Spanning Tree Based State Encoding for Low Power Dissipation

Winfried Nöth and Reiner Kolla
Lehrstuhl für Technische Informatik, Universität Würzburg, 97070 Würzburg, Germany

Abstract

In this paper we address the problem of state encoding for synchronous finite state machines. The primary goal is the reduction of switching activity in the state register. At the beginning the state transition graph is transformed into an undirected graph where the edges are labeled with the state transition probabilities. Next a maximum spanning tree of the undirected graph is constructed, and we formulate the state encoding problem as an embedding of the spanning tree into a Boolean hypercube of unknown dimension. At this point a modification of Prim's maximum spanning tree algorithm is presented to limit the dimension of the hypercube for area constraints. Then we propose a polynomial time embedding heuristic, which removes the restriction of previous works, where the number of state bits used for encoding of a k-state FSM was generally limited to $\lceil \log_2 k \rceil$. Next a more sophisticated embedding algorithm is presented, which takes into account the state transition probabilities not covered by the spanning tree. The resulting encodings of both algorithms often exhibit a lower switching activity and power dissipation in comparison with a known heuristic for low power state encoding.

1. Introduction

The synthesis of circuits with reduced power consumption has grown more and more important over the last years. One driving force behind low power circuit design is the demand for longer battery life of portable computers and telecommunication equipment. Another one results from the excessive power consumption of high performance micro processors, which is currently the limiting factor in integration density of single- and multi-chip modules. This power consumption often leads to reliability problems due to overly high operating temperatures. The growing need for high performance computers however can be expected to further raise the importance of power related research in the future.

Research activity on low power circuit design is widespread and ranges from voltage scaling and process optimization to high level approaches like instruction set designs and hardware/software codesign. This paper focusses on minimizing the power consumption of synchronous finite state machines (FSMs), which form an important part of many VLSI products. Since the circuit realization of a FSM is mostly determined by the state encoding, the encoding can be justly assumed to have a great influence on power dissipation. Our primary goal is the reduction of switching activity in the state register, but we will show that our encodings often also lead to a reduced overall power dissipation of the circuits generated by SIS [10].

Research on FSM state encoding was first targeted at minimization of circuit area and delay. For two level circuits De Micheli et al. devised algorithms for symbolic minimization and bit minimal state encoding [4], while Devadas et al. [5] developed the MUSTANG state assignment system targeting multilevel networks. Power related research was first aimed at precise computation of switching activity in sequential circuits [9][11]. Since then several low power state encoding algorithms have been proposed. Tsui et al. [12] integrated cost functions for state register and transition logic activity, while Benini et al. [1] developed algorithms trading off accuracy vs. computational complexity. Chen et al. [2] already formulated the encoding problem as a hypercube embedding problem. Common to these and other approaches however is the limitation to a predetermined number of bits for the state encoding, which will be removed in this paper. We have just received notice that simultaneously to this work Molitor et al. [6] developed a similar state encoding algorithm targeting the size of the BDD representation.

The paper is organized as follows. Section 2 contains an examination of power related issues in FSM synthesis. In section 3 we describe the connection of state encoding and hypercube embedding and present a modification of Prim's algorithm for spanning tree computation as well as two algorithms for spanning tree directed state encoding. Section 4 contains results and conclusions.

2. Power dissipation in FSMs

FSMs are representations of sequential boolean functions. They are conveniently described by a state transition graph (STG), where nodes represent the states, and directed edges, labeled with inputs and outputs, describe the transition relation between states. When implemented in hardware, FSMs

generally are realized by an architecture shown in figure 1. Each state corresponds to a binary vector stored in the state register. The combinational logic computes the next state and output function based on the current state and input values. The binary values of the inputs and outputs of a FSM are usually determined by external requirements, while the state encoding is left to the designer.

