FSM Re-Engineering for Low Power State Encoding

Lin Yuan and Gang Qu
Electrical and Computer Engineering Department
University of Maryland, College Park, MD, 20742
{yuanl, gangqu}@eng.umd.edu

Abstract

Finite State Machine (FSM) synthesis traditionally starts with state minimization and state encoding, which provide codes of minimal length to the FSM with minimal number of states. Recently, there have been studies on encoding with non-minimal length and synthesis on non-minimized FSM. In this paper, we propose the framework of FSM re-engineering, which starts with synthesizing the original FSM, followed by re-constructing a functionally equivalent but topologically different FSM, and ends with another round of FSM synthesis on the newly constructed FSM. This gives us a larger solution space that consists of synthesis solutions to any of the functionally equivalent FSMs rather than the original FSM only. Guided by the result of the first round FSM synthesis, the solution space exploration process is rapid and cost-efficient. This framework enables us to find better synthesis solutions, sometimes even better than the optimal ones in the original FSM.

We demonstrate this framework on low power state encoding, where we first use POW3 to assign code to each FSM benchmark, then we re-construct the FSM by introducing new states selectively, the re-constructed FSM is encoded again by POW3. Experiments on MCNC benchmarks show that we are able to reduce the FSM's switching activity by 6.0% on average. This results in an average 9.4% energy reduction at the cost of 1.3% area increase in SIS simulation, which is better than other non-minimal length low power encoding techniques on comparable cases. More interestingly, when we obtain the optimal coding via an integer linear programming formulation on small size benchmarks, we find that POW3-encoded original FSMs are 27.0% worse than the optimal, but this number drops to 10.1% when we apply POW3 to the re-engineered FSMs.

1 Introduction

Finite state machine (FSM) is the most commonly used model for system's sequential components. Logic synthesis, which has the goal of converting the symbolic description of the FSM to a hardware implementation, traditionally starts with FSM state minimization and state encoding in order to optimize design objectives such as area, delay, and testability. For example, De Micheli et al. [14] formulate the minimum area state encoding problem as generating a minimum (multi-valued) symbolic cover of the FSM and propose a heuristic row encoding technique in [15]. Villa et al. [23] use the notion of face-posets to tackle this problem and propose a state encoding technique for two-level implementation. State encoding techniques for multi-level logic minimization have been studied in [4] and [13] where the goal is to reduce the number of literals in the Boolean output and next-state functions.

With the increasing popularity of portable computing and personal communication applications in early 90's, power dissipation has become critical in the design of microelectronic circuits. Research on low power circuit design is widespread and ranges from high-level approaches such as power efficient instruction set design and dynamic voltage scaling to low-level techniques like clock gating. Low power state encoding techniques have also been proposed at that time reflecting this system design shift from high performance to low power. In light of the well-known fact that digital CMOS circuit's power dissipation is proportional to the switching activity, state encoding is then re-formulated to minimize the number of state bit switches per transition for low power FSM synthesis. This problem is NP-hard and many heuristic algorithms have been proposed mainly based on the idea of assigning codes with small Hamming distance to pairs of states that have a high transition probability. Such techniques include state encoding with minimal code length [2, 18, 21], non-minimal code length [12, 16] and variable code length [20]; state re-encoding approaches [5, 22] and techniques that try to minimize power and area simultaneously [9, 17].

However, these work all start with the minimized FSM and seek for the best encoding for the existing states to reduce switching activity. On the other hand, there is a much longer history on the study of conducting state minimization and assignment at one step (see, for example, [1, 7, 11]), but reducing switching activity or power has never been the goal for any of these approaches.

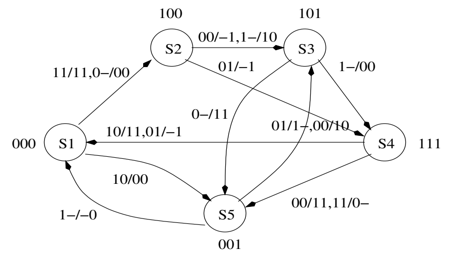
We present the concept of **FSM re-engineering** for logic synthesis. It is based on the observation that the best synthesis solution does not necessarily come from the minimized FSM, as we will see in the following motivational example, or coding with the minimal length, as has already been demonstrated earlier [12, 16, 20]. In this approach, we first apply any FSM synthesis technique to obtain a synthesis solution; we then identify the structure of the FSM that might prevent us from getting better solutions and re-construct the FSM accordingly; the re-engineered FSM will be synthesized again to generate new solution.

In the rest of this section, we first use an example to show that we might lose the optimal solution if we restrict the synthesis on minimized FSM. Then we explain why the proposed FSM re-engineering framework is different from existing approaches and why we believe that it can guide us, both theoretically and practically, to better solutions. We mention that, although we restrict our discussion to low power encoding in the rest of the paper for simplicity, the proposed framework is generic and can be applied for the optimization of other design objectives such as area and testability.

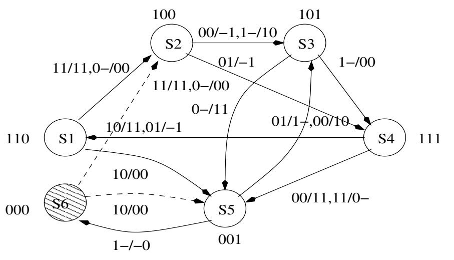
1.1 A Motivational Example

We take the example from a recent paper on power-driven FSM state encoding [8] to show the potential of the proposed FSM re-engineering approach. The state transition graph (STG) in Figure 1(a) represents a 2-input 2-output FSM with five states $\{S1, S2, S3, S4, S5\}$. Each edge represents a transition with the input and output pair shown along the edge. The FSM has already been minimized.

