Chapter 7

State Encoding Techniques for Low-Power FSMs

The problem of minimizing the power consumption in synchronous sequential circuits is explored in this chapter. A general theoretical framework to solve the state assignment problem for Finite State Machines (FSMs) is proposed. The environment enables us to separate the problem in two different tasks. First, we define some methods to visit the State Transition Graph (STG) and to assign a priority to the symbolic states. Second, we define some encoding techniques to associate binary codes to the symbolic states to reduce the switching activity. Based on this framework, four power-oriented state assignment algorithms have been identified. These techniques have been applied to the MCNC benchmark circuits and they have also been compared to other approaches aimed at reducing the switching activity of the state variables. Experimental results have shown consistent improvements with respect to previous low-power encoding methods.

7.1. Introduction

For control dominated embedded systems, traditional state assignment approaches are no longer sufficient to satisfy the system-level constraints in terms of power dissipation, hence several contributions in the field of low power state assignment have recently appeared in the literature [155], [14], [143], [185], [173]. Most of the low power synthesis techniques deal with the problem of dynamic power dissipation, which is the dominant contribution of the power consumption in CMOS circuits [34], [49].

The goal of this chapter is to present an approach to optimize the state encoding for low power embedded controllers, given the probabilistic model of the FSM. More in general, the state assignment problem consists of choosing the binary codes to attribute to the symbolic states, satisfying the cost metrics, through the minimization of a given cost function. The encoding problem is *NP*-complete, therefore most of the proposed state assignment techniques rely on heuristic solutions.

Within our theoretical framework, we examine a new class of algorithms aimed at reducing the switching activity of the state variables. Obviously, the minimization of the register transitions has to be combined with an appropriate implementation of the combinational logic for obtaining a global power saving, so the state encoding can be the starting point for further power optimization of the combinational part. 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, assuming the power consumption is proportional to the switching activity on the state bit lines of the machine.

Therefore, 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, by assuming that all the transitions between state pairs in the STG are equally probable. A more effective state encoding technique should assign adjacent binary codes 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 [117].

The aim of this chapter is to provide a general framework for low power state assignment. The approach consists of two different phases. First, we consider the state ordering problem by iteratively selecting the edge in the STG which maximizes a given weight function to assign a priority to the symbolic states for the successive phase of encoding. The weight of each edge in the graph basically reflects the transition probabilities between the corresponding pair of states. Second, we attribute minimum length codewords to the symbolic states, by optimizing a given cost function. The cost function aims at assigning codewords with minimum Hamming distances to states with higher transition probabilities.

The chapter is structured as follows. A brief survey of the most recent state assignment methods has been provided in Section 7.2. In Section 7.3, we formalize the methodology used to solve the state ordering problem and we investigate different encoding solutions. In Section 7.4, we describe the proposed heuristic algorithms and we analyze their effectiveness. Finally, experimental results derived from the application of these techniques to a standard set of benchmark circuits have been reported and discussed in Section 7.5, while some conclusions and future developments for the work have been reported in Section 7.6.

7.2. Previous Work on State Encoding Techniques

The state encoding problem can be stated as follows. Given the STG representing the initial FSM specification, the problem consists of finding an assignment of codewords for the FSM states targeting the minimization of a given cost function, *C*. In general, the aim of the power oriented state encoding algorithms is the reduction of the switching activity of the state registers that, if combined with an appropriate implementation of the combinational logic, can lead to a global power reduction.

Several techniques to solve the general problem of power-oriented state encoding have been presented in literature [155], [14], [143], [185], [173] and a state-of-the-art survey of state encoding techniques for low power has recently appeared in [117].

Most of state encoding techniques are based on heuristic algorithms, being the encoding problem *NP*-complete. Specific solutions can be applied to particular classes of STGs, such as the Gray encoding for STG representing sequential circuits such as counters.

For a STG representing a generic structure, the One-Hot encoding guarantees exactly two state bit transitions for each clock cycle, however it requires a number of state variables exactly equal to the number of states $(n_{var} = n_s)$, while in general $\lceil lg_2\, n_s \rceil \leq n_{var} \leq n_s$.

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. Therefore, 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. The cost function expressed by the previous equation is based on the assumption that all the transitions between state pairs in the STG are equally probable. By using the probabilistic model of FSM described in the previous section, we can derive the transition probabilities of each edge in the graph, given the input probabilities distribution.