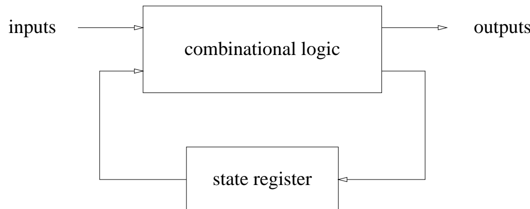

Fig. 1: FSM hardware realization

In a sequential circuit of this type, power is dissipated in the state register as well as in the combinational logic. This primarily results from changing values of circuit signals, where capacitances are charged or discharged (dynamic power dissipation).

The dynamic power dissipation in the combinational part of the circuit is very difficult to estimate, even after the state encoding is determined. At the beginning there are already several different realizations to choose from, depending on what kind of technology will be used. Later, when the gate level implementation is known, the exact computation of the dynamic power dissipation including glitches is often intractable, since it requires the examination of all possible pairs of input patterns of the combinational logic. Due to these difficulties the research in this paper focusses on the minimization of the expected state register switching activity described below. This approach still leads to a good circuit in terms of power consumption, if a low switching activity in the latch outputs corresponds to a low switching activity in the combinational logic.

The average dynamic power dissipation of the state register P_{sb} can be described by following expression:

$$P_{sb} = \frac{1}{2} V_{dd}^2 \times f \times \sum_{i \in sb} C(i)\ E(i)$$

where f is the clock frequency of the state machine, $C(i)$ is the capacitance of the latch storing state bit i and $E(i)$ is the expected switching activity of the latch. Notice, that $C(i)$ is not necessarily the same for all bits, since it includes the capacitance of the latch fanout into the combinational part. Since this fanout cannot be determined before the state encoding is known, we simplify by assuming an overall state register capacity C_{sr} and introduce an expected register switching activity E_{sr}:

$$P_{sb} \approx \frac{1}{2} V_{dd}^2 \times f \times C_{sr} \times E_{sr}$$

Let $\mathcal{S}$ be the set of all states. For an infinitely long series of state transitions, E_{sr} can be expressed by

$$E_{sr} = \sum_{i,j \in \mathcal{S}} p(i \leftrightarrow j)\ h(i,j) \qquad (1)$$

where $p(i \leftrightarrow j)$ is the probability of a transition between states i and j, and $h(i,j)$ is the hamming distance of the state codes of i and j. $p(i \leftrightarrow j)$ can be determined stochastically by assuming equiprobability of all input patterns of the FSM and solving the Chapman Kolmogorov equations [3]. Alternatively they can be obtained statistically by applying a sufficiently long series of input patterns until the state occurrence and transition probabilities converge towards discrete values [9].

For the purpose of minimizing (1), a FSM representation is sufficient, which contains only the state to state transition probabilities. We will therefore transform the initial STG by collapsing all directed edges between any pair of states into an undirected edge. The undirected edges are then weighted with their corresponding transition probability. Since the probabilities concerned are unconditional, the sum of all edge weights including self loops equals one. From now on this undirected weighted graph will be referred as the *probability attraction graph* (PAG, figure 2).

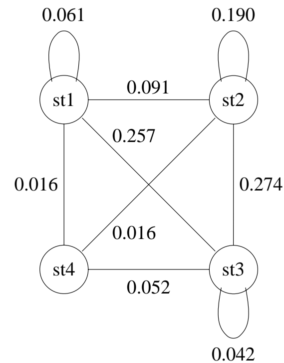

Fig. 2: PAG of `dk15`

3. Hypercube embeddings

A state encoding can always be formulated as an embedding of the STG or PAG into a (Boolean) hypercube. A *hypercube* of dimension n is a graph with 2^n nodes, where every node is labeled with an unique binary value from 0 to $2^n - 1$. Furthermore, every node v has n edges labeled $1 \ldots n$, which lead to all nodes whose labels have hamming distance 1 from v. Consequently, the hamming distance of any two nodes in the hypercube equals the length of the shortest path between the nodes. An *embedding* of a graph G into a host graph H is an injective mapping of the nodes of G to the nodes of H, so that every edge in G corresponds to the shortest path between the mappings of its terminal nodes in H. The *dilation* of an edge of G is defined as the

length of the corresponding path in H. More detailed information on embedding problems can be found in works on parallel computing [8][13]. Since most problems concerning embeddings of general graphs are NP complete, we will only give an informal discussion on what could be a good hypercube embedding with respect to power dissipation:

