

# STATE ENCODING FOR LOW POWER EMBEDDED CONTROLLERS

L. Daldoss † D. Sciuto ‡ C. Silvano †

† Università di Brescia, DEA, Via Branze 38, 25123 Brescia, Italy

‡ Politecnico di Milano, DEI, P.zza L. Da Vinci 32, 20133 Milano, Italy

## ABSTRACT

Power dissipation has become one of the main issues during embedded systems design in the recent years, due to the continuous increase of the integration level and the operating frequency. The present paper affords the problem of reducing the power dissipation in embedded controllers represented as Finite State Machines (FSMs). The aim of the work is to propose an approach to optimize state encoding for low power embedded FSMs, starting from a probabilistic description of the FSMs. Experimental results have been obtained by applying the proposed state encoding techniques to benchmark circuits. The results have also been compared to previous state encoding methods.

## 1. INTRODUCTION

An increasing number of embedded system applications has recently been characterized by low power requirements. For control dominated embedded systems, traditional synthesis techniques, targeting area and speed optimization, are no longer sufficient to meet the system level requirements in terms of power dissipation, hence several research works addressing sequential synthesis for low power have recently appeared in literature [1]. Most of the low power synthesis techniques deal with the problem of switching activity power, being the dominant part of the power dissipation in CMOS circuits [2, 3].

The present paper faces the problem of power-oriented state encoding for embedded controllers represented as synchronous FSMs. A state-of-the-art survey of state encoding techniques for low power can be found in [1]. The state encoding problem consists of finding an assignment of codewords for the FSM states targeting the minimization of a given cost function,  $C$ . Most of the proposed state assignment techniques are based on heuristic algorithms, being the encoding problem  $NP$ -complete.

In general, the aim of power-oriented state encoding is the reduction of the switching activity of the state registers. Therefore, the power-oriented cost function,  $C$ , should account for the minimization of the number of logic transitions of the state registers between two successive clock cycles. The cost function,  $C$ , should consider the sum of the Hamming distances  $H(c_i, c_j)$  between the codewords  $c_i, c_j$  being assigned to all pairs of states  $s_i, s_j$  among which a transition can occur, assuming that all the transitions between state pairs in the STG are equally probable.

A more effective state encoding technique should assign similar codewords to state pairs characterized by very high transition probability. In this case, the cost function should consider a weighted sum of the Hamming distances between the codewords [1]. The Syclop method [4] considers the conditional state transition probabilities as weight coefficients in the cost function  $C$ , while the POW3 method, proposed in [5], uses the total state transition probabilities as weight factors in  $C$ .

An important degree of freedom to be exploited during the state

assignment of FSM is the choice of the number of state variables,  $n_{var}$ . In general, we have that  $\lceil lg_2 n_s \rceil \leq n_{var} \leq n_s$ , while the target of most power-driven state encoding algorithms is to use the minimum possible number of state bits to minimize the number of registers required. However, the use of the minimum number of registers does not guarantee that the corresponding combinational circuits operate with a lower switching activity. In some cases, a fewer number of storage elements can imply more complex circuits to implement the next state logic, leading to a corresponding increase in the transition activity of the combinational part of the FSM [1].

All the above mentioned methods are based on codewords of fixed length, while methods based on variable code lengths have been investigated recently [6]. The basic idea is the application of the Huffman coding technique, where codewords with shorter length are assigned to states with higher steady-state probability. The corresponding FSM implementation requires an additional logic to gate the clock and thus disable a subset of registers, whenever the present state has a shorter code length. A power overhead is introduced by this clock-gating logic, thus a special case of this general variable length approach is proposed in [6], where just two possible codeword lengths are possible. The set of states with higher probabilities is encoded with less than  $\lceil lg_2 n_s \rceil$  bits, while the other set of states, being less probable, is encoded by more than  $\lceil lg_2 n_s \rceil$  bits.