We re-construct this FSM by introducing state *S6* as shown in Figure 1(b). One can easily verify that these two STGs are functionally equivalent. In fact, state $S6$ is an equivalent state of $S1$. We then exhaustively check all the possible state encoding schemes for both FSMs and report the one that minimizeds total switching activity in Figure 1 as shown next to each state.



(a) Original STG with a total switching activity of 1.27



(b) The reconstructed STG with a total switching activity of 1.17

Figure 1. A 5-state FSM and a functionally equivalent 6-state FSM.

We now calculate the switching activity, an indicator of power efficiency of the encoding scheme. We observe a 7.9% reduction in the re-constructed 6-state FSM over the original 5-state FSM. Note that the encoding in the original 5-state FSM is optimal, which implies that we lose the most energy-efficient encoding for this FSM (and its functionally equivalent FSMs) once it is minimized!

FSM re-engineering not only gives the theoretical opportunity to build FSM with better energy efficiency, it can also be applied to existing low-power encoding algorithms. For example, when we use POW3 [2] instead of the exhaustive search to encode the original 5-state FSM, it gives a coding with switching activity 18.9% higher than the optimal. However, when we use POW3 to encode the equivalent 6-state FSM, it successfully finds a coding that is only 5.4% away from the optimal.

1.2 What Is New

FSM re-engineering refers to the procedure of re-constructing an FSM that is functionally equivalent to a given FSM such that one can obtain a better synthesis solution from the newly built FSM. In the context of low power state encoding, it takes an encoded FSM as input and outputs a functionally equivalent FSM with reduced switching activity. The novelty of this approach, which separates it from the state re-encoding techniques, is that we are exploring the equivalent FSMs rather than restricting ourselves to encoding (or re-encoding) of the same minimized FSM.

Traditionally, state encoding (and re-encoding) is performed after state minimization. Minimizing FSM first normally results in simplier function implementation, less hardware, and shorter delay. However, this is not necessary from the point view of power efficiency because power is proportional to the switching activity, not the number of states. Leaving redundancy such as equivalent states in the FSM can be helpful. For example, state $S1$ in Figure 1(a) originally has four edges and contributes a lot to the total switching activity because states $S1$ and $S4$ have the largest Hamming distance. Duplicating state $S1$ solves this problem. Furthermore, implementing non-minimized FSM does not always mean increased hardware. For example, a 36-state FSM and a 42-state FSM need the same number of latches (flip flops, or state registers).

Besides proposing the FSM re-engineering framework and applying it to low power state encoding, we also report the following findings:

- The FSM re-engineering problem is NP-hard.
- FSM re-engineering technique enhances the performance of low power state encoding techniques. For example, POW3's efficiency in reducing switching activity is almost doubled when combined with our FSM re-engineering technique.
- The potential of FSM re-engineering in low power state encoding is analyzed by comparing to the optimal encoding obtained from an integer linear programming formulation of the state encoding problem.

2 Related Work

In this section, we survey the most relevant work on FSM low power state encoding and show their difference from the proposed FSM re-engineering framework.

Dynamic power dissipation in CMOS circuits is composed of power consumed in sequential logic and combinational logic. Power dissipated in the combinational logic mainly depends on the complexity of the Boolean logic functions and their gate level implementation. Power dissipation in sequential logic is due to capacitance charging and discharging in state registers caused by the state bits switching, which is often described as

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

where V_{dd} is supply voltage, f is clock frequency, $C(i)$ is the capacitance of the register storing the ith state bit, and $E(i)$ is the expected switching activity of the ith register. $C(i)$ is technology dependent and remains, in general, constant for all the state bits.

There have been a number of power-driven state encoding algorithms to reduce the switching activity $E(i)$ and hereby power. Roy and Prasad propose a simulated annealing based algorithm to improve any given state encoding scheme [18]. Washabaugh et al. suggest to first obtain state transition probability, then build a weighted state transition graph, and finally apply branch and bound for state encoding [24]. Olson and Kang present a genetic algorithm, where in addition to the state transition probability, they also consider area while encoding in order to achieve different area-power trade-offs [17]. Benini and De Micheli present POW3, a greedy algorithm that assigns code bit by bit. At each step, the codes are selected to minimize the number of states with different partial codes [2]. Iman and Pedram developed a power synthesis methodology and created a complete and unified framework for design and analysis of low power digital circuits [10].

Unlike these power-driven state encoding algorithms, low power state re-encoding techniques start from an encoded FSM and seek for a better coding scheme to reduce switching activity. Hachtel et al. recursively use weighted matching and mincut bi-partitioning methods to re-assign codes [5]. Veeramachaneni et al. propose to perform code exchange locally to improve the coding scheme's power efficiency [22]. Our FSM re-engineering approach is conceptually different from re-encoding in that we look to change the topology of the FSM, not only re-assign codes to the existing states.

