Low Power FSM Design using Huffman-Style Encoding *

Prasoon Surti
Intel Corporation
Folsom, California
USA

L. F. Chao, Akhilesh Tyagi
Iowa State University
Ames, Iowa
USA

Abstract

This paper presents a novel approach to synthesize low power FSMs using non-uniform code length. Switching activity is reduced by decreasing the expected number of state bits switched less than $\lceil \log |S| \rceil$. The state set S of the FSM is decomposed into two sets based on the limit state probabilities. The state set with very high probability is encoded with less than $\lceil \log |S| \rceil$ bits. The other state set, being less probable, is encoded using more than $\lceil \log |S| \rceil$ bits. To the best of our knowledge, this is the first time two code lengths are used for one state machine. This encoding is realized by using flip-flops with gated clock. The logic generating the enable signal of the clock uses only a single minterm. The state sets can be encoded using any uniform-length encoding algorithm with objectives of low power and low area. The experiments show an average of 13% and 18% reduction in power for two encoding algorithms respectively.

1 Introduction

Low Power finite state machine (FSM) synthesis is conventionally identified with low power state assignment [1, 12]. State assignment alone may not provide all possible tradeoffs between area and power. An alternative low power combinational and sequential circuit design method modifies the underlying implementation architecture.

In [3], the clock is gated to avoid unnecessary switching at state-bit flip-flops (FFs) for self loops in the FSM. This incurs overhead in terms of extra circuitry and even the latency. And it is not always possible to have self loops with enough high probability. In [4], the aim is to 'shut off' the modules which are not used for computation in the current clock cycle. This approach also bears an area overhead. The technique proposed in [2] leads to a more general low power design methodology targeting selective disabling of a subset of inputs. In [2], exponential-time algorithms are proposed for both combinational and sequential logic circuits for low power under area constraints. The techniques that selectively shut down the inactive parts of an FSM implementation have not been investigated completely. This paper provides one mechanism in this direction. The proposed work is based on probabilistic analysis of FSMs and exploiting the skew in the limit state probability distribution which appears in most practical controllers. The core argument here is to use less expected number of state bits than $\lceil \log |S| \rceil$, where S is the state set. This is possible because of the skew in the state probability distribution, as entropy H is less than $\lceil \log |S| \rceil$. From information theory [5], H is the minimum number of bits required to encode states in an FSM. The expected code length H can be achieved by Huffman coding, where higher probability states have shorter code length. An FSM using Huffman encoding can be realized by disabling a subset of FFs whenever the present state has a shorter code length. Although the total code length might be longer than $\lceil \log |S| \rceil$, the extra FFs are disabled for high probable states.

¶This work is supported by NSF Research Initiation Award MIP-9410080 and by a grant from Intel Corporation.