The aim of this paper is to provide a low power state encoding technique for embedded controllers, based on a probabilistic description of the FSMs. First, we define the state encoding problem in terms of a cost function to be optimized for power dissipation, second, two heuristic algorithms are proposed to solve the state encoding problem. The paper is organized as follows. The FSM probabilistic model is recalled in Section 2, while the state encoding problem is stated in Section 3, where the proposed heuristic algorithms are detailed. Finally, experimental results derived from the application of the proposed techniques to benchmark circuits are presented in Section 4.

## 2. PROBABILISTIC MODEL FOR FSMs

In the probabilistic model to describe a Finite State Machine, we approximate the average switching activities of the nodes by using the switching probabilities (or transition probabilities) derived by modeling the FSM as a Markov chain [1, 7]. We assume the FSM description to be available in the form of a State Transition Graph (STG), where each state is represented symbolically and nothing is known on the structure of the combinational logic.

The input static signal probabilities and the input switching activity factors are supposed to be given from the system-level specifications, being derived by simulating the FSM at a high abstraction level or by direct knowledge of the typical input behavior. Furthermore, we assume to use a Zero Delay Model for the logic gates and synchronous primary inputs. Under these assumptions, we can ignore the effects of glitches and hazards on the state bit lines, therefore the switching activity of the present and next

state bit lines are equal. Moreover, the supply and ground voltage levels in the FSM are supposed to be fixed, although it is worth noting the impact of supply voltage reduction on power.

Let the FSM, composed of  $n_s$  states, described by using a STG composed of  $n_s$  nodes, corresponding to the states in the set  $S = \{s_1, s_2, \dots, s_{n_s}\}$ , and the related directed edges. The edges are labeled with the set of input configurations that cause a transition from the source state to the destination state. Considering a transition from state  $s_i$  to state  $s_j$ , we can compute the factor  $p_{ij}$ , *conditional state transition probability*, that represents the conditional probability of the transition from  $s_i$  to  $s_j$ , given that the FSM was in state  $s_i$  [5]:

$$p_{ij} = \text{Prob}(\text{Next} = s_j | \text{Present} = s_i).$$

The computation of the  $p_{ij}$ 's can be carried out as in [1], assuming totally independent primary inputs  $PI = \{x_1, x_2, \dots, x_k, \dots, x_{n_i}\}$ . The *steady-state probability*  $P_i$  of a state  $s_i$  is defined as the probability to be in the state  $s_i$  in an arbitrarily long random sequence. Computing the  $P_i$ 's implies solving the system composed of the Chapman-Kolmogorov equations and the normality condition equation:

$$P^T = P^T p$$

$$\sum_{i=1}^{n_s} P_i = 1$$

where  $P^T = (P_1, \dots, P_k, \dots, P_{n_s})$  is the row vector of the steady-state probabilities and  $p$  is the matrix of the conditional state transition probabilities  $p_{ij}$ . Note that the above system has  $(n_s + 1)$  equations and  $n_s$  unknowns, thus one of the Chapman-Kolmogorov equations can be dropped.

Given the state probabilities  $P_i$ 's and the conditional state transition probabilities  $p_{ij}$ 's, the *total state transition probabilities*  $P_{ij}$  between the two states  $s_i$  and  $s_j$  can be expressed as:  $P_{ij} = p_{ij} P_i$ . The computation of the  $P_{ij}$  requires the solution of a system of equations whose cardinality is proportional to  $n_s$  and thus it can imply some computational problems for FSMs with a large number of states. Nevertheless, symbolic techniques [8] can be successfully employed in these cases.

### 3. PROPOSED POWER-ORIENTED STATE ENCODING

Given the FSM description and the input probabilities, we compute the total state transition probabilities for each edge in the STG, by modeling the FSM as a Markov chain as shown in Section 2. Then all the unreachable states and the self-loops are eliminated from the graph. As in POW3, the STG is then transformed into a weighted undirected graph,  $G$ , by collapsing all multiple-directed edges between two states  $s_i$  and  $s_j$  into a single undirected edge whose weight  $w_{ij}$  is given by:  $w_{ij} = P_{ij} + P_{ji}$ . By considering the minimum number of state variables  $n_{var} = \lceil \lg_2 n_s \rceil$ , the problem can be formalized as finding a set of distinct codewords  $C = \{c_1, c_2, \dots, c_{n_s}\}$ , composed of  $n_{var}$ -bits, that minimizes the following cost function:

$$C = \sum_{ij} w_{ij} H(c_i, c_j)$$

The goal is to assign codewords with minimum Hamming distances to states with higher total transition probabilities. For FSMs with a large number of states, the exact solution may be unattainable, being the problem NP-complete, and the solution space to be explored is  $O(n_s 2^{n_s})$ . The Gray encoding, that guarantees that two consecutive codewords have minimum Hamming distance, can represent an optimal solution of the assignment problem for a particular class of sequential circuits, such as counters, where all transitions are equally probable.

In practice, this particular class of FSMs is characterized by the fact that a consecutive order of the nodes in the graph can be easily found and the Gray encoding can be applied successfully. The basic idea of the proposed approach is to identify a path of consecutive nodes in the graph characterized by high transition probabilities. The nodes in the identified path are characterized by a high probability to be consecutive, hence our method applies the Gray encoding to these consecutive states. In particular, two greedy methods have been devised.

In the first method (*Weighted Gray Encoding Algorithm - Depth First Version*), we sort the edges by  $w_{ij}$  in decreasing order, then we consider the  $w_{ij}$  one at a time. The edges selected at each step  $k$  are reported in the state assignment table, shown in Table I. The table columns summarize the values of  $k$ ,  $w_{ij}(k)$ ,  $s_i(k)$ ,  $s_j(k)$  and the state assignment operated at each step. At the first step ( $k = 1$ ), two codewords are assigned, while at the generic step ( $k$ ), one or two codewords are assigned.

Step	Selected $w_{ij}$	$s_i$	$s_j$	Assignment
1	$w_{ij}(1)$	$s_i(1)$	$s_j(1)$	$s_i(1) = c_0, s_j(1) = c_1$
...	...	...	...	...
$k-1$	$w_{ij}(k-1)$	$s_i(k-1)$	$s_j(k-1)$	$s_i(k-1) = c_{k-1}$
$k$	$w_{ij}(k)$	$s_i(k)$	$s_j(k)$	$s_i(k) = c_k$
$k+1$	$w_{ij}(k+1)$	$s_i(k+1)$	$s_j(k+1)$	$s_i(k+1) = c_{k+1}$
...	...	...	...	...
$n_s - 1$	$w_{ij}(n_s - 1)$	$s_i(n_s - 1)$	$s_j(n_s - 1)$	$s_i(n_s - 1) = c_{n_s - 1}$

**Table I:** State assignment table

At the first step ( $k = 1$ ), we select  $w_{ij}(1) = \max w_{ij}$  and the corresponding edge ( $s_i(1), s_j(1)$ ). Then we eliminate  $w_{ij}(1)$  from the sorted list and we select  $m_1 = \max w_{pq}$  such that  $p$  or  $q$  is equal to the index  $i$  of  $s_i(1)$  and  $m_2 = \max w_{pq}$  such that  $p$  or  $q$  is equal to the index  $j$  of  $s_j(1)$ . If  $m_1 \geq m_2$ , then  $s_i(1)$  and  $s_j(1)$  are swapped to  $s_j(1)$  and  $s_i(1)$ , otherwise they remain unchanged. Finally, we assign  $c_0$  to  $s_i(1)$  and  $c_1$  to  $s_j(1)$  and go to the next  $w_{pq}$ .

At the generic step ( $k$ ), the algorithm searches for  $w_{ij} = \max w_{pq}$  in the sorted list satisfying the condition that  $p$  or  $q$  correspond to the index  $j$  of state  $s_j(k-1)$ . If the other state  $s_p(k)$  or  $s_q(k)$  has already been assigned in the previous steps, we eliminate the now considered  $w_{pq}$  from the list and go to the next  $w_{pq}$ ; otherwise we assign the codeword  $c_k$  to the unassigned state  $s_p(k)$  or  $s_q(k)$ , we eliminate the now considered  $w_{pq}$  from the list and start again from the current top of the list.