The above work takes two common assumptions, 1) they look for codes with the minimal length, that is, the number of bits to represent a state will be $\lceil \log n \rceil$ for any n-state FSM; 2) their encoding (or re-encoding) algorithms are applied after state minimization is done. There are a couple of recent work on non-minimal length encoding algorithms showing that power may be improved with code length longer than this bound [12, 16]. These methods require extra state register(s) in the FSM implementation which will add to the hardware cost and cause area increase. However, none of the papers have reported the area overhead. Our approach is essentially different from theirs in that we do not introduce extra state bits (when the number of states is not 2^k). Therefore, the area overhead in our approach expects to be much less. Besides, as we have mentioned earlier, our technique is a stand-alone FSM encoding enhancement. FSM re-engineering can also be applied to non-minimal length encoding algorithms to find better solutions.

Finally, we mention the one-bit hot encoding where each state in an n-state FSM receives an n-bit code with exactly one bit to be 1. This encoding scheme can greatly simplify the logic implementation of the FSM and could also reduce the switching activity because now every pair of states will have a Hamming distance equal to two. However, it requires a code of length the same as the number of states and this makes it impractical for FSMs of large size.

3 Preliminary

We consider the standard state transition graph (STG) representation of an encoded FSM $G = (V, E)$, where a node $v_i \in V$ represents a state s_i with code C_i in the FSM M, and a directed edge $(v_i, v_j) \in E$ represents a transition from state s_i to state s_j with transition probability P_{ij}. We simplify this directed weighted graph G to an undirected weighted graph $\tilde{G} = (V, \tilde{E}, \{C_i\}, \{p_{ij}\})$:

- V, the set of states, which is the same as in G;
- $\tilde{E}$, the set of edges. An edge $(v_i, v_j) \in \tilde{E}$ if and only if $(v_i, v_j) \in E$, or $(v_j, v_i) \in E$, or both;
- C_i, the weight of node $v_i \in V$, which is the code of state s_i;
- p_{ij}, the weight of edge $(v_i, v_j) \in \tilde{E}$, $p_{ij} = P_{ij} + P_{ji}$.

Denote $H(v_i, v_j)$ as the Hamming distance between the codes, two bitstreams C_i and C_j, of states s_i and s_j under the given encoding scheme. The total switching activity of the encoded FSM can be calculated as

$$\sum_{(v_i, v_j) \in \tilde{E}} p_{ij} H(v_i, v_j) \tag{2}$$

Recall that two FSMs, M and M', are equivalent if and only if they always produce the same sequence of outputs on the same sequence of inputs, regardless of the topological structure of their STGs. We formally formulate the FSM re-engineering problem as:

Given an encoded FSM M and its corresponding graph $\tilde{G} = (V, \tilde{E}, \{C_i\}, \{p_{ij}\})$, construct an equivalent FSM M' and encode it such that in the corresponding graph $\tilde{G}' = (V', \tilde{E}', \{C'_i\}, \{p'_{ij}\})$, we maximize the total switching activity reduction:

$$\sum_{(v_i, v_j) \in \tilde{E}} p_{ij} H(v_i, v_j) - \sum_{(u_i, u_j) \in \tilde{E}'} p'_{ij} H(u_i, u_j) \tag{3}$$

The FSM re-engineering problem targets the re-construction and encoding of a functionally equivalent FSM for low power FSM implementation. Clearly, it is NP-hard because it requires the best state encoding for the re-constructed FSM M', which is an NP-complete problem. Furthermore, when we restrict M' to be the same as M, the problem becomes "determining a new encoding scheme to minimize the total switching activity", which is the existing FSM re-encoding problem.

The novel contribution of the FSM re-engineering problem is that it re-constructs the original (minimized and encoded) FSM to allow us explore a larger design space for power-efficient FSM encoding. In this paper, we focus on the FSM re-construction and defer the state encoding problem to existing algorithms. We give an example on how to re-engineer an FSM and explain why it can reduce the switching activity.

Figure 2 illustrates one way to change the topology of the STG without altering the FSM's functionality. We see that a new state, S', is added as a duplicate of state S as follows: S' goes to the same next state under the same transition condition as state S; the transitions from other states to state S

in the original STG will be split such that some of them still go to state S while the rest go to the new state S'. Suppose that state Sp_j and S have a large Hamming distance in the original encoding and contribute a lot to the total switching activity, now we can redirect this transition to S' and assign S' a code of a small Hamming distance from Sp_j.

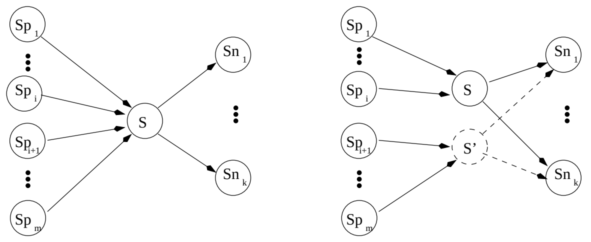
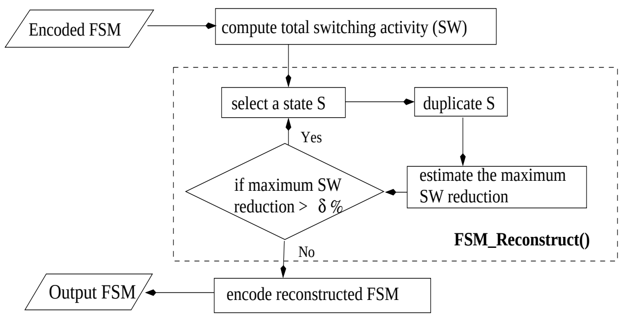


Figure 2. Re-constructing an FSM by duplicating a state S.