(A) Regarding equation (1) it is desirable to embed with small dilation, since the dilation of an edge (v, w) corresponds directly to the hamming distance of the encodings of v and w, and the hamming distance determines the register switching activity for a given state transition. The best obvious solution for our purpose would be an embedding with dilation 1 for all edges. Graphs with such an embedding are called *cubical*. Unfortunately it can be shown that there is no dilation-1 embedding for many graphs (e.g. graphs containing odd cycles). Furthermore, the problem of finding an embedding with minimum overall dilation is NP complete for general graphs [13]. For most cubical graphs it is also difficult to determine their *cubical dimension*, which is the dimension of the smallest hypercube, where they can be embedded with dilation 1.

(B) Regarding overall power dissipation, it is also desirable to embed into a hypercube with low dimension, since the dimension of the hypercube corresponds to the number of bits of the state encoding, and unnecessary large state registers may increase power consumption. Area restrictions may even limit the number of state bits available for encoding. Without boundaries to dilation it is always possible to embed a graph with k nodes into a hypercube of dimension $\lceil \log_2 k \rceil$. Unfortunately this will often lead to a high state register switching activity for many state transitions.

While most research has concentrated on B with A as a side issue, we will try to find a solution to A, with B as a secondary criterion. For that we have to embed G into a hypercube H, so that dilation of edges of G with high weight is minimized while the dimension of H is kept small. Obviously it is infeasible to optimally embed an arbitrary graph $G(V, E)$, but the problem can be simplified by embedding a subgraph $G'(V, E')$, so that dilation > 1 occurs only on edges (v, w) with $(v, w) \in E$ and $(v, w) \notin E'$. That is, if we construct a cubical subgraph of G, which contains the edges with the highest weights, a dilation-1 embedding of this subgraph would intuitively lead to a low switching activity in the state register. Such a subgraph is the maximum spanning tree of the PAG.

SPANNING TREES

Let $G(V, E)$ be a weighted connected graph. A *spanning tree* of G is a subgraph $T(V, E')$ of G, so that T is connected and $\#E' = \#V - 1$. Let $\mathcal{T}$ be the set of all spanning trees of G. A *maximum spanning tree* $T_{\max}(V, E_{\max})$ of G is a spanning tree, so that $\forall\, T(V, E') \in \mathcal{T}$:

$$\sum_{(v,w)\in E_{\max}} W(v, w) \geq \sum_{(v,w)\in E'} W(v, w)$$

A maximum spanning tree can be constructed in time $O(\#E \log \#V)$ e.g. by Prim's algorithm [7], and it is unique by construction, if no two edges of G have the same weight. Furthermore, all trees are cubical, and while the exact determination of their cubical dimension cd is NP hard, some lower and upper bounds of cd are known. Let $T(V, E)$ be a tree and let k be the maximum degree of any node $v \in V$. Then

$$\max(\lceil \log_2 \#V \rceil, k) \leq cd(T) \leq \#V - 1$$

Both lower and upper bounds of this inequation are attained by certain types of trees. A path graph T_P of length $\#V - 1$ e.g. can be embedded by a Gray code with $cd(T_P) = \lceil \log_2 \#V \rceil$, while the star graph T_S with $k = \#V - 1$ for the center node has $cd(T_S) = k = \#V - 1$. Since the dimension of the embedding is strongly connected to the degree of nodes in the tree, we have modified Prim's algorithm to accept a parameter $d_{\max}$ limiting the degree of any node in the resulting spanning tree:

```
modified_prim(graph G(V,E), int d_max)
{
  V_T := {one initial v_init ∈ V}
  E_T := ∅
  cut := {(v_init, w) ∈ E}
  do #V − 1 times
    (u,v) := select_edge(cut, d_max)
      where u ∈ V_T, v ∈ V\V_T
    E_T := E_T ∪ (u,v)
    V_T := V_T ∪ v
    remove edges containing v from cut
    cut := cut ∪ {(v,w) | w ∈ V\V_T}
  return T(V_T, E_T)
}
```

