An FSM Re-Engineering Approach to Sequential Circuit Synthesis by State Splitting

Lin Yuan[1], Gang Qu[2], Tiziano Villa[3,4], and Alberto Sangiovanni- Vincentelli[4,5]
[1]Synopsys Inc., Mountain View, CA 94043
[2]ECE Dept., Univ. of Maryland, College Park, MD 20742
[3]DI, Univ. of Verona, 37134 Verona, Italy
[4]PARADES, 00186 Roma, Italy
[5]EECS Dept., Univ. of California, Berkeley, CA 94720

Abstract

We propose Finite State Machine (FSM) re-engineering, a performance enhancement framework for FSM synthesis and optimization. It starts with the traditional FSM synthesis procedure, then proceeds to re-construct a functionally equivalent but topologically different FSM based on the optimization objective, and concludes with another round of FSM synthesis on the re-constructed FSM. This approach explores a larger solution space that consists of a set of FSMs functionally equivalent to the original one, making it possible to obtain better solutions than in the original FSM. Guided by the result from the first round of synthesis, the solution space exploration process can be rapid and cost-efficient.

We apply this framework to FSM state encoding for power minimization and area minimization. The FSM is first minimized and encoded using existing state encoding algorithms. Then we develop both a heuristic algorithm and a genetic algorithm to re-construct the FSM. Finally, the FSM is re-encoded by the same encoding algorithms. To demonstrate the effectiveness of this framework, we conduct experiments on MCNC91 sequential circuit benchmarks. The circuits are read in and synthesized in SIS environment. After FSM re-engineering are performed, we measure the power, area and delay in the newly synthesized circuits. In the power-driven synthesis, we observe an average 5.5% of total power reduction with 1.3% area increase and 1.3% delay increase. This results are in general better than other low power state encoding techniques on comparable cases. In the area-driven synthesis, we observe an average 2.7% area reduction, 1.8% delay reduction, and 0.4% power increase. Finally, we use integer linear programming to obtain the optimal low power state encoding for benchmarks of small size. We find that the optimal solutions in the re- engineered FSMs are 1% to 8% better than the optimal solutions in the original FSMs in terms of power minimization.

I. Introduction

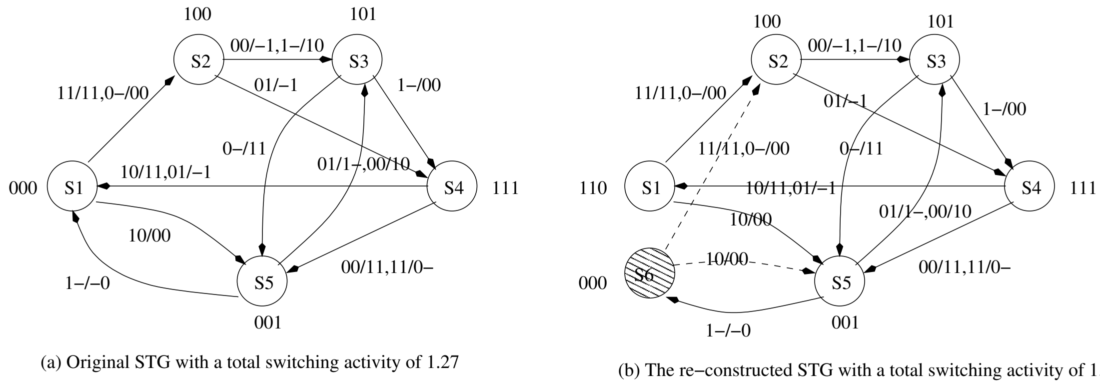
Sequential circuits, which play a major role in digital system control part, are commonly modeled by the finite state machines (FSMs). Sequential circuit synthesis, often referred to as FSM synthesis, is the process of converting the symbolic description of an FSM into a hardware implementation with certain design optimization objectives, such as speed, area, and power.

State encoding or state assignment is a crucial step in FSM synthesis. This classical optimization problem has received an extensive amount of research attention. Originally, the goal of state encoding was to minimize the circuit area and the problem was formulated as "how to simplify the logic representation of the output and next-state functions." Such work can be found in [5], [20], [6], [22], [23], [32], [34]. In the early 90's, with low power becoming an increasingly important design objective, efforts in state encoding have also been shifted to focus on power minimization in sequential circuits. Due to the well-known fact that dynamic power dissipation in a CMOS circuit is proportional to its total switching activity, state encoding problem is formulated as "minimizing the number of state bit switches per transition." This problem is known to be NP-hard and many heuristic algorithms have been proposed. Such techniques include state encoding with minimal code length [3], [26], [30], non-minimal code length [19], [24] and variable code length [28], state re-encoding approaches [8], [31], and simultaneous power and area optimization encoding [17], [25].