4 Power-Driven FSM Re-Engineering Approach

Figure 3 outlines the proposed low power state encoding approach by FSM re-engineering. We first compute the original FSM's total switching activity for a reference. Then we re-construct a functionally equivalent FSM and encode it for reduced switching activity. We will use the state duplication technique as an example to illustrate this approach. Figure 3 outlines the three key steps in the state duplication method: 1) select the best candidate state for duplication; 2) decide how to duplicate the selected state; 3) estimate the (maximum) switching activity reduction after the state duplication.



Encoded FSM
compute total switching activity (SW)
select a state S
duplicate S
Yes
if maximum SW reduction > δ %
estimate the maximum SW reduction
FSM_Reconstruct()
No
Output FSM
encode reconstructed FSM

Figure 3. FSM re-engineering for low power state encoding.

We now elaborate on how to compute an encoded FSM's switching activity and the pseudocode of state duplication based FSM re-engineering technique (Figure 4).

4.0 Compute FSM's Switching Activity

According to Equation (2), the state transition probability of each edge and the Hamming distance between the two states of each edge must be determined before the calculation of total switching activity. The former measures how frequently each transition occurs and the latter gives the amount that each transition contributes to the total switching activity.

To compute the transition probability, it is necessary to have the input distribution at each state, which can be obtained by simulating the FSM at a higher level of abstraction [24]. This gives us $p_{j|i}$, the conditional probability that the next state is s_j if the current state is s_i. Then we build a Markov chain based on these conditional probabilities to model the FSM. The Markov chain is a stochastic process whose dynamic behavior depends only on the present state and not on how the present state is reached [6]. We now can obtain the steady probability P_i of each state s_i corresponding to the stationary distribution of the Markov chain. The state transition probability P_{ij} for the transition $s_i \to s_j$ is given by

$$P_{ij} = p_{j|i} P_i \tag{4}$$

The Hamming distance between the two states of each transition can be conveniently determined after state encoding is performed. As we have mentioned earlier, it is not our goal to develop any power efficient encoding scheme. The proposed FSM re-engineering method seeks for a functionally equivalent FSM in order to provide opportunities for any encoding scheme to reduce the switching activity. This strength is shown in Figure 3 as we use the same algorithm to encode the original FSM and the re-constructed FSM. In our simulation, POW3 developed by Benini and De Micheli [2] is used as the state encoding scheme.

Algorithm: FSM Reconstruction

```
/* Step I: selection of state for duplication. */
 1. for each state $s_i$ in the FSM
 2.    compute $r(s_i)$ as defined in Equation (5);
 3.    sort the states by their $r$-values in descending order;
 4.    if tie, the one with less number of previous states first;
 5.    if still tie, the one with less number of next states first;
 6.    if still tie, break it randomly;
 7. do
 8.    select state $s$, the first one in the list, for duplication;
/* Step II: state duplication. */
 9.    for each pair $s_i$ and $s_j$ in $PS$, the previous states of $s$
10.       compute the Hamming distance $H(s_i, s_j)$;
11.    pick $s_1$ and $s_2$ s.t. $H(s_1, s_2) = \max_{s_i, s_j \in PS} \{H(s_i, s_j)\}$;
12.    $PT_1 = \{s_1\}; PT_2 = \{s_2\}$;
13.    $c_1 = s_1; c_2 = s_2$;
14.    for each state $t \in PS$
15.       if ($H(t, c_1) < H(t, c_2)$)
16.          $PT_1 = PT_1 \cup \{t\}$;
17.       else $PT_2 = PT_2 \cup \{t\}$;
18.    re-compute $c_1$ and $c_2$, the centers of $PT_1$ and $PT_2$;
19.    $H_{total} = \sum_{t \in PT_1} H(t, c_1) + \sum_{t \in PT_2} H(t, c_2)$;
20.    if ($H_{total}$ decreases)     goto line 14;
21.    for each state $t \in PT_1$
22.       add $t$ as a previous state of state $s$;
23.    for each state $t \in PT_2$
24.       add $t$ as a previous state of state $s'$;
25.    for each state $t \in NS$, the next state of $s$
26.       add $t$ as a next state of state $s'$;
/* Step III: estimate the gain in switching activity reduction. */
27.    determine the ideal codes for states $s$ and $s'$;
28.    compute the total switching activity locally at $s$ and $s'$;
29.    for each state $t \in PT_1 \cup PT_2 \cup \{s, s'\}$
30.       compute $r(t)$ and insert state $t$ back to the list;
31. while(gain in the total (local) switching activity exceeds $\delta$%)
```

Figure 4. Pseudocode: FSM reconstruction by state duplication.

4.1 Selection of States for Duplication

By duplicating a state, we make it possible to assign the same state more than one codes, one for the original state and the rest for the duplicate(s). As we have seen from Figure 2, states with large (average) Hamming distance from its previous states will benefit because they will have less previous states in the re-constructed FSM, which allows the encoding scheme to find a better code to reduce the Hamming distance. Outgoing edges to the next states and the codes of the next states do not have the same importance because each duplicate state will be connected to the same set of next states to preserve the correct functionality.

For each state s_i, we define:

$$r(s_i) = \sum_{(v_j,v_i)\in E} H(v_i, v_j)/\text{indgree}(v_i) \qquad (5)$$

where node v_i represents state s_i in the STG and the sum is taken over the number of all the incoming edges (v_j, v_i) at node v_i.

This value measures the average Hamming distance between state s_i and all its previous states. We duplicate one state at a time and each time we select the state with the largest r-value. If there is a tie, we select the state with fewer previous and/or next states to reduce the size of the re-constructed FSM. This could eventually help the encoding algorithm to find a better encoding scheme. If there is still a tie, we break the tie by selecting one state randomly.