If we reach the end of the list without finding any states that satisfy the above condition, we start back from the current top of the list and we select the  $\max w_{pq}$ . The algorithm then proceeds depending on the following conditions:

- If both states  $s_p$  and  $s_q$  have already been assigned, we eliminate the now considered  $w_{pq}$  and go to the next  $w_{pq}$ ;
- If just one of the two states has been assigned, we assign  $c_k$  to the unassigned state between  $s_p$  and  $s_q$ , then we eliminate  $w_{pq}$  and start again from the current top of the list;
- If none of  $s_p$  and  $s_q$  has already been assigned, we assign the two codewords  $c_k$  and  $c_{k+1}$  applying the same algorithm used in the first step, then we eliminate  $w_{pq}$  and start again from the current top of the list.

As an example of application of the proposed algorithm, the ex5 FSM has been selected from the MCNC benchmark suite. The ex5 has  $n_s = 9$ , hence  $n_{var} = 4$ . The binary codes corresponding to the 4-bit length codewords have been indicated as  $c_0 = 0000$ ,  $c_1 = 0001$ ,

...,  $c_{15} = 1000$ , thus considering the Gray encoding of the indexes. Table II contains the sorted list of  $w_{ij}$ , obtained by solving the Chapman-Kolmogorov equations by the MATLAB linear algebra package and assuming equally probable inputs.

w <sub>05</sub>	w <sub>56</sub>	w <sub>14</sub>	w <sub>06</sub>	w <sub>04</sub>	w <sub>03</sub>	w <sub>08</sub>	w <sub>02</sub>	w <sub>15</sub>	w <sub>17</sub>	w <sub>01</sub>
83	50	49	44	41	36	30	30	28	28	28

  

w <sub>16</sub>	w <sub>27</sub>	w <sub>07</sub>	w <sub>67</sub>	w <sub>78</sub>	w <sub>45</sub>	w <sub>37</sub>	w <sub>24</sub>	w <sub>18</sub>	w <sub>12</sub>	w <sub>38</sub>
22	21	21	21	21	20	18	15	15	15	15

**Table II:** The sorted list of  $w_{ij}$  for ex5.

Finally, Table III represents the state assignment table derived by applying the proposed algorithm to ex5.

Step	w <sub>ij</sub>	s <sub>i</sub>	s <sub>j</sub>	Assignment
1	w <sub>05</sub>	s <sub>0</sub>	s <sub>5</sub>	s <sub>0</sub> = c <sub>0</sub> , s <sub>5</sub> = c <sub>1</sub>
2	w <sub>56</sub>	s <sub>5</sub>	s <sub>6</sub>	s <sub>6</sub> = c <sub>2</sub>
3	w <sub>16</sub>	s <sub>6</sub>	s <sub>1</sub>	s <sub>1</sub> = c <sub>3</sub>
4	w <sub>14</sub>	s <sub>1</sub>	s <sub>4</sub>	s <sub>4</sub> = c <sub>4</sub>
5	w <sub>24</sub>	s <sub>4</sub>	s <sub>2</sub>	s <sub>2</sub> = c <sub>5</sub>
6	w <sub>27</sub>	s <sub>2</sub>	s <sub>7</sub>	s <sub>7</sub> = c <sub>6</sub>
7	w <sub>78</sub>	s <sub>7</sub>	s <sub>8</sub>	s <sub>8</sub> = c <sub>7</sub>
8	w <sub>38</sub>	s <sub>8</sub>	s <sub>3</sub>	s <sub>3</sub> = c <sub>8</sub>

**Table III:** State assignment table for ex5.

In the second method (*Weighted Gray Encoding Algorithm - Minimum Distance Version*), we sort the edges by  $w_{ij}$  in decreasing order, then we consider the  $w_{ij}$  one at a time. The edges selected at

each step  $k$  are reported in the state assignment table, shown in Table IV. The table columns summarize respectively the values of  $k$ ,  $w_{ij}(k)$ ,  $s_i(k)$ ,  $s_j(k)$ , the assigned state  $s_g(k)$ , the unassigned state  $s_h(k)$ , the possible state assignments and the state assignment performed at step  $k$ .