All of these works start with the minimized FSM, which has a minimum number of states, and then seek the best encoding scheme for these states. Such serial strategy may prevent finding optimal state encodings [11]. Therefore, conducting state minimization and state encoding in one step has been studied in [2], [10], [18], with the target of area reduction; power minimization was not considered. In addition, considering non-minimal FSMs dramatically increases the solution space for state encoding. Although better state encoding solutions may exist, it is not easy to find them without an efficient solution space exploration algorithm.

In this work, we propose FSM re-engineering, a novel approach that re-constructs a minimized FSM and re-encodes it to achieve better synthesis solutions. As we will see in the following motivational examples, the best solutions, in terms of power and area, do not necessarily come from the minimized FSM. Via the FSM re-engineering approach, better encoding solutions can be found. In fact, low power encoding

A preliminary version of parts of this manuscript has been published at the 10th Asia and South Pacific Design Automation Conference, pp. 254-259, January 2005



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

(b) The re−constructed STG with a total switching activity of 1.17

Fig. 1. Power-driven state encoding on a 5-state FSM and its functionally equivalent 6-state FSM.

on the re-engineered FSMs can give solutions with lower power consumption than the optimal encoding for the original FSMs.

A. A Motivational Example

Power minimization

We take the example from a paper on power-driven FSM state encoding [16] to show the potential of the proposed FSM re-engineering approach in power minimization.

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. For example, the edge between $S1$ and $S2$ with label "11/11,0-/00" means that at state $S1$, on input '11', the next state will be $S2$ with output '11'; on input '00' or '01', the next state will be $S2$ with output '00'. This 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 search for all the possible state encoding schemes for both FSMs and report the one that minimizes the total switching activity in Figure 1 (the 3-bit codes shown next to each state). For example, state $S1$ is encoded as '000' in the original FSM and its code becomes '110' in the re-constructed FSM. We observe that the switching activity in the FSM, an indicator of power efficiency of the encoding scheme, drops from 1.27 to 1.17 (a 7.9% reduction) after we add state $S6$. Note that the encoding scheme for the original 5-state FSM is the optimal one obtained from exhaustive search. In another word, the most energy-efficient encoding for this FSM is lost (and its functionally equivalent FSMs) once it is minimized! This implies that the optimal synthesis solution does not necessarily come from the minimized FSMs, which was observed by Hartmanis and Stearns [11].

Area minimization

The goal of sequential circuit synthesis is to implement both the output and next state as functions of the current state and input. The complexity of such functions, normally measured by the number of literals and the logic depth, has a direct impact on the area of the circuits. Area-driven state encoding aims to encode the FSM such that the output and next state functions are simplified. In this example, the complexity of a function is measured by the number of literals in a two-level logic representation.

Consider an encoded 5-state FSM shown in Figure 2(a). At current state $X_2X_1X_0$, on input I, we denote the next state to be $Y_2Y_1Y_0$ and the output to be O. We can express Y_2, Y_1, Y_0, and O as:

$$Y_2 = I'X_2'X_0 + I'X_2X_1' + IX_2'X_0' \quad (1)$$

$$Y_1 = IX_0 + X_2'X_1 \quad (2)$$

$$Y_0 = I'X_2'X_0' + X_1 + IX_0 \quad (3)$$

$$O = I'X_2' + X_1'X_0 + IX_2 \quad (4)$$

These functions have a total of 25 literals.

Now we consider the re-constructed 6-state FSM in Figure 2(b). The expressions for Y_1, Y_0, and O remain the same; however, Y_2 is simplified to (5). Compared to expression (1), we see a reduction of 2 literals. When these two FSMs are mapped to circuits in SIS, the re-constructed FSM gives a 5% area reduction.

$$Y_2 = I'X_0 + I'X_2 + IX_2'X_0' \quad (5)$$

B. FSM Re-Engineering: Goal, Novelty, and Contributions

The goal of FSM re-engineering is to improve the quality of the FSM synthesis and optimization solutions. FSM re-engineering not only provides the theoretical opportunity to produce synthesis solutions of higher implementation quality, it can also enhance practically the performance of existing FSM synthesis algorithms. For example, the power-driven state encoding heuristic POW3 [3] encodes the original 5-state FSM in Figure 1(a) with a switching activity 18.9% higher than the optimal encoding solution. However, when we use POW3 to encode the equivalent 6-state FSM, it successfully finds an encoding solution that is only 5.4% worse than the optimal.