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:

$$C = \sum_{i}^{j} W_{ij}\, H(c_i\,;\, c_j) \qquad (1)$$

where W_{ij} is the weight assigned to the edge from state s_i to state s_j in the STG.

The *Syclop* method [155] considers the conditional state transition probabilities, p_{ij}, as weight coefficients in the cost function C and it uses the minimum number of state bits $(\lceil lg_2\, n_s \rceil)$. After the computation of the conditional state transition probabilities to label all the edges of the STG, the *Syclop* method is based on the simulated annealing approach, starting from an initial random encoding.

The *POW3* state assignment method, proposed by Benini and De Micheli in [14], uses the total state transition probabilities P_{ij} as weight factors in C and it enables the designer to arbitrarily select the number of state variables n_{var}. After the computation of the P_{ij} values to label the edges of the STG, the method assumes to eliminate all the unreachable states and the self-loops. Therewith the STG is 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:

$$w_{ij} = P_{ij} + P_{ji} \quad \textit{for each } i, j \qquad (2)$$

The encoding problem is then expressed as an Integer Linear Programming (ILP) problem, whose complexity is NP-complete. Two algorithms have been proposed. The first one is the semi-exact algorithm, that is based on the notion of indistinguishability classes of states and a column-based approach to assign the binary codes to states on a bit-per-bit base. The global ILP problem is thus decomposed into a set of smaller ILP problems and it is suitable for medium size FSMs. The second algorithm is a heuristic algorithm to eliminate the exponential complexity of the semi-exact version, giving a sub-optimal polynomial time assignment applicable to large FSMs.

Another factor to consider during the state assignment for low power is circuit area, in order to reduce the capacitive loads being switched at each clock cycle. With this purpose, *POW3* can apply a modified cost function considering area-related constraints. The weight coefficient w_{ij} of each edge in the undirected graph, G, becomes a convex combination of the power and area weights:

$$w_{ij} = \alpha\, w^{P}_{ij} + (1-\alpha)\, w^{A}_{ji} \qquad (3)$$

Similarly, the *Galops* encoding method [143] issues the same cost function as *POW3,* considering a convex combination of the switching activity and the area of the combinational logic as edge weights in the STG. Then they adopt a genetic local search to perform a local optimization and in particular they assign codewords by collapsing states based on hybrid genetic algorithms. However, given the different nature of the considered quantities (number of transitions and number of literals), the goodness of the global solution is not guaranteed.

On the other hand, the *LPSA* approach [185] addresses the state assignment problem for both the *2*-level and the multi-level implementations of the next state and output logic, accounting the loading factors and the switching activities of the present state inputs. Then a simulated-annealing search strategy is adopted.

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 strategy of most power-driven state encoding algorithms consists of introducing the minimum possible number of state bits to minimize the corresponding number of registers. Nevertheless, 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 smaller 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 [117].

All the above mentioned methods rely on codewords of fixed length, while other methods based on variable code lengths have been investigated recently [173]. 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, P_i. Although the code length might be longer than $\lceil lg_2\, n_s \rceil$, the additional registers are disabled for high probable states.

The corresponding FSM implementation requires an additional logic to gate the clock and thus disable a sub-set 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 [173], in which only two codeword lengths are allowed. 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. In this way, the clock-gating logic uses only a single minterm. The states in the two sub-sets are then encoded with a fixed length encoding method.

Explain that exists best case and worst case transition activity for Huffman-based encoding.

7.3. The Framework for Low-Power State Assignment

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 Chapter 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 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 $C = \{c_1, c_2, ..., c_{nS}\}$, composed of n_{var}-bits, that minimizes the following cost function:

$$C = \sum_{ij} w_{ij}\, H(c_i; c_j) \qquad (4)$$

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, as the problem is NP-hard, and the solution space to be explored is $O\,(n_s\, 2^{ns})$. We propose to organize the state assignment process in *two* successive phases. First, we examine the state ordering problem, basically by selecting each time the edge with the largest weight in *G* to assign a priority to the symbolic states. Then, we investigate the application to the states of different encoding techniques, which minimize the adopted cost function.

7.3.1. State Ordering

Let us introduce now *three* greedy strategies to solve the state ordering problem.