At the first step ( $k = 1$ ), we select  $w_{ij}(1) = \max w_{ij}$  and the corresponding edge ( $s_i(1), s_j(1)$ ) and we assign the two codewords:  $c_0$  to  $s_i(1)$  and  $c_1$  to  $s_j(1)$  and we go to the next  $w_{pq}$ . At the generic step ( $k$ ), the algorithm searches for  $m_1 = \max w_{pq}$  such that  $p$  or  $q$  is equal to the index  $i$  of  $s_i(1)$  and  $m_2 = \max w_{pq}$  such that  $p$  or  $q$  is equal to the index  $j$  of  $s_j(1)$ .

If  $m_1 \geq m_2$ , the weighted edge ( $s_p, s_q$ ) corresponding to  $m_1$  is selected, otherwise the edge corresponding to  $m_2$  is selected. The algorithm proceeds depending on the following conditions:

- If both states  $s_p$  and  $s_q$  have already been assigned in the previous steps, we eliminate  $w_{pq}$  from the list and the algorithm searches for the new values of  $m_1$  and  $m_2$ ;
- If just one of the two states has been assigned, the weighted edge  $w_{pq}$  is selected and the assigned and unassigned states are computed:

$$s_g = \text{assigned}(s_p(k), s_q(k))$$

$$s_h = \text{unassigned state}(s_p(k), s_q(k))$$

The possible codewords to be assigned to  $s_h(k)$  are those codewords satisfying the condition:  $\min H(s_h(k), s_g(k))$ . Thus, the algorithm assigns  $s_h(k)$  to one of these codewords. Then we eliminate  $w_{pq}$  from the list and start again from the top of the list.

If we are able to find just  $m_1$  (or  $m_2$ ), we select  $m_1$  (or  $m_2$ ) and we apply the same procedure above described. If we reach the end of the list and we are unable to find both  $m_1$  and  $m_2$ , we go back to the current top of the list and we select the  $\max w_{pq}$ .

Step	w <sub>ij</sub>	s <sub>i</sub> (k)	s <sub>j</sub> (k)	Assigned State s <sub>g</sub> (k)	Unassigned State s <sub>h</sub> (k)	Possible State Assignments	Assign.
1	w <sub>ij</sub> (1)	s <sub>i</sub> (1)	s <sub>j</sub> (1)		s <sub>i</sub> (1), s <sub>j</sub> (1)		s <sub>i</sub> (1) = c <sub>0</sub> s <sub>j</sub> (1) = c <sub>1</sub>
...	...	...	...	...	...	...	...
k	w <sub>ij</sub> (k)	s <sub>i</sub> (k)	s <sub>j</sub> (k)	s <sub>g</sub> (k) = ass (s <sub>i</sub> (k), s <sub>j</sub> (k))	s <sub>h</sub> (k) = unass (s <sub>i</sub> (k), s <sub>j</sub> (k))	s <sub>h</sub> (k) such that: min H(s <sub>h</sub> (k), s <sub>g</sub> (k))	s <sub>h</sub> (k)
...	...	...	...	...	...	...	...
n <sub>s</sub> - 1	w <sub>ij</sub> (n <sub>s</sub> - 1)	s <sub>i</sub> (n <sub>s</sub> - 1)	s <sub>j</sub> (n <sub>s</sub> - 1)	s <sub>g</sub> (n <sub>s</sub> - 1) = ass (s <sub>i</sub> (n <sub>s</sub> -1), s <sub>j</sub> (n <sub>s</sub> -1))	s <sub>h</sub> (n <sub>s</sub> - 1) = unass (s <sub>i</sub> (n <sub>s</sub> -1), s <sub>j</sub> (n <sub>s</sub> -1))	s <sub>h</sub> (k) such that: min H(s <sub>h</sub> (n <sub>s</sub> -1), s <sub>g</sub> (n <sub>s</sub> -1))	s <sub>h</sub> (n <sub>s</sub> - 1)