In the original version, `select_edge` simply selects the edge with the highest weight from the cut. The new procedure selects the highest weighted edge (u, v), which does not increase the degree of the node $u \in V_T$ above $d_{\max}$, if this is possible. Subsequent tests with our embedding algorithms showed, that for $d_{\max} = \log_2 \#V + 1$ all benchmarks could be embedded into a hypercube of dimension $2 \log_2 \#V$ or less with no significant penalty in switching activity. The resulting spanning tree, which in most cases is a maximum spanning tree, proved to be a good structure for directing a hypercube embedding.

TREE EMBEDDINGS

For a given tree $T(V, E)$ our embeddings begin always at subsets of nodes and edges V_C, E_C of T, which form the center of the tree with respect to longest paths. This is a direct way to construct Gray code embeddings with logarithmical dimensions for simple paths. Besides, for a well balanced tree the subtrees connected by the center of the tree can be embedded with about the same dimension, so that a

divide and conquer approach should find an embedding of low overall dimension. V_C and E_C are defined as follows:

Let for $p = v_0, \ldots, v_k$ path of T be $\lambda(p) = k$ the number of edges on p. Then

$$V_C{}^p = \{v_{\lfloor \frac{k}{2} \rfloor}, v_{\lceil \frac{k}{2} \rceil}\}$$

is the set of nodes in the center of p. We now define the center of the tree as

$$V_C := \bigcup_{p:\lambda(p)=k \text{ maximum}} V_C{}^p$$

The center of the tree has the following property:

$$V_C = \bigcap_{p:\lambda(p)=k \text{ maximum}} V_C{}^p$$

Proof: It is sufficient to prove the expression for any pair of longest paths.

Assume there are two longest paths $p = v_0 \ldots v_k$ and $q = u_0 \ldots u_k$. The size of the path centers exclusively depend on k, so that:

$$\#V_C{}^p = \#V_C{}^q$$

Assume further, that there are nodes $v_i \in V_C{}^p$ and $u_j \in V_C{}^q$, with $v_i \notin V_C{}^q$ and $u_j \notin V_C{}^p$. Let c be the path between v_i and u_j. Then there are subpaths p', q' of p, q of maximum length, which end in v_i, u_j, so that following equations hold:

$$\begin{aligned} p' \cap q' &= \emptyset \\ p' \cap c &= \{v_i\} \\ q' \cap c &= \{u_j\} \end{aligned}$$

We will now show, that

$$\lambda(p' \circ c \circ q') = \lambda(p') + \lambda(c) + \lambda(u') > k \qquad (2)$$

The equation on the left is true, since c has one node in common with each p' and q'. For the inequation on the right there are two different cases to be examined:

1. $p \cap q = \emptyset$: In this case $\lambda(p') = \lambda(q') = \lceil \frac{k}{2} \rceil$, because the longest subpaths of p, q to a node in their path centers contain at least half of the edges of p, q. Since $\lambda(c) \geq 1$, here (2) is valid. See figure 3 for illustration.

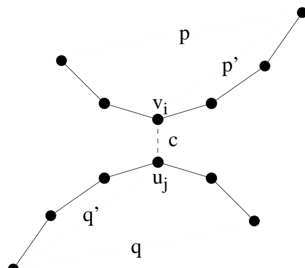

Fig. 3

2. $p \cap q = V' \neq \emptyset$: In this case c contains a node $v' \in V'$, therefore $\lambda(c) \geq 2$. Since at least one of the subpaths from a terminal node of p, q to v_i, u_j does not contain nodes from V', $\lambda(p') = \lambda(q') = \lfloor \frac{k}{2} \rfloor$, and (2) is valid again. This is illustrated in figure 4.