Method I

The basic idea of the first approach [42] is to identify the path of consecutive nodes in the graph characterized by the highest transition probabilities. At each step we make a locally optimal choice of the maximum weight edge with respect to the arrival node selected during the previous step. More in detail, after computing the weights of the undirected edges, we sort them by w_{ij} in decreasing order, then we consider the w_{ij}'s one at a time. At the first step $(k = 1)$, two codewords are assigned, while at the generic step (k), one or two codewords are attributed.

At the first step $(k = 1)$, we select $w_{pq}(1) = max\, w_{ij}$ and the corresponding edge $(s_p(1), s_q(1))$. Then we erase $w_{pq}(1)$ from the sorted list and we select $m_1 = max\, w_{ij}$ such that i or j is equal to the index p of $s_p(1)$ and $m_2 = max\, w_{ij}$ such that i or j is equal to the index q of $s_q(1)$. If $m_1 \geq m_2$, then $s_p(1)$ and $s_q(1)$ are swapped to $s_q(1)$ and $s_p(1)$, otherwise they remain unchanged. Finally, we can assign $s_p(1)$ and $s_q(1)$ by applying a suitable encoding style and go to the next w_{ij}.

At the generic step (k), the algorithm searches for $w_{pq}(k) = max\, w_{ij}$ in the sorted list satisfying the condition that $p(k)$ or $q(k)$ correspond to the index $q(k\text{-}1)$ of the previously assigned state $s_q(k - 1)$. If the other state $s_p(k)$ or $s_q(k)$ has already been assigned in the previous steps, we eliminate the current edge from the list and go to the next w_{ij}; otherwise we attribute a codeword to the unassigned state $s_p(k)$ or $s_q(k)$, we erase the considered $w_{pq}(k)$ 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 choose the $max\, w_{ij}$. Then the algorithm proceeds depending on:

- If both states s_i and s_j have already been assigned, we eliminate the current edge and go to the next w_{ij};
- If just one of the two states has been assigned, we apply a binary code to the unassigned state between s_i and s_j, then we eliminate the present edge and start again from the current top of the list;
- If none of s_i and s_j has already been assigned, we exploit the same algorithm used in the first step, then we eliminate w_{ij} and start again from the current top of the list.

Method II

In the second method [42], we determine the visiting path by making a locally optimal choice at each step with respect to both the starting and arrival nodes selected during the previous step. In particular, we sort the edges by w_{ij} in decreasing order, then we consider the w_{ij}'s one at a time. At the first step $(k = 1)$, we select $w_{pq}(1) = max\ w_{ij}$ and the corresponding edge $(s_p(1), s_q(1))$; we assign two codewords to $s_p(1)$ and to $s_q(1)$ and we proceed.

At the generic step *(k)*, the algorithm searches for $m_1 = max\ w_{ij}$ such that *i* or *j* is equal to the index *p(k)* of $s_p(k)$ and $m_2 = max\ w_{ij}$ such that *i* or *j* is equal to the index *q(k)* of $s_q(k)$.

If $m_1 \geq m_2$, we select the weighted edge corresponding to m_1, otherwise the edge corresponding to m_2. The algorithm proceeds depending on:

- If both states $s_p(k)$ and $s_q(k)$ have already been assigned in the previous steps, we eliminate the current edge 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 $w_{pq}(k)$ is selected and a binary code is applied to the unassigned state $s_p(k)$ (or $s_q(k)$). Then we eliminate $w_{pq}(k)$ 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 top and we select the $max\ w_{ij}$. Then the algorithm works as shown before.

Method III

In the third method, we identify the path by making a locally optimal choice at each step with respect to the nodes assigned during the previous steps. More in detail, we start in the same way as case I and II, sorting the edges by w_{ij} in decreasing order and considering the w_{ij}'s one at a time. At the first step $(k = 1)$, we select $w_{pq}(1) = max\ w_{ij}$ and the corresponding edge $(s_p(1), s_q(1))$ and we assign two codewords to $s_p(1)$ and to $s_q(1)$ by applying one of the encoding methods proposed below and we go to the next w_{ij}. At the generic step *(k)*, the algorithm searches in the whole list for the maximum weight edge $m_{pq}(k) = max\ w_{ij}$ such that *p(k)* or *q(k)* is equal to the index of a state, which has been assigned in one of the previous steps.

The algorithm proceeds depending on:

- If both states $s_p(k)$ and $s_q(k)$ have been already assigned in the previous steps, we eliminate the current edge from the list and the algorithm searches for the new $max\ w_{ij}$;
- If just one of the two states has been assigned, the weighted edge $w_{pq}(k)$ is selected and a binary code is applied to the unassigned state $s_p(k)$ (or $s_q(k)$). Then we eliminate $w_{pq}(k)$ from the list and restart.