4.2 Heuristics for State Duplication

Step II of the algorithm will actually duplicate the selected state. Ideally, we want to duplicate the state in such a way that the new FSM will maximally reduce the switching activity when encoded optimally. Apparently, this requires solving the NP-hard state encoding problem optimally. Instead, we focus on how to duplicate a state to minimize switching activity locally.

More specifically, let s be the state we select for duplication, PS and NS be the sets of previous states and next states of s respectively in the original FSM. The state duplication procedure 1) creates a state s' that also has NS as its next states, and 2) splits PS into PT_1 and PT_2 and make them as the previous states for s and s' in the new FSM. The goal of such local state duplication is to minimize

$$\sum_{t\in PT_1} P_{ts}H(t,s) + \sum_{t\in NS} P_{st}H(t,s) + \sum_{t\in PT_2} P_{ts'}H(t,s') + \sum_{t\in NS} P_{s't}H(t,s')$$

where P_{ts} is the transition probability from state t to state s and $H(t, s)$ is the Hamming distance between the two states.

The challenge is how to partition the previous states PS into two subsets. Our solution, as shown at Step II in Figure 4, is based on the fact that the two states in PS with the largest Hamming distance should belong to different partitions. We find, in line 11, states v_k and v_l that have the largest Hamming distance and put them into PT_1 and PT_2 as their respective centers (lines 12-13). For each of the other states $t \in PS$, we include it to the subset whose center is closer to t (lines 14-17). After we finish the partition, we re-compute the centers c_1 and c_2 of the two subsets (line 18) following the method described in Lemma 1 below. We then re-partition set PS based on these new centers and continue if the new partition results in reduced total Hamming distance (line 20).

The following lemmas show the correctness of this approach.

Lemma 1. In any optimal partition, state s and its duplicate s will have the codes of the two centers.

[Proof]. Suppose that one partition has k states with codes $\{x_{i1}x_{i2}\cdots x_{in} : i = 1, 2, \cdots, k\}$ and they will have state s as their next state in the re-constructed FSM. We want to find the code $c_1c_2\cdots c_n$ for state s to minimize the total Hamming distance

$$\sum_{i=1}^{k} H(s, x_i) = \sum_{i=1}^{k}\sum_{j=1}^{n} |x_{ij} - c_j| = \sum_{j=1}^{n}(\sum_{i=1}^{k} |x_{ij} - c_j|)$$

Because each bit is independent, the above is minimized if and only if $\sum_{i=1}^{k} |x_{ij} - c_j|$ is minimized for each $j = 1, 2, \cdots, n$. Let a be the number of 1's in $\{x_{ij} : i = 1, 2, \cdots, k\}$ and b be the number of 0's. $\sum_{i=1}^{k} |x_{ij} - c_j| = b$ if $c_j = 1$ and $\sum_{i=1}^{k} |x_{ij} - c_j| = a$ if $c_j = 0$. Clearly, it is minimized when c_j is defined as the majority of $\{x_{ij} : i = 1, 2, \cdots, k\}$.

Lemma 2. The optimal partition is reached in time linear to the size of set PS, i.e., the number of previous states of state s.

[Proof]. Because of its discrete nature, every time the loop (lines 14-20) is repeated, the total Hamming distance is reduced by at least 1. Therefore, this loop will stop after being repeated finite times. Furthermore, the largest Hamming distance from s (or its duplicate s') to any state in PS is n. If there are k states in PS, then the loop will not be executed more than kn times.

4.3 Estimate the Switching Activity Reduction

The goal of duplicating state is to reduce the total switching activity. After we construct the duplicated state s' of the selected state s, it becomes possible to estimate/evaluate whether this goal is achieved. Based on this, we make the decision whether more states will be selected for duplication or not.

One way is to encode the re-constructed FSM and compute the total switching activity using Equation (2). This gives us the actual gain in switching activity reduction by duplicating state s. When it is expensive to apply the state encoding algorithm on the entire FSM, we use the following alternative to estimate the maximum gain locally at states s and s' by assigning them the best codes.

Lemma 3. Let $\{x_i : (x_{i1}x_{i2}\cdots x_{in})\}$ be the set of states that have transition to/from state s and p_{x_is} is the transition probability between state x_i and s, then the switching activity is minimized at state s when it has code $c_1c_2\cdots c_n$, where

$$c_j = \begin{cases} 1 & \text{if } \sum_{x_i} p_{x_is}(1 - 2x_{ij}) < 0 \\ 0 & \text{otherwise} \end{cases}$$

[Proof]. Similar to the proof of Lemma 1 but note that now the transition probability p_{x_is} is available. The switching activity at the j-th bit will be $\sum_{x_i} p_{x_is}x_{ij}$ if $c_j = 0$,

and $\sum_{x_i} p_{x_i s}(1 - x_{ij})$ if $c_j = 1$. Comparing these two values gives the assignment to c_j as above. The code for the duplicated state s' can be determined in the same way.

As described in the Step III of Figure 4, only when the estimated (or calculated if encoding is performed) gain exceeds a threshold, do we actually duplicate state s, compute the r-value for s and its duplicate s', update the r-values for their next states, and select the next candidate state for duplication.

4.4 Determine the Minimum Switching Activity

There are two reasons for us to determine the optimal encoding scheme for a given FSM. First, it allows us to test the quality of low power state encoding heuristics. Second, comparing the minimum switching activity of the original FSM with that of the re-constructed FSM provides us insight of FSM re-engineering approach's potential power efficiency.