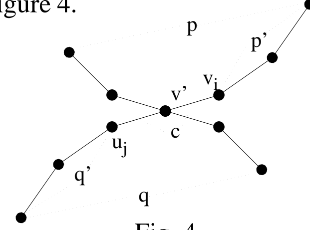

Fig. 4

Expression (2) already implies, that neither p nor q are longest paths, if they have different centers. ■

E_C is defined as the set of edges in the center of longest paths. Here we have to distinguish between two cases for the length k of longest paths:

1. k is *even*: We know from our definition above, that $\#V_C = 1$. E_C is now defined by

$$E_C := \{(v, w) \mid v \in V_C \ \wedge \ \exists\, p = v, w, \ldots, u : \lambda(p) = \frac{k}{2}\}$$

2. k is *odd*: Here $\#V_C = 2$ and consequently

$$E_C := \{(v, w) \mid v \in V_C \ \wedge \ w \in V_C \ \wedge \ v \neq w\}$$

V_C and E_C can be efficiently computed by a single pass over the tree: Our algorithm iteratively removes the set of leafs from the tree, until $\#V \leq 2$. V_C remains unchanged during this operation, since

1. no leaf is in V_C, if $\#V > 2$, and

2. any longest path looses both terminal nodes.

therefore the path centers do not move. After the last iteration, V_C consists of the nodes remaining in V, and E_C is either the final edge in E or the set of edges, which were removed last from the tree:

```
get_tree_center(T(V,E), E_C, V_C)
{
   V_leafs := {v ∈ V | degree(v) = 1}
   E_leafs := ∅
   while #V > 2
      E_leafs := {(v,w) ∈ E | {v,w} ∩ V_leafs ≠ ∅}
      V := V \ V_leafs
      E := E \ E_leafs
      V_leafs := {v ∈ V | degree(v) = 1}
   V_C := V
   if #E > 0
      E_C := E
   else
      E_C := E_leafs
}
```

We will now present polynomial time divide and conquer algorithms, which construct dilation 1 embeddings of trees into a hypercube by dividing the trees at the center defined above.

FAST EMBEDDING ALGORITHM

We have shown that the removal of an edge of E_C breaks up a longest path at or near it's center, leaving two subtrees of unknown size and structure. Since both subtrees are to be embedded recursively, it would be best to balance the subtree embeddings with respect to dimension to minimize the dimension of the overall embedding. It is however difficult to determine in advance, what dimension a subtree embedding will have. Our first algorithm therefore selects an edge $(v, w) \in E_C$, whose removal from E leads to the most evenly sized subtrees T' and T'' with respect to the number of edges of the subtrees. Then an index $i \in \mathbb{N}$ corresponding to the edge label in the host hypercube is assigned to (v, w), which means that the labels of the nodes connected by e differ exactly in position i. Now (v, w) is removed from the tree, splitting it up into two subtrees. The algorithm then recursively processes the subtrees T' and T'':

```
embed_tree_fast(tree T(V,E), int i)
{
  get_tree_center(T, V_C, E_C)
  select edge (v,w) ∈ E_C
    connecting most balanced subtrees
  (v,w).idx := i
  remove (v,w) from E
  get subtrees T'(V',E'), T''(V'',E'')
  if #E' > 0
    embed_tree_fast(T', i+1)
  if #E'' > 0
    embed_tree_fast(T'', i+1)
}
```

The algorithm starts with `embed_tree_fast(T,1)` and it terminates, when all edges have indices assigned. Overall runtime is of the order of $O(\#V^2)$ of the original graph, since the procedure is called once for every edge, while an efficient implementation of `get_tree_center` runs in $O(\#V)$. To derive an encoding from the embedded tree a code is first assigned to one of the nodes. Then the codes of adjacent nodes are computed by toggling the bit adressed by the index of the connecting edge. This procedure is iterated, until all nodes are encoded. Since there are no cycles, the overall encoding is uniquely determined by the code of any node.