7.3.2. State Encoding

The second phase of the state assignment procedure concerns the application of an encoding technique to the states by following the priority order suggested by one of the previous spanning algorithms. The Gray encoding, which guarantees that two consecutive codewords have minimum Hamming distance, can represent an optimal solution 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. On the other hand, if we consider a general STG structure, we can exploit the One-Hot encoding to minimize the average number of transitions of the state lines. In this way, only two state variables switch when a transition occurs between two states, even if we introduce a number of state variables equal to the number of states, with an increased consumption related to registers.

For a general solution, we propose two different encoding styles, without assuming any specific STG structure and without imposing any constraint on the required state variables. In the first approach, we start building the vector V_C which contains all the possible codewords of minimum length ($n_{var} = \lceil lg_2\, n_s \rceil$); at the generic step of the graph visit we consider the edge $(s_p(k), s_q(k))$ and we apply one of the available codewords to the unassigned state $s_p(k)$ (or $s_q(k)$), satisfying the condition: $min\ H(s_p(k), s_q(k))$. In the second approach, we impose a different criterion during the assignment, since we choose, for the unassigned state $s_p(k)$ (or $s_q(k)$), the binary code that minimizes a partial cost function C_P. As local cost metric, we adopt the weighted sum of the Hamming distance considering the current edge $(s_p(k), s_q(k))$ and the edge with the largest weight which contains the unassigned state $s_p(k)$. We investigated the possibility of directly evaluating the total cost function, but the results did not confirm the effectiveness of this choice.

7.4. Proposed Power-Oriented State Assignment Techniques

In this section, we present four different state assignment methods targeting low-power consumption, although the proposed framework allows us to combine any one of the proposed state ordering algorithms with whatever encoding. The primary goal of the proposed methods consists of reducing the number of logic transitions at the state lines between two consecutive clock cycles.

Depth_First **Method**

In the method called *Depth_First*, we exploit the first ordering method to identify a path of consecutive nodes in the graph, with the highest transition probabilities. Since the nodes belonging to the selected path are characterized by a high probability to be consecutive, our method attributes the Gray encoding to the states. As an example, the `bbara` FSM has been selected from the *MCNC* benchmark suite. The `bbara` has $n_s = 10$, 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. Table 1 contains the related sorted list of w_{ij}, obtained by solving the Chapman-Kolmogorov equations by assuming equally probable inputs. The second and third columns of Table 2 represent, respectively, the priority assigned to each edge and the state encoding derived with the proposed algorithm to `bbara`.

w_{12}	w_{14}	w_{01}	w_{04}	w_{23}	w_{45}	w_{24}	w_{34}	w_{37}	w_{15}	w_{17}	w_{56}	w_{47}	w_{78}	w_{16}	w_{18}	w_{67}	w_{89}	w_{48}	w_{19}	w_{49}	w_{09}
416	412	360	219	166	153	83	83	83	61	46	30	23	23	20	11	10	5	5	2	1	1

Table 1: The sorted list of w_{ij} for `bbara`

Minimum_Distance **Method**

In the *Minimum_Distance* method, we visit the graph by choosing each time the edge with a maximum weight in the list, which contains one of the two states belonging to the previously considered edge. Therefore, we follow the second ordering approach, then we apply an available codeword that minimizes the Hamming distance between the pair of states of the current edge, as in the second encoding. The corresponding results for the `bbara` example are reported in Table 2.