**Table IV:** State assignment table

Step	w <sub>ij</sub>	s <sub>i</sub> (k)	s <sub>j</sub> (k)	Assigned state s <sub>g</sub> (k)	Unassigned state s <sub>h</sub> (k)	Possible State Assignments	Assignment
1	w <sub>05</sub>	s <sub>0</sub>	s <sub>5</sub>	s <sub>0</sub>	s <sub>5</sub>	s <sub>5</sub> to c <sub>1</sub> OR c <sub>3</sub> OR c <sub>7</sub> OR c <sub>15</sub>	s <sub>0</sub> = c <sub>0</sub> , s <sub>5</sub> = c <sub>1</sub>
2	w <sub>56</sub>	s <sub>5</sub>	s <sub>6</sub>	s <sub>5</sub>	s <sub>6</sub>	s <sub>6</sub> to c <sub>2</sub> OR c <sub>6</sub> OR c <sub>14</sub>	s <sub>6</sub> = c <sub>2</sub>
3	w <sub>15</sub>	s <sub>1</sub>	s <sub>5</sub>	s <sub>5</sub>	s <sub>1</sub>	s <sub>1</sub> to c <sub>6</sub> OR c <sub>14</sub>	s <sub>1</sub> = c <sub>6</sub>
4	w <sub>14</sub>	s <sub>1</sub>	s <sub>4</sub>	s <sub>1</sub>	s <sub>4</sub>	s <sub>4</sub> to c <sub>5</sub> OR c <sub>7</sub> OR c <sub>9</sub>	s <sub>4</sub> = c <sub>5</sub>
5	w <sub>17</sub>	s <sub>1</sub>	s <sub>7</sub>	s <sub>1</sub>	s <sub>7</sub>	s <sub>7</sub> to c <sub>7</sub> OR c <sub>9</sub>	s <sub>7</sub> = c <sub>7</sub>
6	w <sub>27</sub>	s <sub>2</sub>	s <sub>7</sub>	s <sub>7</sub>	s <sub>2</sub>	s <sub>2</sub> to c <sub>4</sub> OR c <sub>8</sub>	s <sub>2</sub> = c <sub>4</sub>
7	w <sub>78</sub>	s <sub>7</sub>	s <sub>8</sub>	s <sub>7</sub>	s <sub>8</sub>	s <sub>8</sub> to c <sub>8</sub>	s <sub>8</sub> = c <sub>8</sub>
8	w <sub>37</sub>	s <sub>3</sub>	s <sub>7</sub>	s <sub>7</sub>	s <sub>3</sub>	s <sub>3</sub> to c <sub>3</sub> OR c <sub>9</sub> OR c <sub>11</sub> OR c <sub>13</sub>	s <sub>3</sub> = c <sub>3</sub>

**Table V:** State assignment table for ex5.

The algorithm proceeds depending on the following conditions:

- If both states  $s_p$  and  $s_q$  have already been assigned, we eliminate  $w_{pq}$  and go to the next  $w_{pq}$ ;
- If just one of the two states has been assigned, the weighted edge  $w_{pq}$  is selected and the unassigned states is assigned as shown before;
- If none of  $s_p$  and  $s_q$  has already been assigned, we assign the two codewords  $c_k$  and  $c_{k+1}$  at minimum distance to each other, then we eliminate  $w_{pq}$  from the list and start again from the current top of the list.

For the ex5 example (see Table V), we have  $s_i(1) = s_0$ ,  $s_f(1) = s_5$ ,  $m_1 = w_{06}$  and  $m_2 = w_{56}$ . Being  $m_2 > m_1$ , we select  $w_{ij}(2) = w_{56}$ . The assigned state between  $s_5$  and  $s_6$  is  $s_6$ , hence the possible state assignments for  $s_6$  are those assignments that satisfy the condition:  $\min H(s_5, s_6)$ , thus  $c_2$  or  $c_6$  or  $c_{14}$ . The state assignment table for ex5 is shown in Table V.