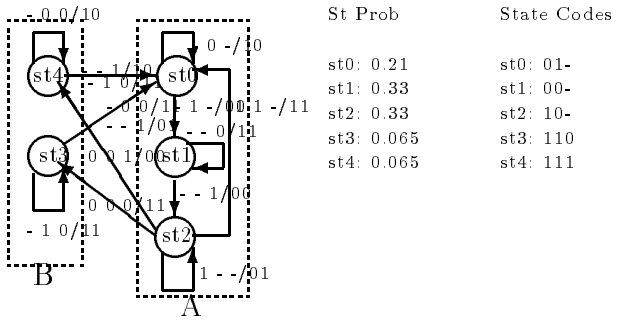


Figure 1: The example FSM having five states, state probabilities and codes

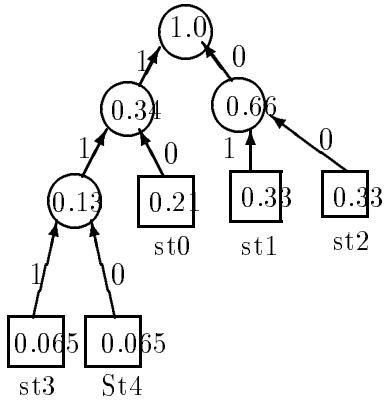


Figure 2: The Huffman coding algorithm

Consider an example FSM shown in Figure 1(a). The proposed approach will be explained using this five-state FSM throughout this paper. Figure 2 shows Huffman coding tree for this FSM, where leaf nodes are the states to be encoded. The Huffman coding algorithm assigns a state code derived from the labels of the path leading to that state. Hence the codes for the five states are as shown in Figure 1(c). The entropy in our example is 2.04 bits. The conventional minimal (uniform) length code would have used 3 bits. If general Huffman codes are used, this approach may not always be practical since variable code lengths have significant management overhead to generate flip-flop enable signals. A special case of this general approach is to use just two code lengths. In the proposed *Huffman-code architecture*, the overhead of generating the enable signal involves only one minterm, as

ED&TC '97 on CD-ROM

discussed in Section 3.

The state set of an FSM is decomposed into two sets such that one of them consists of *highly probable* states. This set is encoded with less than $\lceil \log |S| \rceil$ bits. The less probable set is encoded with more than $\lceil \log |S| \rceil$ bits. The two sets are determined such that the expected number of bits is less than $\lceil \log |S| \rceil$. Hence the rationale is that the savings in power from the highly probable state set dominate the extra cost for the other set. The above example clearly explains this concept. In Figure 2, the three states in the second level form the highly probable state set and the two third-level states form the less probable state set.

A representative set of existing low power state assignment algorithms attempts to minimize switching activity by reducing the Hamming distance between the states with high state transition probability between them. Various cost functions combined with these probabilities are suggested to control both area and power. The proposed method is not mutually exclusive with any of these state assignment techniques, but complements them.

Section 3 introduces criteria for this decomposition, the proposed encoding scheme and Huffman-code architecture. The partitioning and a hierarchical encoding algorithms are described in Sections 4 and 5. Finally, experimental results are presented.

2 Preliminaries

The terms used in this paper are defined here:

Finite State Machine: A FSM (or *sequential machine*) is a quintuple $M =< S, I, O, \delta, \lambda >$, where S is a set of states, I is a set of primary inputs (PIs), O is a set of primary outputs (POs), $\delta : S \times I \rightarrow S$ is the transition function, λ is an output function. For **Moore machines**, $\lambda : S \rightarrow O$ and for **Mealy Machines**, $\lambda : S \times I \rightarrow O$. For a transition from state i to state j, the **conditional state-transition probability** $p_c(i,j)$ is the probability that a transition from i to j occurs given that the present state is i. The **limit state probability** $p_s(i)$ of the state i is that of the FSM being in the state i as time approaches infinity. The absolute probability that a state transition from state i to state j occurs, i.e. **absolute state-transition probability** is defined as

$$p_a(i,j) = p_s(i) \times p_c(i,j).$$

In practice, many primary inputs of an FSM correspond to conditional signals which represent the outcome of the system under control. Behavior information can be used to derive accurate probabilities for primary inputs. Primary input probability distribution may have significant skew. For example, a conditional signal for zero detection or error checking would be negated with very high probability. The assumption that all primary inputs are equally probable, is impractical. To accurately capture the state probabilities of an FSM, this skew in primary inputs should be taken into account. Let $P(I_i)$ be the probability of primary input literal I_i being asserted. $PI\_weight(i,j)$, the weight for the edge (i,j) in the FSM, is computed as

$$PI\_weight(i,j) = \sum_{\forall m_i \in CC(i,j)} \Pi_{i=1}^{i=|I|} P(I_i)$$

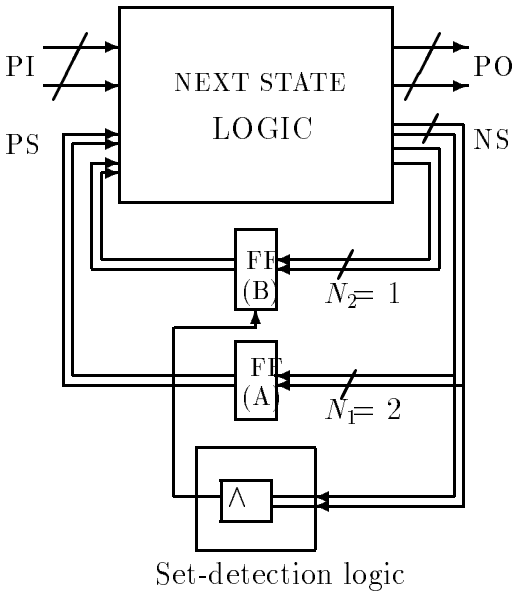


Figure 3: The Huffman-code architecture for FSMs

where $CC(i,j)$ is the set of primary input minterms enabling transition (i,j). The conditional state-transition probability can be computed as