The proposed FSM re-engineering approach is a three-phase performance enhancement framework that can be combined with most existing FSM synthesis techniques. We first obtain a synthesis solution using existing tools. Then we analyze this

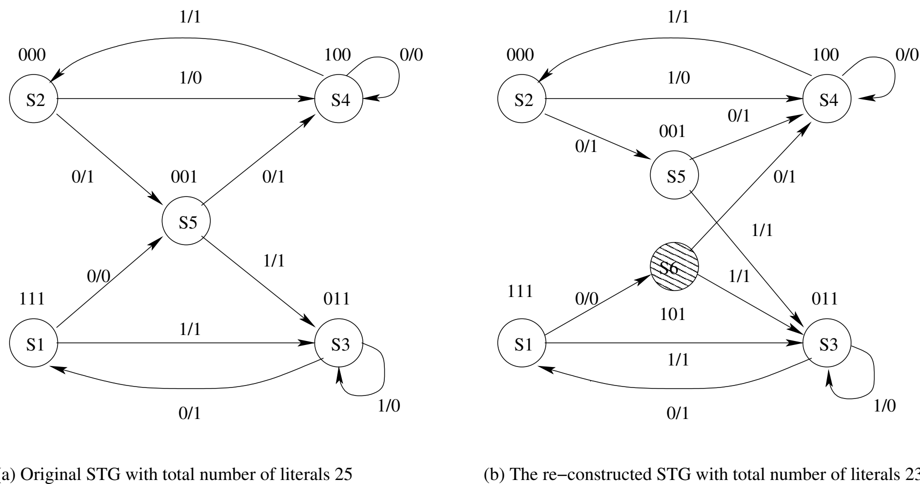


Fig. 2. Area-driven state encoding on the original and re-constructed FSMs.

solution and re-construct functionally equivalent FSMs based on such analysis. Finally we apply the same synthesis tools on the re-constructed FSM to obtain a new solution. Compared to techniques such as state re-encoding that reshape the synthesis solution based on the minimized FSM, our approach has the potential to find solutions of higher quality because it investigates a larger solution space that includes functionally equivalent non-minimized FSMs. Compared to techniques such as simultaneous state minimization and encoding that are not restricted to the minimized FSM, our approach selectively enlarges the solution space. Although this will not guarantee to find an optimal solution, its high efficiency makes it applicable to large machines.

In this paper, we demonstrate this approach on power-driven and area-driven state encoding algorithms. We propose efficient and practical methods to re-construct functionally equivalent FSMs. We conduct extensive experiments and carefully analyze the results to show the effectiveness of our approach. To address the natural concerns of overhead and search cost we mention the following:

- Overhead: Implementing the non-minimized FSM may require more hardware which increase the design costs. However, this is not always true. For example, a 36-state FSM and a 42-state FSM need the same number of latches (flip flops, or state registers). Furthermore, it is known that synthesis of minimized FSMs does not guarantee the optimality in implementation [11]. One example is the one-bit hot encoding scheme that we will survey in the next section.
- Search cost: Although the solution quality can be improved theoretically if one searches in a larger solution space that includes all the (non-minimized) functionally equivalent FSMs, such exploration may become infeasible as the searching complexity increases. Our FSM re-construction is done after the first round of synthesis and driven by the optimization objective. Therefore, our search is actually restricted to a subset of the functionally equivalent FSMs that might yield good solutions. This reduces the search cost dramatically.

C. Paper Organization

In Section II, we survey the most relevant works on FSM state encoding and discuss their relationship with the proposed FSM re-engineering framework. We give the notation and problem formulation in Section III. The generic FSM re-engineering approach is presented in Section IV together with two heuristic algorithms and a genetic algorithm to re-construct FSMs. We report the experimental results in Section V and conclude in Section VI.