By incrementing the index parameter i during the recursion, the embedding procedure ensures, that for any depth the index of the edge (v, w) selected from E_C is not again used in one of the subtrees connected by (v, w). This ensures, that any path from one subtree to another traverses at least one edge with an unique label i. Therefore all state codes of nodes in different subtrees differ at least in position i. Since any pair of nodes is somewhere in the recursion splitted up and assigned to different subtrees, the encoding is injective.

GREEDY EMBEDDING ALGORITHM

The above algorithm is very fast, since it only embeds edges from the spanning tree without regard to costs from other edges in the PAG. Our second algorithm tries to take into account those edges of lower weight, too. This however is only possible between nodes, which are connected by a path of already embedded edges in T. The new procedure therefore always maintains a region of encoded nodes V_{enc}, which are connected by edges with known indices. V_{enc} is initialized with a node $v \in V_C$ of the center of the overall tree. Every time an index is assigned to an edge (v, w), where one of the nodes, say v, is already encoded, the encoded region is expanded by w and others, which are connected to w by previously embedded edges:

```
embed_tree_greedy(tree T(V,E), int i)
{
  get_tree_center(T, V_C, E_C)
  if V_enc = ∅
    v_init := v ∈ V_C
    v_init.code := 0
    V_enc := {v_init}
  for all (v,w) ∈ E_C
    (v,w).idx := select_index(T, V_enc, i)
    i := i+1
    remove (v,w) from E
    expand_region(V_enc)
  for all T'(V',E') subtree of T
    if #E' > 0
      embed_tree_greedy(T', i)
}
```

There are two further main differences to the fast algorithm. First, the index to be assigned to an edge is determined by a special greedy procedure `select_index`, which is described below. Second, all edges in E_C are embedded within the actual call without selection. This usually leads to a faster growth of the encoded region in comparison with the previous algorithm. For the same reason the subtrees are processed in the final loop, so that subtrees containing encoded nodes are embedded first. This also immediately increases the encoded region, the size of which determines the accuracy of the index selection, as follows:

For an edge (v, w) not connected to the encoded region V_{enc} `select_index` simply returns the maximum index value, which corresponds to the edge index assigned in the fast embedding algorithm. If however (v, w) is connected to V_{enc}, the node $w \notin V_{\text{enc}}$ can be encoded, and `select_index` for all possible indices i computes the new code from the code of v by toggling bit i. Then the switching

costs between w and all other encoded nodes are determined from the product of hamming distance and transition probability from the PAG. If the actual i leads to a code that is already used for another node, cost is set to ∞. Finally the index leading to the lowest switching costs among the encoded nodes is selected:

```
int select_index(edge (v,w), node_set
                         V_enc, int i_max)
{
  if v ∉ V_enc and w ∉ V_enc
    return i_max
  \\ assume v ∈ V_enc and w ∉ V_enc
  for all i ∈ {1,...,i_max}
    w.code := v.code toggled in ith bit
    cost(i) := 0
    for all u ∈ V_enc
      if u.code = w.code
        cost(i) := ∞
      if (u,w) ∈ attraction graph
        cost(i) :=cost(i) + h(u,w) × W(u,w)
    if cost(i) <cost(i_min)
      i_min := i
  return i_min
}
```

The greedy algorithm is considerably slower than the fast one, since for every edge $O(\#V)$ indices have to be tested, and for every index all codes from the subset of encoded nodes may be examined. It is however still polynomial, and feasible at least for medium sized problems, since all benchmark embeddings were generated within few minutes.