$$p_c(i,j) = \frac{PI\_weight(i,j)}{2^{|I|}},$$

By modeling FSMs as Markov chains, limit state probabilities are computed from a system of linear equations, using conditional state transition probabilities.

3 The Overall Scheme

The proposed approach first partitions the FSM into highly probable state set A and less probable state set B, and then encodes states in A using N_1 bits and states in B using $N_1 + N_2$ bits, where the values of N_1 and N_2 are determined by the proposed encoding scheme. The partition is obtained such that the *expected code length*,

$$E(A) = p_A \cdot N_1 + p_B \cdot (N_1 + N_2) < \lceil \log |S| \rceil ,$$

where p_A and p_B are the total probabilities of states in A and B respectively. Two different code lengths for states of an FSM are realized using the proposed Huffman-code architecture, shown in Figure 3. The *set-detection logic* disables N_2 FFs when the next state is in set A. Therefore, N_2 FFs are disabled with high probability.

Entropy is maximum when all the states in the FSM are equally probable, in which case the minimum expected number of bits required is $\lceil \log |S| \rceil$. The skew in the state probability distribution gives rise to lower entropy and therefore, less expected number of bits for encoding. The proposed architecture is expected to reduce switching activities for FSMs with lower entropy in the state set.

In order to keep the set-detection logic simple, a fixed prefix of N_1 bits is used for the states in B. The *hierarchical encoding* scheme, presented in Section 5, first determines N_1-bit codes for states in A and the prefix of set B, and then assigns

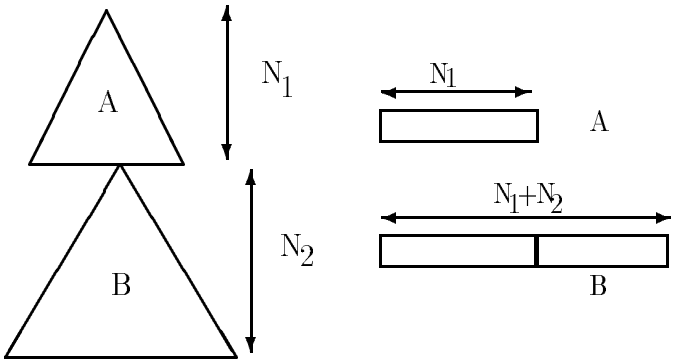


Figure 4: Single sub-tree coding

N_2-bit codes for states in B. Hence, $N_1 = \lceil \log |A| + 1 \rceil$ and $N_2 = \lceil \log |B| \rceil$. The coding tree is shown in Figure 4.

For example in Figure 1, $N_1 = 2$ and $N_2 = 1$ and the encoding is shown in Figure 1(c), where the prefix of set B is 11. Therefore the set-detection logic is a 2-input AND gate as shown in Figure 3.

4 FSM Partitioning

The primary objective of partitioning the FSM is to extract a subset A of the state set S of the FSM with highly probable states. Therefore the objective is to find a subgraph of the FSM where the FSM spends a significant amount of time. The total probability of the set A should be at least β and its size should be no more than $\gamma \cdot |S|$, where β and γ are user given. The parameter β enforces set A to have high enough probability so that N_2 FFs are disabled *most of the time*. Therefore, this ensures less switching on the corresponding high capacitance feedback state lines. Since the set-detection logic corresponds to a minterm of size N_1, the size constraint γ on set A attempts to keep area overhead low. An initial partition is first obtained by a greedy approach and then improved by a directed min-cut bi-partitioning heuristic derived from Kerninghan and Lin's (K & L) algorithm [13].