II. Related Work

FSM synthesis is a well studied area that includes state minimization and state encoding. For a given FSM, state minimization (SM) aims to find another FSM that has the same input/output behavior as the original machine and has the minimum number of states. On the other hand, state encoding (SE) is to find a binary code for each state of the FSM such that the FSM (i.e., the next-state and output functions) can be efficiently implemented with a given technology library, where the efficiency is measured by speed, area, power, testability, and so on.

Although there are several standard methods to solve the SM problem [15], SE algorithms evolve as the design objectives and implementation platforms changes. In this section we focus our review on SE problem, in particular on SE for area and power minimization.

A. State Encoding for Area

Given the computational challenge of the problem, whose version for truth tables in NP-complete and for sums of cubes is Σ_p^2-complete [33], [35], the literature is huge and we mention here only a few landmark contributions, referring to [33] for a complete discussion.

The earliest state encoding algorithms date back to 1960s [1], and were based on grouping together 1s in the Karnaugh tables of the resulting binary output and the next-state functions in order to minimize the number of product terms in the sum-of-product expressions describing the combinational

part of the FSM realization. An important area of research was state assignment of asynchronous circuits to avoid critical races [29].

Later, De Micheli et al. formulated the minimum area state encoding problem for PLA realization as generating a minimum (multi-valued) symbolic cover of the FSM followed by a step of satisfying the encoding constraints. The idea was implemented in KISS [22], with a restriction to input constraints and a heuristic row encoding technique. Successive extensions, from CAPPUCCINO [22] to NOVA [32] and ESP-SA [34] introduced also output constraints and more efficient algorithms to satisfy the input and output encoding constraints.

MUSTANG [5] is one of the earliest state encoding techniques for multi-level logic minimization; it assigns a weight to each pair of symbols and gives adjacent codes to pairs of states with large weight. JEDI [20] and MUSE [6] adopt a weighted graph model similar to the one in MUSTANG, but JEDI uses a simulated annealing algorithm to perform the embedding and MUSE starts from a multi-level representation of the FSM to derive the weights. Instead MIS-MV [21] applies the multi-valued minimization paradigm followed by extraction and satisfaction of encoding constraints, generalizing it to multi-level logic, i.e., it applies multi-level minimization first to the combinational component of FSM when the state variable is still in its symbolic, multi-valued form and then it derives input constraints.

All the above approaches perform state encoding after state minimization. Concurrent state minimization and state encoding was suggested in [2], [7], [10], [18], [10] with little success. Hallbauer et al. [10] proposed a method for asynchronous circuits based on pseudo-dichotomies trying to perform state minimization while heuristically reducing the encoding length, with no reported results. To explore the solution space in the non-minimized FSM, Lee's method [18] employs a branch-and-bound technique; however, it is only feasible for very small machines (no more than sixteen states). Avedillo et al. [2] presented a heuristic method in which the encoding is generated incrementally, and may create incompletely specified codes for the states in the original FSM. Although reasonably efficient, the experimental results on a subset of the MCNC benchmarks do not show improvements over a serial synthesis strategy. Fuhrer et al. proposed OPTIMIST [7], a concurrent state minimization and state encoding algorithm for two-level logic implementation. It provides an exact solution to FSM optimization for two-level logic implementation. But the largest FSM in the reported table of results has only nine states, and the authors pointed out that their algorithm does not scale well.

Our three-phase FSM re-engineering approach can be applied to improve most of the above state encoding techniques (JEDI is used in our experiments), where an accurate and efficient cost function is defined. It allows the state encoding algorithms to explore functionally equivalent non-minimized FSMs; but unlike the simultaneous state minimization and state encoding strategy, our approach is more efficient in solution space exploration and so is capable of handling larger FSMs.

B. State Encoding for Low Power

Dynamic power dissipation in CMOS circuits is caused by the charging and discharging of capacitive loads, also known as switching activity, in both sequential logic and combinational logic. Switching activity in sequential logic mainly comes from the switching in the state registers [30] and can be described as

$$P = \frac{1}{2} V_{dd}^2 f \sum_{i \in sb} C(i) E(i) \tag{6}$$

where V_{dd} is supply voltage, f is clock frequency, $C(i)$ is the capacitance of the register storing the i-th state bit, and $E(i)$ is the expected switching activity of the i-th state register.