The power-driven state encoding problem can be formulated as follows: *finding a code $x_{i1}x_{i2}\cdots x_{in}$ for each state x_i of a k-state FSM, such that*

$$\sum_{l=1}^{n} |x_{il} - x_{jl}| \geq 1 \tag{6}$$

and the following (total switching activity) is minimized

$$\sum_{1\leq i\leq j\leq k} p_{ij} \sum_{l=1}^{n} |x_{il} - x_{jl}| \tag{7}$$

where $p_{ij} = P_{ij} + P_{ji}$ is the total transition probability between states x_i and x_j as we have defined earlier.

Equation (6) enforces that no two states can have the same code. Expression (7) is the same as the switching activity given in Equation (2) because the Hamming distance between states x_i and x_j is defined as $H(x_i, x_j) = \sum_{l=1}^{n} |x_{il} - x_{jl}|$.

We introduce (Boolean) variables $d_{ij}^{(l)} = |x_{il} - x_{jl}|$ and $d_{ii}^{l} = 0$ for $1 \leq i < j \leq k$ and $1 \leq l \leq n$. Equations (6) and (7) can be re-written in the following linear form:

$$\sum_{l=1}^{n} d_{ij}^{(l)} \geq 1 \tag{8}$$

$$\sum_{0<i\leq j\leq k} p_{ij} \sum_{l=1}^{n} d_{ij}^{(l)} \tag{9}$$

The definition of $d_{ij}^{(l)}$ is equivalent to the following:

$$\begin{aligned} x_{il} + x_{jl} + (1 - d_{ij}^{(l)}) &\geq 1 \\ x_{il} + (1 - x_{jl}) + d_{ij}^{(l)} &\geq 1 \\ (1 - x_{il}) + x_{jl} + d_{ij}^{(l)} &\geq 1 \\ (1 - x_{il}) + (1 - x_{jl}) + (1 - d_{ij}^{(l)}) &\geq 1 \end{aligned}$$

The problem then becomes a (0-1) integer linear programming (ILP) problem and we can use the off-the-shelf ILP solver to solve it and thus determine the minimum switching activity.

5 Experimental Results

We simulate the FSM re-engineering framework on MCNC benchmark suite using POW3 as the low-power state encoding algorithm. For simplicity, we control the state duplication technique such that the encoding bits remains minimal. Therefore, no state will be duplicated for FSMs with exactly 2^k states. The 26 applicable benchmarks have states from 5 to 48. Our simulation is designed to compare POW3's performance before and after FSM re-engineering using the following metrics: switching activity (calculated from Equation (2)), power and area (simulated using SIS), overhead over the optimal (from solving the ILP problem).

Switching Activity Comparison

Table 1 reports the switching activity of the original FSMs and the re-engineered FSMs, both encoded by POW3. The second column is the length of the code, the third column lists the number of states in the original FSM, the fourth column gives the number of states duplicated by our approach, the next two columns are the switching activities before and after state duplication, the last column shows the reduction.

In the 26 benchmarks, POW3 is able to reduce the switching activity on 17 re-constructed FSMs with an average 9.4% reduction (6.0% if we take average over all 26 benchmarks). We mention that this improvement is significant. First, it is achieved over the encoding by POW3, a state-of-the-art low power encoding algorithm. Second, POW3 itself can achieve an average 12% switching activity reduction over area-driven encoding algorithms [2]. Finally, although a 13% improvement over POW3 has been reported in [16]. this improvement is based on the better one of two different encoding schemes. Moreover, their improvement is achieved at the cost of increased code length (or equivalently, the number of state registers) and the area change is not reported.

Seven benchmarks have no improvement after re-constructing the FSMs. In five of them, the encoding on the original FSM are very close to or have already achieved the minimum switching activity. For example, POW3 generates a Gray code for *bbtas*, which is the optimum in switching activity. In these cases, our algorithm correctly chooses not to duplicate any state because further improvement is unlikely to achieve. In the other four benchmarks, our algorithm duplicates one or more states but results in either no gain or negative gain in switching activity reduction. This is due to the inaccurate estimation of switching activity reduction in step III of the proposed algorithm (Figure 4). We mention that this problem can be avoided if we run POW3 every time to decide whether a state should be duplicate.

Power and Area Comparison

As one may observe from Table 1, although the code length does not increase, we do duplicate 1.7 states on average. What is the impact of this to area and power when we implement the re-constructed FSM? Table 2 reports this on the circuit implementation obtained by SIS package. We use the standard *script.rugged* to simplify the circuits and *lib2* library for technology mapping. The area is obtained by *map -s* command. The power is measured in μW using the sequential power estimation package in SIS, assuming a 5V power supply and 20MHz clock frequency.

We are able to get the area and power information from SIS on 14 benchmarks as reported in Table 2. We see that an average 9.4% power reduction is achieved at the cost of only 1.3% area increase. Interestingly, more than one third

Table 1. Total switching activity reduction on re-constructed FSMs.