#### 4. EXPERIMENTAL RESULTS

The proposed encoding methods have been applied to some MCNC benchmarks and the best result of the two algorithms, in terms of the value of the cost function, has been reported in the third column of Table VI.

Other state assignment algorithms have been applied to the same circuits, aiming at comparing the resulting cost function values. The fourth column of Table VI reports the results obtained by applying the POW3 method, while the fifth column contains the results derived from the Huffman-style encoding. The Huffman encodings have been obtained by allowing codewords of any lengths and starting from the steady state probabilities; in this case the average switching activity has been reported.

Experimental results have shown that the proposed W\_GRAY algorithm gives an average of 4.12% better results with respect to the POW3 encoding algorithm, even if there are five cases in which it

shows a lower cost function. Moreover, the proposed W\_GRAY based algorithms show a better behavior in all cases but one with respect to the Huffman based encoding, with an average of improvement of 29.88%.

Finally, work is in progress aiming at evaluating other state encoding methods of variable lengths. In particular, a method has been devised based on the combination of the Huffman-style encoding to the proposed methods.

#### 5. REFERENCES

- [1] E. Macii, "Sequential Synthesis and Optimization for Low Power", in *Low Power Design in Deep Submicron Electronics*, NATO ASI Series, Series E: Applied Sciences, Vol. 337, Kluwer Academic Publisher, 1997, pp. 321-353.
- [2] A. P. Chandrakasan and R. W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits", *Proceedings of the IEEE*, 83 (4), 1995, pp. 498-523.
- [3] S. Devadas and S. Malik, "A Survey of Optimization Techniques Targeting Low Power VLSI Circuits", *Proceedings of 32nd IEEE Design Automation Conference*, 1995.
- [4] K. Roy and S.C. Prasad, "Syclop: Synthesis of CMOS Logic for Low Power Application", *Proceedings of ICCD'92 IEEE International Conference on Computer Design*, 1992, pp. 464-467.
- [5] L. Benini and G. De Micheli, "State Assignment for Low Power Dissipation", *IEEE Journal of Solid State Circuits*, 30 (3), 1995, pp. 258-268.
- [6] P. Surti, L.F. Chao and A. Tyagi, "Low Power FSM Design Using Huffman-Style Encoding", *Proceedings of EDTC-97, IEEE European Design and Test Conference*, 1997, pp. 521-525.
- [7] A. Papoulis, "Probability, Random Variables and Stochastic Processes", McGraw-Hill, 1984.
- [8] G. Hatchel, E. Macii, A. Pardo and F. Somenzi, "Symbolic Algorithms to Calculate Steady-State Probabilities of a Finite State Machine", *Proceedings of EDTC-94: IEEE European Design and Test Conference*, 1994, pp. 214-218.

Circuit	$n_s$	W_GRAY	POW3	%	HUFFMAN	%
bbtas	6	44.348	56.0872	+26.47	63.261	+42.65
beecount	7	51.749	52.557	+1.56	61.724	+19.28
dk14	7	119.279	119.279	0	142.268	+19.27
dk17	8	109.729	109.011	-0.65	112.998	+2.98
dk27	7	123.809	164.286	+32.69	149.405	+20.67
ex3	10	121.279	114.638	-5.48	126.854	+4.6
ex4	14	54.602	61.630	+12.87	98.168	+79.79
ex5	9	104.778	107.256	+2.37	101.504	-3.12
ex6	8	113.636	100.890	-11.22	114.593	+0.84
lion9	9	47.552	47.552	0	84.365	+77.42
mc	4	37.500	37.500	0	50.000	+33.33
modulo12	12	58.334	58.334	0	89.584	+53.57
mopus	10	40.776	40.726	-0.12	59.887	+46.86
s27	6	90.369	89.434	-1.03	106.619	+17.98
s8	5	58.621	68.966	+17.65	89.655	+52.94
shiftreg	8	68.200	62.000	-9.09	74.400	+9.09
Average				4.12		29.88

Table VI: Comparison of cost function obtained by applying several state encodings to some benchmarks.