4. Results

We have run our **fast** and **greedy** encoding algorithms on a set of MCNC FSM examples in kiss2 format. As a reference the $O(\#V\#E)$ encoding algorithm **pow3** presented by Benini [1] was selected, since it computes a minimum length encoding targeting low state register activity in a comparable small runtime. The results are summarized in table I. Columns 1 and 2 contain the circuit name and the number of states after elimination of unreachable and unleavable states. The following six columns in groups of three present information about state register size and expected register activity for each of the algorithms tested. The final three columns display the estimated power consumption in μW of the circuit generated by SIS after extraction of sequential don't cares and optimization with *script.rugged* [10]. The runtimes of the three encoding algorithms are not printed, since being in the order of seconds or minutes they were always dominated by the preceding Chapman Kolmogorov and the following sequential optimization script. An asterisk is printed where the optimization script did not terminate within several hours.

As we can see, for 28 out of 44 examples the expected switching activity E_{sr} is reduced with respect to **pow3** either by the fast algorithm or by the greedy algorithm, and it is increased by both in only three cases. Reduction varies from 2% to 28% with an average of about 13 percent, while the state register size is always kept at or below $2\log_2 \#V$. Sometimes a reduction in switching activity is achieved without a penalty in the size of the state register. For the power consumption after synthesis, as estimated by SIS, the results can be summarized as follows. Out of 37 circuits, where SIS optimization terminated for all encodings, 26 of our best circuits are better in terms of power consumption than those encoded by **pow3**, while only six are worse. Here improvement varies from 1% up to 29% with an average of about 17 percent. It is also shown in table I, that for 26 out of 37 circuits improvements were achieved either for both register switching activity and power consumption or for neither of them. This as well as many particular results confirm our assumption, that register switching activity and power consumption are highly correlated. Comparison of state register size and power consumption however reveals no clear correlation. Sometimes circuits with larger state register dissipate more power, but there are also cases, where the overall power dissipation is reduced despite a larger state register *and* higher switching activity in the state register. This probably comes from larger sequential don't cares leading to better optimization by SIS.

This paper addressed the FSM state assignment problem targeted towards low power dissipation. The problem was formulated as a hypercube embedding problem, where the embedding process is directed by a maximum spanning tree of the probability attraction graph of the FSM. We proposed a modification of Prim's algorithm to limit the degree of the spanning tree and along with it the dimension of the embedding. Then two different state embedding algorithms were presented, which in about two out of three cases produced encodings with lower switching activity *and* power consumption than a known heuristic of comparable complexity. Due to the polynomial runtimes of our algorithms they are applicable to many large FSMs, where the states can be explicitly enumerated. It is also worth mentioning, that the proposed heuristics can be used for any state encoding problem, where the costs can be described by an edge weight function of the STG.

References

[1] L. Benini and G. DeMicheli: State Ass. for Low Power Diss. *IEEE Journ. on Solid State Circ.*, 11(4): 32-40, March 1994

[2] Chen, Sarrafzadeh, Yeap: State Enc. of FSMs for Low Power Design. To appear in VLSI Design.

[3] D.R. Cox and H.D. Miller: *The Theory of Stochastic Processes.* Chapman Hall, 1965

[4] DeMicheli, Brayton, Sangiovanni: Optimal State Ass. for FSMs. *IEEE Trans. on CAD*, 4(3): 269-284, July 1985