Step	**Depth_First**		**Minimum_Distance**		**1_Level**		**1_Level_Tree**	
	w_{ij}	**Assignment**	**w_{ij}**	**Assignment**	**w_{ij}**	**Assignment**	**w_{ij}**	**Assignment**
1	w_{21}	$s_2 = c_0, s_1 = c_1$	w_{12}	$s_1 = c_0, s_2 = c_1$	w_{12}	$s_1 = c_0, s_2 = c_1$	w_{12}	$s_1 = c_0, s_2 = c_1$
2	w_{14}	$s_4 = c_2$	w_{14}	$s_4 = c_3$	w_{14}	$s_4 = c_{15}$	w_{14}	$s_4 = c_{15}$
3	w_{40}	$s_0 = c_3$	w_{10}	$s_0 = c_7$	w_{10}	$s_0 = c_7$	w_{01}	$s_0 = c_7$
4	w_{09}	$s_9 = c_4$	w_{15}	$s_5 = c_{15}$	w_{15}	$s_5 = c_8$	w_{23}	$s_3 = c_{14}$
5	w_{98}	$s_8 = c_5$	w_{17}	$s_7 = c_2$	w_{17}	$s_7 = c_3$	w_{45}	$s_5 = c_{12}$
6	w_{87}	$s_7 = c_6$	w_{73}	$s_3 = c_5$	w_{73}	$s_3 = c_2$	w_{37}	$s_7 = c_{13}$
7	w_{73}	$s_3 = c_7$	w_{78}	$s_8 = c_{13}$	w_{78}	$s_8 = c_{12}$	w_{56}	$s_6 = c_3$
8	w_{45}	$s_5 = c_8$	w_{76}	$s_6 = c_4$	w_{76}	$s_6 = c_{11}$	w_{78}	$s_8 = c_2$
9	w_{56}	$s_6 = c_9$	w_{89}	$s_9 = c_{10}$	w_{89}	$s_9 = c_{13}$	w_{89}	$s_9 = c_5$

Table 2: State assignment table for `bbara`

1_Level Method

The *1_Level* method follows the same procedure for the state ordering, then it applies one of the possible codewords to the unassigned state $s_p(k)$ (or $s_q(k)$), accounting for a partial cost function. We choose the binary code c_k that minimizes the weighted sum of the Hamming distance, considering the binary codes in the current edge and also in the edge characterized by the maximum weight value which contains the unassigned state, $s_p(k)$. If the w_{ij} coefficients of these two edges are the same, we prefer to introduce a smaller number of different bits between the states in the current edge. The visiting order and the performed encoding have been reported in Table 2.

1_Level_Tree Method

In the *1_Level_Tree* method, we follow the third visiting approach, since at the generic step we select the max w_{ij} such that it contains at least one state previously assigned, as shown in Table 2. For the code assignment, we adopt the partial cost function C_P.

Finally, in Table 3, we compare the Hamming distances between each pair of states in the graph encoded by several assignment methods for the `bbara` example. The final cost function values, reported in the last column, show better results for the *1_Level_Tree* method.

Hamming Distances	w_{12}	w_{14}	w_{01}	w_{04}	w_{23}	w_{45}	w_{24}	w_{34}	w_{37}	w_{15}	w_{17}	w_{56}	w_{47}	w_{78}	w_{16}	w_{18}	w_{67}	w_{89}	w_{48}	w_{19}	w_{49}	w_{09}	**Cost Function**
	416	412	360	219	166	153	83	83	83	61	46	30	23	23	20	11	10	5	5	2	1	1	
DepthFirst	1	1	2	1	1	4	2	3	1	3	1	1	2	1	2	2	1	1	1	2	2	1	3460
Min_Dist.	1	1	1	2	2	2	2	2	1	1	2	3	1	1	2	3	2	1	2	4	3	3	3090
1_Level	1	1	1	2	1	1	2	3	1	2	1	1	2	1	3	2	2	1	1	3	2	4	2834
1_Level_Tree	1	1	1	2	1	1	2	1	1	2	3	1	2	1	1	2	2	1	3	3	4	2	2730
POW3	1	1	1	2	1	1	2	1	1	2	3	1	2	1	3	2	2	1	3	3	4	2	2770
Huffman	1	1	1	2	2	1	2	1	1	2	2	1	1	1	2	2	1	1	1	2	1	2	2822

Table 3: Comparison among several state assignment methods for `bbara`

7.5. Experimental Results

The four proposed state assignment methods, as well as the *POW3* and Huffman-style encodings, have been applied to the whole set of *MCNC* benchmark circuits. The Huffman encodings have been obtained by allowing codewords of any lengths and starting from the steady state probabilities; the average switching activity has been considered. The values of the cost functions obtained by applying the different encoding techniques to the benchmarks circuits are shown in Table 4. A comparison between the *1_Level_Tree* method and the *POW3* and Huffman methods has been reported in columns seven and nine of Table 4. The values of the cost functions corresponding to the *1_Level_Tree, POW3* and Huffman methods have also been reported in Figure 1. Globally, the experimental results have shown that the proposed method gives an average of *9.92%* better results with respect to the *POW3* encoding algorithm. In all cases but one it shows a lower cost function. Moreover, the proposed *1Level-Tree* algorithm presents a better behavior in all cases with respect to the Huffman based encoding, with an average improvement of *28.65%*.