4.1 Partitioning Problem Formulation:

Given a STG $M =< S, I, O, \delta, \lambda >$ and parameters β and γ; $0 < \beta, \gamma < 1$, find a partition $\pi = (A, B)$ such that $A \cup B = S$ and $A \cap B = \phi$, and following conditions are satisfied:

1. $\sum_{\forall i \in A} p_s(i) \geq \beta$ (Probability constraint).
2. $|A| \leq \gamma \cdot |S|$ (Set size constraint).
3. $p_A \times \lceil \log(|A|+1) \rceil + (1 - p_A) \times (\lceil \log(|S| - |A|) \rceil + \lceil \log(|A|+1) \rceil) < \lceil \log(|S|) \rceil$, where $p_A = \sum_{\forall i \in A} p_s(i)$, $N_1 = \lceil \log(|A|+1) \rceil$ and $N_2 = \lceil \log |B| \rceil$ (Expected code length constraint).
4. Minimize $W(A,B) - W(B,A)$, where $W(A,B) = \sum_{\forall (i,j), i \in A, j \in B} p_a(i,j)$ and $W(B,A) = \sum_{\forall (i,j), i \in B, j \in A} p_a(i,j)$ (Interset transition constraint).

The first and third conditions are necessary. The first condition guarantees that set A has a high probability to disable N_2 FFs. Hence, larger β implies that the switching probabilities on the disabled state bits will be less. The third condition confirms that the available skew is exploited and the expected code length is less than $\lceil \log |S| \rceil$. More states in A tend to increase the size of set-detection logic. The second condition puts an upper bound on the size of set A in favor of simple set-detection logic.

The fourth objective attempts to reduce switching on N_2 bits due to inter-set transitions. The N_2 bits do not contribute to switching for state transitions from set B to set A because N_2 FFs are disabled for these transitions. However, switching might occur at N_2 FFs for transitions from set A to set B because N_2 bits are switched from the previous exit state in B to a different entry state in B. Furthermore, the fourth objective also minimizes the probability of leaving set A and maximizes the probability of returning to set A, so that the FSM tend to stay in set A as much as possible.

4.2 Initial partition using greedy approach

An initial partition is obtained to satisfy the probability and size constraints and minimize the expected code length. In order to allow freedom in moving states in the modified K & L algorithm, a slack σ is introduced in the size constraint. The greedy approach sorts states in the decreasing order of their limit state probabilities, and add states to A one by one until the adjusted size constraint, $|A| \leq (\gamma - \sigma) \cdot |S|$ is violated. If no partition satisfies the first, third and the adjusted size constraints, states are added to A until $|A| \leq \gamma \cdot |S|$ is violated. If no feasible partitions are found, it implies that the FSM is not suitable for the Huffman-code architecture to save power. Among feasible partitions, the one with the minimum expected code length is chosen as the initial partition. In our implementation, the slack variable σ is set to 0.1.

4.3 Improving partition using K & L approach:

The initial partition is improved to minimize $W(A,B) - W(B,A)$ using a bi-partitioning heuristic from the K & L algorithm, which is originally designed for undirected graphs and equal-size bi-partition. The objective of min cut here differs because the cut in our problem is directed. Also, basic K & L allows only exchange of nodes between the sets of initial partition. We enforce the size-constraint and also allow moving a state from a set to the other. Typically K & L allows a node to move across the partition only once in the inner loop. Here, a node can be moved as many times as a given *lock-level* [9]. In this work, *lock-level* = 3. This allows more flexibility of considering a state for moving or exchanging more than once.