FSM	*Bits*	*States*	*Dup*	POW3 Swg.		reduction
				orig.	re-eng.	(%)
example	3	5	3	1.5229	1.2703	16.6
s8	3	5	1	0.2128	0.1553	27
ex3	3	5	3	1.2	1.075	10.4
ex5	4	9	2	1.1972	1.0442	12.8
lion9	4	9	2	0.5626	0.4571	18.8
ex7	4	10	1	1.0085	0.9487	6
train11	4	11	1	0.5540	0.5087	8.2
modulo12	4	12	2	0.5833	0.5	14.3
mark1	4	12	1	0.9493	0.9342	1.58
dk512	4	15	1	1.6012	1.4167	11.5
s1	5	20	1	1.2535	1.1986	4.4
ex1	5	20	2	0.9823	0.9366	4.7
donfile	5	24	3	1.5208	1.3906	8.6
dk16	5	27	2	1.9169	1.849	3.5
styr	5	30	2	0.5302	0.5239	1.2
s510	6	47	1	0.9245	0.8868	4.1
planet	6	48	1	1.5268	1.4375	5.8
switching activity reduction over the above benchmarks: 9.4%						
beecount	3	7	0	0.5027	0.5027	0
dk14	3	7	0	1.1671	1.1671	0
bbara	4	10	0	0.3	0.3	0
pma	5	24	0	0.9112	0.9112	0
s1488	5	48	0	0.3462	0.3462	0
s27	3	6	1	0.8866	0.8866	0
s208	5	18	13	0.4751	0.4751	0
bbtas	3	6	1	0.4435	0.4565	-2.9
ex4	4	14	1	0.5921	0.6074	-2.6
switching activity reduction over all 26 benchmarks: 6.0%						

of the circuits have area reduced after state duplication. The negative power reduction occurs when the power increase in the combinational part of the circuits exceeds the reduction in the sequential part.

We also compared our power-saving results with one of the existing non-minimal length encoding algorithms reported in [16]. The comparison is made based on the improvement in dynamic power consumption over POW3. We copied their results from [16] and listed in column 8 and 9. An asterisk in a cell means the power improvement data is not reported in the paper for that benchmark. We see that our methods can achieve greater power-saving improvements over POW3 in almost all the benchmarks than both approaches presented in [16].

Comparison with Optimal Encodings

For a subset of benchmarks, we are able to find the optimal encodings for both the original and the re-constructed FSMs. This allows us to quantitatively judge the quality of the encodings obtained by POW3. Figure 5 depicts the switching activity of optimally encoded new FSM, POW3's encoding on the new FSM, and POW3's encoding on the original FSM (from bottom to top). These numbers are all normalized to the switching activity of the original FSM with the optimal encoding.

We see that although FSM re-engineering has the potential to reduce the minimum switching activity by only 2.5% on average, the power efficiency of POW3 is greatly enhanced. From Figure 5, POW3 finds codes for the original FSMs that have switching activity from 11.6% to 48.6% higher than the optimal with average 27.0%. However, when we encode the new FSMs using POW3, the average overhead drops to 10.4%. It even finds an coding that achieves the optimal in *ex3* and codings better than the original optimal in benchmarks *example* and *lion9*.

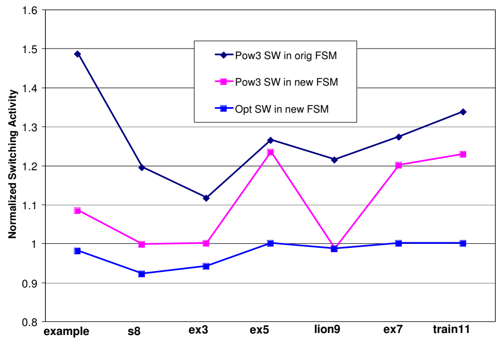


Figure 5. Switching activity of POW3's encoding schemes on the original and re-constructed FSMs and the optimal encoding (Opt) on the new FSMs. Normalized to the optimal encoding on the original FSMs.

6 Conclusions

The concept of FSM re-engineering is introduced in this paper. It is a generic framework for FSM synthesis based on the observation that minimizing the number of states in an FSM may lose the optimal solutions, or make it harder to find such solutions, for many FSM related optimization problems. To keep the discussion concrete, we study the low power state encoding problem by using a state duplication based FSM re-engineering technique. Our technique does not necessarily provide a power efficient state encoding scheme. Instead, we demonstrate its strength in enhancing the performance of any given power-driven encoding algorithms. We apply this on MCNC benchmark using POW3 as the encoding tool. Experimental results show that POW3's power in reducing circuit's total switching activity has almost been doubled by the proposed FSM re-engineering approach. Simulation on SIS indicates that an average 9.4% power reduction is achievable with only 1.3% area increase and no additional state registers. We further use an integer linear programming formulation to identify the optimal coding that achieves the minimum switching activity, where we find that the re-engineered FSMs have better optimal codes.

REFERENCES

[1] M.J. Avedillo, J.M. Quintana, and J.L. Huertas, "SMAS: a program for concurrent state reduction and state assignment of finite state machines," *IEEE International Symposium on Circuits and Systems*, pp. 1781-1784, 1991.

[2] L. Benini and G. D. Micheli, "State Assignment for Low Power Dissipation," *IEEE Journal of Solid-State Circuits*, Vol.30, pp.258-268, March 1995.

[3] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, Vol.27, pp. 473-484, April 1992.

[4] S. Devadas, H-T. Ma, R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State Assignment of Finite State Machines Targeting Multi-level Logic Implementations," *IEEE Transactions on Computer-Aided Design*, pp/ 1290-1300, December 1988.