	Depth First	**Min_Dist**	**1Lev**	**1Lev_Tree**	**POW3**	**%1LevTree vs POW3**	**Huff**	**%1LevTree vs Huff**
bbara	348369214	311592835	285553300	275315936	279414298	-1.47	341275454	-19.33
bbse	876445488	781572724	775728757	781165781	781295221	-0.02	995242043	-21.51
bbtas	560869572	560869572	443478266	443478266	560869572	-20.93	632611523	-29.90
bbtasmod	736956546	795652199	766304374	678260893	795652202	-14.75	1206527406	-43.78
beecount	461300320	429566572	429566572	429566572	429566572	0.00	576431008	-25.48
cse	340663211	250017437	243140945	243173843	246243887	-1.25	363026018.5	-33.01
dk14	1246602541	1191074164	1199418369	1192786493	1192786493	0.00	1422676049	-16.16
dk15	850434786	850434786	850434786	850434786	850434786	0.00	906521531	-6.19
dk16	2056886695	1858252587	1837417250	1710844268	1836987540	-6.87	2229878388	-23.28
dk17	1197767155	1197767155	1035087727	1035087727	1035087727	0.00	1129982453	-8.40
dk27	1238095248	1190476200	1190476200	1190476200	1357142868	-12.28	1494047929	-20.32
dk512	1711309536	1398809537	1318452392	1235119058	1532738108	-19.42	1933037339	-36.10
donfile	1375000044	1395833378	1270833374	1270833374	1583333384	-19.74	1604218446	-20.78
ex1	881491114	814035082	755849869	709589759	798246811	-11.11	934473686.5	-24.07
ex4	521739142	543478273	543478273	500000010	608695663	-17.86	728261071	-31.34
ex6	1136358116	1136358116	1082571068	1071716088	1008903549	6.23	1145928498	-6.48
keyb	732586001	557527812	556511249	556515833	556521667	0.00	781923409	-28.83
kirkman	250000010	250000010	250000010	281250010	343750014	-18.18	1859375000	-84.87
lion	375000000	375000000	375000000	375000000	375000000	0.00	500000000	-25.00
lion9	444444448	444444448	444444448	444444448	722222228	-38.46	791673050.5	-43.86
mc	428571432	428571432	428571432	428571432	428571432	0.00	535715027.5	-20.00
modulo12	583333338	583333338	583333338	583333338	666666672	-12.50	895840454.5	-34.88
planet	1481424324	1432460836	1260775543	1134389349	1578854809	-28.15	2427091294	-53.26
planet1	1481424324	1432460836	1260775543	1134389349	1578854809	-28.15	2427091294	-53.26
s1	1307137517	1246467796	1111954284	1104615102	1278975255	-13.63	1709338838	-35.38
s1488	346142350	340549136	332656021	334672874	338272216	-1.06	418616028	-20.05
s1a	1307137517	1246467796	1111954284	1104615102	1278975255	-13.63	1709338838	-35.38
s208	476104317	476104317	475004342	475004342	475090165	-0.02	562499008.5	-15.55
s27	976190484	998599447	897759111	897759111	897759111	0.00	1065300599	-15.73
s386	876445485	781572721	775728754	781165778	781165778	0.00	995241966.5	-21.51
s420	476104317	476104317	475004342	475004342	475090165	-0.02	562499008.5	-15.55
s510	716981169	716981169	716981169	716981169	1094339681	-34.48	1537742082	-53.37
s8	146551728	146551728	146551728	146551728	172413798	-15.00	224135990.5	-34.61
s820	585868299	555052156	548511763	548556122	553705440	-0.93	737975540.5	-25.67
s832	584316521	554029118	549546154	549546163	553704019	-0.75	737975540.5	-25.53
sand	710048745	669127075	602552419	617787022	754698569	-18.14	1071852990	-42.36
shiftreg	1375000000	1000000000	1000000000	1000000000	1375000000	-27.27	1500000000	-33.33
styr	672352456	576434981	554459041	570323162	570503426	-0.03	762730980.5	-25.23
tav	1000000000	1000000000	1000000000	1000000000	1000000000	0.00	1500000000	-33.33
tbk	1134317126	1078203754	1098476979	1069153210	1087616330	-1.70	1228897992	-13.00
tma	262041291	224385038	195066485	190590292	219740721	-13.27	263484507	-27.67
train11	583333350	458333346	458333346	458333346	500000014	-8.33	541661053.5	-15.38
train4	400000001	400000001	400000001	400000001	600000001	-33.33	600000000	-33.33
Average %						**-9.92**		**-28.65**