5 State Assignment

Huffman-code architecture takes maximal advantage of skew in the state probability distribution for reducing switching on N_2 bits. Switching on N_1 bits is minimized by encoding all states in A and the prefix for set B using an algorithm targeted to low area and power. Encoding states in B using

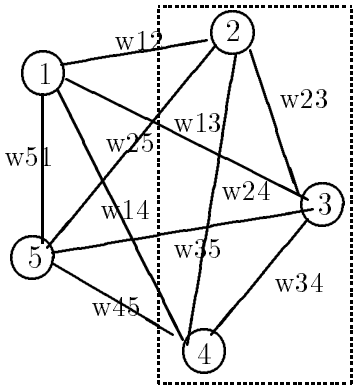
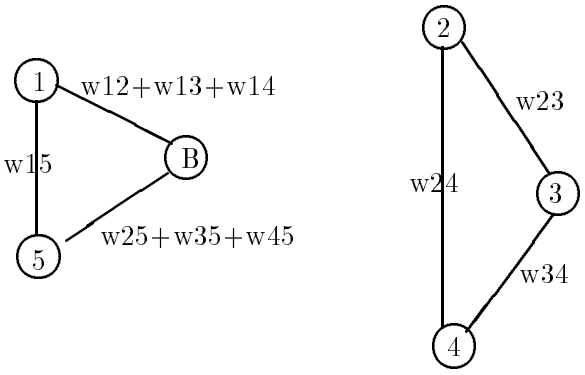


Figure 5: An example complete graph for a 5 state FSM



(a) Upper level graph (b) Lower level graph

Figure 6: The hierarchical graphs with edge-weights

Benchmark	*FO-Switching*		*FIFO-switching*	
	FO	HUFF	FIFO	HUFF
bbara	0.811247	0.449362	0.520057	0.438332
bbsse	0.802670	0.797596	1.192758	0.797596
bbtas	0.649703	0.735634	0.649703	0.510929
beecount	0.603442	0.553743	0.605448	0.605446
cse	0.888903	0.888885	0.444471	0.555567
dk17	1.199776	0.903591	1.200618	1.019485
dk27	1.517879	1.018726	1.020507	0.556136
ex1	0.849418	0.803677	0.838020	0.627168
keyb	1.222544	0.812433	0.946497	1.006067
lion	0.927350	0.634003	0.927350	0.656509
sse	1.094890	1.059110	1.576459	1.119090
Average	0.960711	0.786978	0.901990	0.717484

Table 1: Improvement in avg switching

the same algorithm should further reduce switching on N_2 bits for state transitions within set B. Thus, using a minimal and uniform length encoding algorithm, N_1- and N_2-bit state codes are derived hierarchically. It is interesting to note that this hierarchical state encoding approach involves smaller state sets than the original state set. Therefore, the computation time for state encoding might be reduced.

Many state assignment algorithms [8, 11] for multi-level logic implementation first create an edge-weighted complete graph induced by the state set and then solve the graph embedding problem. Further discussion assumes that hierarchical encoding uses a state assignment algorithm of similar nature. Techniques to adapt such algorithms to our approach are presented below.

An example of such edge-weighted complete graph is given in Figure 5. The edge weight $w(i,j)$ is a measure of possible power and area savings by encoding states corresponding to nodes i and j with less Hamming distance. The hierarchical encoding approach first builds an edge-weighted graph for set S. Complete graphs for the two sets to be encoded are then derived from it. The complete graph induced by the state set A plus one node for prefix of B, is called the *upper level graph*. The upper level graph in our example is shown in Figure 6(a). The circled portion of the graph in Figure 5 (i.e. the partition B) is represented as a single node B and the weights on the incident edges have the weights obtained by summing up all the weights of the corresponding edges, i.e., the weight on edge (i, B) is

$$w_u(i,B) = \sum_{\forall j \in B} w(i,j),\ \forall i \in A$$

For the second application of the encoding algorithm, the subgraph of the original complete graph, induced by the state set B is used. The edge-weights are the same as those in the original graph. This is due to the fact that these N_2 bits for set B codes do not contribute to the switching activity during inter-set transitions.

6 Experimental results

We use the two state assignment algorithms described in [11]:

- **FO-CLIQUE:** This algorithm first computes edge-weights for the complete graph induced by the state set. These weights are similar to MUSTANG's *fan-out oriented* weights [8]. In FO-CLIQUE, these weights are modulated by absolute state transition probabilities in order to minimize $\sum_{\forall (i,j)} p_a(i,j) \times HD(i,j)$, where $HD(i,j)$ is the hamming distance between the codes of the states i and j.
- **FIFO-CLIQUE:** Here, the only difference is that the weights are computed combining both *fan-in and fan-out oriented* Mustang weights modulated by absolute state transition probabilities.