Table 2. Area and power comparison between original FSM and reconstructed FSM

Circuit	Area		increase	Power (μW)		decrease	Power decrease in [16] over POW3	
	orig.	re-eng.		orig.	re-eng.		fast	greedy
example	43616	44544	2.1%	280.4	287.7	-2.6%	*	*
ex3	46400	50112	8%	314	282.5	10%	5.2%	7%
ex5	70528	80272	13.8%	405.2	271.9	32.9%	0%	0%
lion9	38976	45472	16.7%	178.3	165.7	7.1%	2.1%	-3.5%
ex7	78416	70064	-10.7%	405.8	287.7	29.1%	11.1%	14.6%
train11	47792	49184	2.9%	212.3	172	19%	12.3%	24.5%
mark1	94656	99760	5.4%	280.7	316.9	-12.9%	-4.2%	-21.3%
dk512	79344	81200	2.3%	430.1	408.1	5.1%	3.3%	11.7%
s1	321088	313664	-2.3%	1388.7	1210.1	12.9%	*	*
ex1	234784	213904	-8.9%	744.9	643.5	13.6%	6%	-10%
dk16	282112	254736	-9.7%	1547.3	1341.7	13.3%	4.8%	9.6%
styr	407856	421312	3.3%	1347.6	1353.1	-0.4%	*	*
s510	302064	283040	-6.3%	923.1	859.4	6.9%	*	*
planet	504832	514112	1.8%	2042.1	2081.8	-1.9%	*	*
	Average increase 1.3%			Average reduction 9.4%			Avg. 4.5%	Avg. 3.6%

[5] G. D. Hachtel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi, “Re-Encoding Sequential Circuits to Reduce Power Dissipation,” *International Workshop on Low-Power Design*, Napa, April 1994.

[6] G. D. Hachtel, B. Macii, A. Pardo, and F. Somenzi, “Probabilistic Analysis of Large Finite State Machines,” *Proceedings of the ACM Design Automation Conferences,* San Diego, CA, June 1994.

[7] G. Hallbauer, “Procedures of state reduction and assignment in one step in synthesis of asynchronous sequential circuits,” *International IFAC Symposium on Discrete Systems,* pp. 272-282, 1974.

[8] M. Koegst, G. Franke, and K. Feske, “State Assignments for FSM Low Power Design”, *Proceedings of the Conference on European Design Automation,* pp. 28-33, 1996.

[9] M. Koegst, S. Rulke, G. Franke, and M. Avedillo, “Two-Criterial Constraint-Driven FSM State Encoding for Low Power”, *Euromicro Symposium on Digital Systems Design*, Warsaw, Poland, September 2001.

[10] S. Iman, and M. Pedram, “POSE: Power Optimization and Synthesis Environment”, *Proceedings of the 33rd Design Automation Conferences,* pp. 21-26, Las Vegas, NV, June 1996.

[11] E.B. Lee and M. Perkowski, “Concurrent minimization and state assignment of finite state machines”, *International Conference on Systems Man and Cybernetics,* pp. 248-260, 1984.

[12] I. Lemberski, M. Koegst, S. Cotofana, and B. Juurlink, “FSM Non-Minimal State Encoding for Low Power,” *Proceedings of the 23rd International Conference on Microelectronics,* Yugoslavia, May 2002.

[13] B. Lin, and A. R. Newton, “Synthesis of multiple-level logic from symbolic high-level description languages,” *Proceedings of the IFIP TC 10/WG 10.5 International Conference on Very Large Scale Integration,* pp. 187-196, Federal Republic of Germany, August 1989.

[14] G. D. Micheli, R. Brayton, and A. Sangiovanni-Vincentelli, “Optimal State Assignment for Finite State Machines,” *IEEE Transactions on Computer-Aided Design,* pp. 269-285, July 1985 .

[15] G. D. Micheli, “Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-level Logic Macros,” *IEEE Transactions on Computer-Aided Design,* pp. 597-616, October 1986.

[16] W. Noth, and R. Kolla, “Spanning Tree Based State Encoding for Low Power Dissipation,” *Proceedings of the Design Automation and Test in Europe ’99,* pp. 168, Munich, Germany, March 1999.

[17] E. Olson and S.M.Kang, “State assignment for low-power FSM synthesis using genetic local search,” *Proceedings of the IEEE 1994 Custom Integrated Circuits Conference,* pp.140-143, San Diego, CA, May 1994.

[18] K. Roy and S. C. Prasad, “SYCLOP: Synthesis of CMOS logic for low power application,” *Proceedings of the International Conference on Computer Design,* pp. 464-467, October 1992.

[19] E. Sentovich, et al., “SIS: A System for Sequential Circuit Synthesis,” *Electronics Research Laboratory Memorandum, U.C.Berkeley,* No. UCB/ERL M92/41.

[20] P. Surti, L. F. Chao and A. Tyagi, “Low Power FSM Design Using Huffman-Style Encoding,” *Proceedings of IEEE European Design and Test Conference,* pp. 521-525, Paris, France 1997.

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

[22] V. Veeramachaneni, A. Tyagi, and S. Rajgopal, “Re-encoding for Low Power State Assignment of FSMs,” *International Symposium on Low Power Design,* pp. 173-178, Dana Point, CA, April 1995.

[23] T. Villa and A. Sangiovanni-Vincentelli, “NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations,” *IEEE Tranctions on Computer-Aided Design of Integrated Circuits and Systems,* pp. 905-924, September 1990.

[24] S. Washabaugh, P. Franzon, and H. Nagle, “SABSA: Switching Activity Based State Assignment,” *Proceedings of IEEE Solid State Circuits and Technology Committee Workshop on Low Power Electronics,* 1993.