCNC LGSynth93 Benchmark Set (version 4.0). The test benchmarks are divided into two sets: a set of small machines (with at most 8 states) and a set of large machines (with more than 8 states). For small machines, both exact (optimal) and column-based algorithms were also implemented for comparison. For large machines, only column-based algorithm was tested since the CPU time is too long for the optimal algorithm.

The experimental results are summarized in Table 2 and Table 3, where the CPU time is based on a PC with 1.5GHz Pentium-4 processor. As we can see from Table 2, our semi-Gray encoding algorithm produces the optimum for 6 out of 9 machines, and leads to a big improvement over the column-based algorithm. Fig. 2 shows their comparison.

For the large benchmarks shown in Table 3, the semi-Gray encoding algorithm significantly outperforms the column-based algorithm. The reduction percentage ranges from 6% to 28% with an average of about 15%. This is shown in Fig. 3. Also, it can be seen that in terms of CPU time, our algorithm is the fastest for all tested machines, indicating its computational efficiency.

5. CONCLUSIONS

We have addressed the FSM state encoding problem for low power, and proposed the semi-Gray encoding algorithm to minimize the switching activity of state register. The proposed approach first partitions the states into groups, and then takes advantage of Gray codes within each group. The algorithm not only improves the results from the previous technique by an average of 15%, but also obtains a dramatic speedup. Future work includes performing further area and power optimization on combinational logic by SIS package.

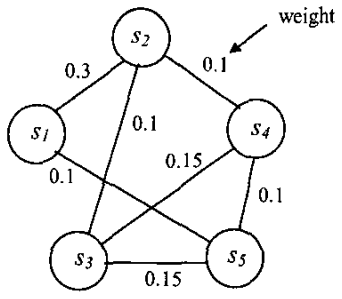

Fig. 1. An example of weighted undirected graph

	our algorithm	column-based	optimal
switching activity	1.35	1.65	1.20
state encoding	s_1: 000 s_2: 001 s_3: 011 s_4: 111 s_5: 110	s_1: 111 s_2: 101 s_3: 011 s_4: 110 s_5: 001	s_1: 011 s_2: 001 s_3: 000 s_4: 100 s_5: 010

Table 1: State encoding for Fig. 1

		switching activity			CPU time (s)		
ckt	# states	opt	column -based	semi-Gray	opt	column -based	semi-Gray
bbtas	6	0.58	0.75	0.58	19.70	0.36	0.16
beecount	7	1.08	1.23	1.08	37.50	0.28	0.21
dk14	7	1.21	1.27	1.21	36.50	0.41	0.25
dk15	4	0.97	0.97	0.97	0.02	0.22	0.08
dk17	8	1.22	1.41	1.28	37.51	0.39	0.27
dk27	7	1.29	1.71	1.29	38.47	0.29	0.19
ex6	8	1.16	1.35	1.23	38.94	0.31	0.22
lion	4	0.40	0.40	0.40	0.06	0.09	0.05
s8	5	0.65	0.90	0.75	6.31	0.28	0.17

Table 2. Performance comparison of different algorithms on small FSMs

		switching activity		CPU time (s)	
ckt	# states	column-based	semi-Gray	column-based	semi-Gray
bbara	10	0.43	0.36	0.53	0.28
bbsse	16	1.95	1.77	1.27	0.33
dk512	15	2.00	1.60	0.99	0.33
donfile	24	2.76	2.59	1.74	1.30
keyb	19	1.95	1.57	1.46	0.87
planet	48	1.99	1.63	12.55	3.58
s1488	48	2.69	2.31	18.53	4.61
sand	32	0.36	0.26	5.03	0.98
tbk	32	0.80	0.73	4.67	1.57

Table 3. Performance comparison of different algorithms on large FSMs

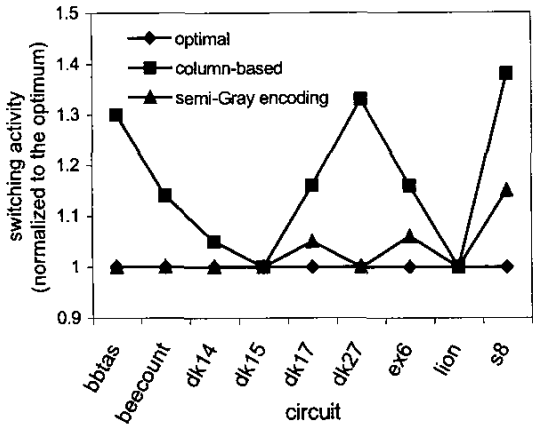

Fig. 2. Comparison of three algorithms on small FSMs

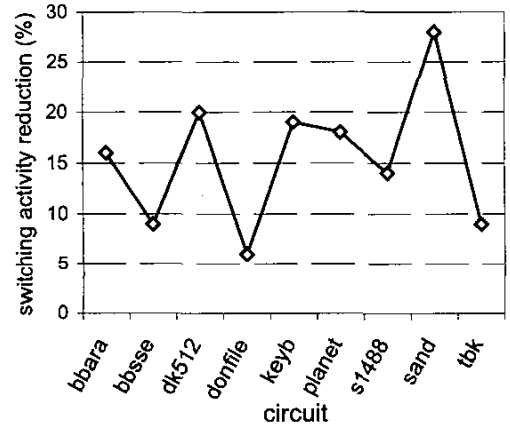

Fig. 3. The percentage improvement of semi-Gray encoding algorithm over column-based algorithm for large FSMs

REFERENCES

[1] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," IEEE Trans. on CAD, vol. CAD-4, no. 3, pp. 269-284, July 1985.

[2] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementation," IEEE Trans. on CAD, vol. 9, no. 9, pp.905-924, Sept. 1990.

[3] X. Du, G. Hachtel, B. Lin, and A. R. Newton, "MUSE: A multilevel symbolic encoding algorithms for state assignment," IEEE Trans. on CAD, vol. 10, no. 1, pp. 28-38, Jan. 1991.

[4] E. Olson and S. Kang, "State assignment for low-power synthesis using genetic local search," in Proc. of IEEE Custom Integrated Circuit Conference, pp. 140-143, May 1994.

[5] A. R. Newton, S. Devadas, Hi-keung Ma, and A. Sngiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multilevel logic implementations," IEEE Trans. on CAD, vol. 7, no. 12, pp. 1290-1300, Dec. 1988.

[6] B. Lin and A. R. Newton, "Synthesis of multiple level logic from symbolic high-level description languages," in Proc. of IFIP International Conference on VLSI, pp. 187-196, 1989.

[7] G. De Micheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros," IEEE Trans. on CAD, vol. 5, no. 4, pp. 597-616, Oct. 1986.

[8] L. Benini and G. De Micheli, "State assignment for low power dissipation," IEEE Journal of Solid-State Circuits, vol. 30, no. 3, pp.258-267, March 1995.

[9] C-Y Tsui, M. Pedram, and A. M. Despain, "Low power state assignment targeting two- and multilevel logic implementations," IEEE Trans. on CAD, vol. 17, no. 12, pp. 1281-1291, 1998.

[10] G. De Micheli, "Synthesis and optimization of digital circuits," McGraw-Hill, 1994.

[11] K. Trivedi, "Probability and statistics with reliability, queuing and computer science applications," Prentice-Hall, 1982.

[12] S-W Yeong and F. Somenzi, "A new algorithm for 0-1 programming based on binary decision diagrams," in Logic Synthesis Workshop, pp. 177-184, 1992.