circuit	#V	#bits			E_{sr}			SIS power (μW)		
		pow3	fast	greedy	pow3	fast	greedy	pow3	fast	greedy
bbara	10	4	4	4	0.30	0.29	0.28	155	146	146
bbsse	13	4	5	6	0.86	0.78	0.77	339	320	340
bbtas	6	3	3	3	0.44	0.44	0.44	98	94	97
beecount	7	3	3	4	0.48	0.48	0.47	198	214	192
cse	16	4	6	7	0.29	0.24	0.24	313	332	331
dk14	7	3	3	3	1.24	1.11	1.11	501	455	443
dk15	4	2	3	3	0.85	0.83	0.83	406	391	391
dk16	27	5	6	7	1.84	1.81	1.67	1176	1119	1063
dk17	8	3	4	4	1.04	1.04	1.04	333	308	304
dk27	7	3	3	3	1.19	1.36	1.19	182	207	180
dk512	14	4	5	5	1.48	1.37	1.19	360	348	318
donfile	24	5	5	7	1.25	1.42	1.33	506	593	615
dvram	35	6	6	10	1.03	0.93	0.88	398	406	343
ex1	20	5	5	7	1.23	1.22	1.20	480	451	528
ex3	9	4	4	4	1.49	1.28	1.20	344	326	320
ex4	13	4	4	5	1.04	0.96	0.96	183	241	222
ex5	8	3	3	3	1.13	1.13	1.13	300	300	300
ex6	7	3	4	3	1.01	1.03	1.01	388	387	422
ex7	9	4	4	4	1.00	1.00	1.00	171	152	146
fetch	26	5	5	8	1.19	1.38	1.11	410	457	362
keyb	19	5	8	9	0.65	0.56	0.56	392	528	455
kirkman	16	4	4	7	0.58	0.61	0.61	501	500	397
lion	4	2	2	2	0.40	0.40	0.40	87	87	86
lion9	9	4	4	4	0.80	0.64	0.64	144	141	149
log	17	5	5	8	0.87	0.93	0.76	294	327	*
mark1	13	4	6	6	0.93	0.97	0.95	263	274	319
mc	4	2	2	2	0.43	0.43	0.43	88	88	88
nucpwr	29	5	6	10	1.20	1.16	1.04	506	445	389
opus	10	4	5	5	0.71	0.71	0.71	248	222	239
planet	48	6	8	9	1.30	1.16	1.10	*	*	*
ram_test	72	7	7	9	0.89	0.80	0.76	910	*	*
rie	29	5	5	8	0.86	0.77	0.72	376	366	414
s1	20	5	5	8	1.33	1.28	1.12	*	*	721
s8	5	3	3	3	0.59	0.69	0.59	156	180	163
sand	32	5	8	9	0.66	0.61	0.57	1278	1267	*
scf	115	7	12	11	0.85	0.87	0.85	1260	997	901
shiftreg	8	3	3	4	1.25	1.00	1.25	190	164	164
sse	13	4	5	6	0.86	0.78	0.77	339	320	331
styr	30	5	6	6	0.59	0.58	0.55	*	*	*
sync	52	6	6	9	0.88	0.74	0.74	720	536	577
tav	4	2	2	2	1.00	1.00	1.00	158	158	158
tbk	32	5	8	10	1.14	0.98	0.99	*	*	1121
train11	11	4	4	6	0.79	0.79	0.57	253	222	191
train4	4	2	2	2	0.47	0.47	0.47	74	74	74

Table I

[5] Devadas, Ma, Newton, Sangiovanni: MUSTANG - State Ass. of FSMs Targeting Multilevel Logic Implementations. *IEEE Trans. on CAD*, 12(11):1290-1300, December 1988

[6] Forth, Molitor, Vogt: State Encoding of FSMs Target. BDD Repres. *Priv. Comm. Univ. Halle*, Germany, March 1998

[7] T. Lengauer: *Combinatorial Algorithms for Integrated Circuit Layout.* Teubner Verlag, 1990

[8] Livingston, Stout. Embeddings in Hypercubes. *Mathematical and Computational Modelling.* 11:222-227, 1988

[9] Najm, Goel, Hajj: Power Estimation in Sequential Circuits. *Proc. of the 32th DAC*, 635-640, 1995.

[10] Sentovich, Singh, Brayton, Sangiovanni: SIS - A System for Sequential Circuit Synth. Tech. Rep., UC Berkeley, 1992.

[11] Tsui, Pedram, Despain: Exact and Approximate Methods for Calculating Signal and Transition Probabilities in FSMs. *Proc. of the 31th DAC*, 18-23, 1994

[12] Tsui, Pedram, Despain: Low Power State Assignment Targeting Two- and Multilevel Logic Implementations. *Proc. of the 31th DAC*, 82-87, 1994

[13] Wagner, Corneil: On the Complexity of the Embedding Problem for Hypercube Related Graphs. *Discrete Applied Mathematics*, 43:75-95, 1993