In both the algorithms, the state assignments are determined using clique-based recursive bi-partitioning heuristic. The proposed approach is applied to the benchmark FSMs from MCNC benchmark suite. The probabilistic analysis of the FSMs is performed and limit state probabilities are computed. To validate the proposed approach, skew in the probability distribution is introduced by assigning random probabilities for primary inputs. In the partitioning phase the user defined parameters are kept at $\beta = 0.8$ and $\gamma = 0.5$. In order to implement set-detection logic efficiently, *kiss* files are modified to include flip-flop enable signal as a primary output. Therefore, the set-detection logic is synthesized as a

Benchmark	*FO-CLIQUE*		*HUFFMAN*			
	Area	Power	Area	Power	%area inc	%power red
bbara	71,456	237.3	87,232	166.9	18	29
bbsse	1,17,392	634.5	1,15,072	404.1	-2	36.3
bbtas	34,336	146.2	42,688	145.1	19.5	0.01
beecount	43,616	231.9	55,216	214.0	21	7.71
cse	1,95,808	597.5	2,22,720	548.7	8.79	8.16
dk17	60,320	398.2	78,416	391.9	22.95	1.75
dk27	37,120	249.4	49,184	242.9	24.34	2.8
ex1	2,57,520	1084.4	2,76,080	888.8	6.72	18.03
ex6	87,696	516.3	1,20,640	523.1	37.56	-1.29
keyb	1,80,960	735.1	1,86,064	663.2	2.74	9.78
lion	19,952	88.3	20,416	78.2	2.72	11.32
sse	1,18,320	620.6	1,52,656	475.2	22.49	23.42
Average					13.34	13.54

Table 2: Results for MCNC benchmarks compared with FO-CLIQUE

Benchmark	*FIFO-CLIQUE*		*HUFFMAN*			
(N_1, N_2, log(|S|))	Area	Power	Area	Power	%area inc	%power red
bbara (2,3,4)	71,920	249.0	98,832	226.2	27.2	9.15
bbsse (1,4,4)	1,15,072	642.0	1,22,960	451.7	6.41	29.64
bbtas (3,1,3)	36,192	125.9	44,080	106.1	17.89	15.73
beecount (2,3,3)	43,616	234.4	62,176	204.0	29.85	12.97
cse (1,4,,4)	1,93,952	795.9	2,00,912	461.4	3.46	42.03
dk17 (2,3,3)	63,104	425.1	83,984	389.2	14.92	8.45
dk27 (1,3,3)	30,624	231.5	46,400	240.3	51.5	-3.53
ex1 (1,5,5)	2,54,736	1118.3	3,25,264	1100.5	24.86	1.6
ex6 (2,3,3)	83,056	412.3	1,01,152	383.1	21.68	7.08
keyb (1,5,5)	2,10,192	1027.9	2,32,928	715.2	17.89	30.42
lion (1,2,2)	21,808	74.9	21,808	64.9	0	13.33
sse (2,4,4)	1,21,568	669.6	1,42,912	471.1	14.93	29.64
Average					14.92	18.19

Table 3: Results for MCNC benchmarks compared with FIFO-CLIQUE

part of the next state/output logic to further reduce the area overhead.

The proposed technique improves upon both FO-CLIQUE and FIFO-CLIQUE state-assignment algorithms [11] in terms of average switching activity. The average switching values for MCNC kiss benchmarks are presented in Table 1. HUFF refers to the proposed approach.

The next state/output logic is synthesized under Berkeley's **sis-1.2** using *script.rugged* for logic minimization and *lib2.genlib* as a library. The power and area are estimated for the final implementation. The results are compared with the original FO-CLIQUE and FIFO-CLIQUE state assignment algorithms in Tables 2 and 3.