Table 4. Comparison of cost functions obtained by applying several state encoding techniques to MCNC benchmarks

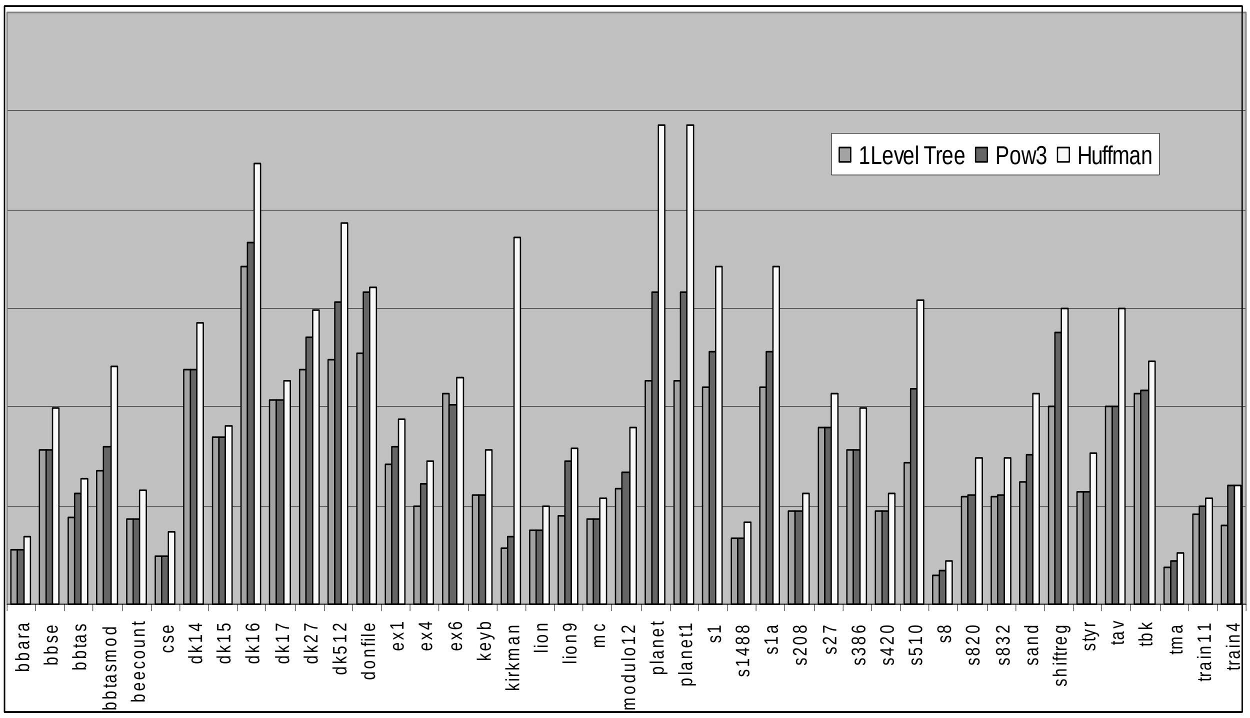

Figure 1: Comparison of cost functions obtained by applying several state encoding techniques to MCNC benchmarks.

7.6. Summary

This chapter approached the problem of low-power state encoding for sequential circuits. In particular, after a review of the most recent works related to the state assignment of FSMs for power optimization, the chapter has been devoted to propose a general framework to support the problem of finding a state assignment to minimize the power consumption. The theoretical framework aims at reducing the switching activity of the state variables. Based on this framework, we proposed four different state assignment techniques. Experimental results have shown the effectiveness of the proposed methods. More in detail, the cost functions obtained by the best of the proposed methods provides an average power saving of *9.92% 28.65%* with respect to the *POW3* and the Huffman-based encoding algorithms. Finally, work is in progress aiming at evaluating other state encoding methods. We are investigating a step one backward method for the state ordering process, while we are focusing on variable length codes to be applied during the successive encoding process.