These results clearly show the potential of the proposed approach. Compared to FO-CLIQUE, HUFFMAN reduces power by an average of 13.54% at an average cost of 13.34% increase in area. Also, compared to FIFO-CLIQUE, HUFFMAN reduces power by an average of 18.19 % and the average area overhead is 14.92%. Both FO-CLIQUE and FIFO-CLIQUE outperform JEDI and MUSTANG. FIFO-CLIQUE does better than FO-CLIQUE in terms of both area and power. In our experiments, larger power savings and less area overhead are delivered by FIFO-CLIQUE. Hence, we believe, for any low power state assignment algorithm, the proposed approach can potentially improve upon it.

The area overhead in this approach arises mainly due to increased number of FFs but this does not lead to increase in power because a subset of FFs are disabled with high probability. First column of Table 3 gives the numbers N_1, N_2 and minimal number of FFs, i.e., $\lceil \log |S| \rceil$ for our experiments. For small FSMs, the area overhead may offset gain in switching and therefore result in little power savings, as for benchmarks *ex6* in Table 2 and *dk27* in Table 3. Low overhead in area implies significant savings in power as seen from benchmark *cse*.

7 Conclusions

This work, to the best of our knowledge, is the first attempt to use non-uniform code length in order to synthesize low power sequential circuits. We proposed the Huffman-code architecture to realize encoding using two different code lengths. The results in this work using a single prefix are encouraging and therefore, it is worth to explore the use of more than one prefix for B states to simplify set-detection logic further and possibly provide more scope for reducing interset switching. Results presented clearly show that the Huffman-code architecture can provide a promising tool to explore the trade-off between area and power. A low power FSM synthesis framework can integrate the proposed technique with a variety of state assignment algorithms. Future work involves extending the concept of decomposition of FSM to the implementation level by using a notion similar to the interacting FSMs.

References

[1] C. Y. Tsui, M. Pedram, C. Chen and A. M. Despain. Low Power State Assignment Targeting Two- and Multi-level Logic Implementations. *Proceedings of the ACM/IEEE International Conference on Computer-Aided-Design*, pp.82-87, 1994.

[2] M. Alidina, J. Monterio, S. Devadas and A. Ghosh, Precomputation-Based Sequential Logic Low Power. *IEEE Transactions on VLSI Systems*, pp.426-435, December 1994.

[3] L. Benini and G. D. Micheli, Transformation and Synthesis of Low-power Gated-clock Implementation. *International Symposium on Low Power Design*, pp.21-26, 1995.

[4] V. Tiwari, S. Malik and P. Ashar, Guarded Evaluation : Pushing Power Management to Logic Synthesis/Design. *International Symposium on Low Power Design*, pp.221-226, 1995.

[5] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, NJ, 1991.

[6] G. D. Hatchel, E. Macii, A. Pardo and F. Somenzi, Probabilistic Analysis of Large Finite State Machines. *31st Design Automation Conference*, pp.270-275, 1994.

[7] S. Devadas, and A. R. Newton, Decomposition and Factorization of Sequential Finite State Machines. *IEEE Trans. Computer-Aided Design*, pp. 1206-1217, Nov. 1989.

[8] S. Devadas, H-K. T. Ma, A. R. Newton and A. Sangiovanni-Vincentelli, MUSTANG: State Assignment of Finite State Machines Targeting Multi-level Logic Implementations. *IEEE Trans. on Computer Aided Design*, vol.7, pp. 1290-1300, Dec. 1988.

[9] V. M. Veeramachaneni, Low Power State Assignment of FSMs, Master's Thesis, Iowa State University, 1995.

[10] S. Devadas, Optimizing Interacting Finite State Machines Using Sequential Don't Cares, *IEEE Trans. on Computer Aided Design*, vol. 10, pp. 1473-1484, Dec. 1991.

[11] A. Tyagi, Integrated Area-Power State Assignment, *Proceedings of the Synthesis and Simulation Meeting and International Interchange, SASIMI '96*, pp. 24-31, November, 1996.

[12] L. Benini and G. De Micheli, State Assignment for Low Power Dissipation, *IEEE Solid State Journal*, 1995.

[13] B. W. Kernighan and S. Lin, An Efficient Heuristic Procedure for Partitioning Graphs, *Bell System Journal*, v.49, Feb 1970, pp. 291-307.