The switching activity in combinational logic is the total switching at each logic gate. Therefore, it depends on the complexity of the implementation of the combinational logic. However, it is hard to estimate, at FSM synthesis level and before technology mapping, the impact of state encoding on switching activity in combinational circuit. Normally, the complexity of the next-state and output functions, in terms of the number of literals, is used to estimate the power dissipation in the combinational logic. This complexity is also used as the indicator of the circuit area. So in the following, we briefly survey power-driven state encoding algorithms that reduce the switching activity in state registers and hereby power.

Roy and Prasad proposed a simulated annealing based algorithm to improve any given state encoding scheme [26]. Washabaugh et al. suggested to first obtain state transition probability, then build a weighted state transition graph, and finally apply branch and bound for state encoding [36]. Olson and Kang presented a genetic algorithm, where in addition to the state transition probability, they also considered area while encoding in order to achieve different area-power trade-offs [25]. Benini and De Micheli presented 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 [3]. Iman and Pedram developed a power synthesis methodology and created a complete and unified framework for design and analysis of low power digital circuits [13].

Unlike these power-driven state encoding algorithms, low power state re-encoding techniques start from an encoded FSM and seek a better encoding scheme to reduce switching activity. Hachtel et al. recursively used weighted matching and min-cut bi-partitioning methods to re-assign codes [8]. Veeramachaneni et al. proposed to perform code exchange locally to improve the coding scheme's power efficiency [31]. Our FSM re-engineering approach is conceptually different from re-encoding in that we do not only re-assign codes to the existing states, but also change the topology of the FSM.

The above work is based on two common assumptions: 1) the length of the state code is minimal, that is, the number of bits to represent a state will be $\lceil \log n \rceil$ for an n-state FSM; 2) state encoding (or re-encoding) should be performed after state minimization. A couple of recent papers on non-minimal length encoding algorithms show that power may be improved with code length longer than the lower bound [19], [24]. These methods require extra state register(s) in

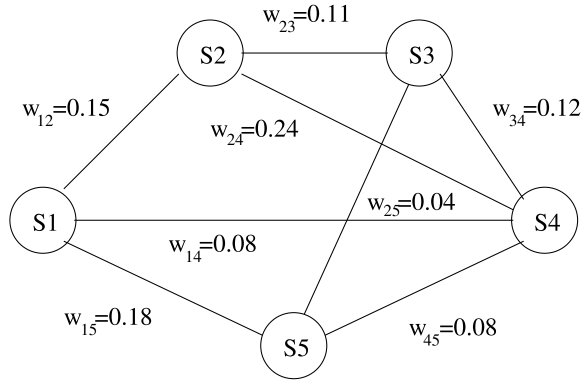


Fig. 3. Weighted STG of the original FSM in Fig. 1; the weights are calculated based on transition probability.

the FSM implementation which will add to the hardware cost and cause area increase. Unfortunately, none of the papers reported the area overhead. One of the advantages in our approach is that usually no extra state bit is required (when the number of states is not 2^k), so The hardware overhead can be kept to a minimum. Besides, as we mentioned earlier, our technique is a stand-alone enhancement tool. Therefore, it can be applied to non-minimal length encoding algorithms to find better solutions as well.

Finally, we mention one-hot encoding where each state in an n-state FSM receives an n-bit code with exactly one bit set to '1'. This encoding scheme can greatly simplify the logic implementation of the FSM and also reduce the switching activity because it guarantees that the Hamming distance of every pair of states is exactly two. However, it requires that the code length is the same as the number of states, which makes it impractical for large FSMs.

III. Definitions and Problem Formulation

A. FSM representation

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. We transform this directed graph G to an undirected weighted graph $\tilde{G} = (V, \tilde{E}, \{C_i\}, \{w_{ij}\})$:

- 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 label of node $v_i \in V$, which is the code of state s_i;
- w_{ij}, the weight of edge $(v_i, v_j) \in \tilde{E}$, which depends on the optimization objective ($w_{ij} > 0$).

We denote $H(v_i, v_j)$ as the Hamming distance between the codes (vectors of 0s and 1s), C_i and C_j, of states s_i and s_j, under the given encoding scheme. We define the weighted sum of an encoded FSM as:

$$weighted_sum = \sum_{(v_i,v_j)\in\tilde{E}} w_{ij} H(v_i, v_j) \quad (7)$$

Equation (7) has a significant implication for state encoding, because in many state encoding algorithms, both power-driven and area-driven, such weighted sum is used as the objective function in logic optimization. In power-driven state encoding algorithms, like POW3 [3], GALOPS [25], SYCLOP [26] and SABSA [36], the weight is defined by the transition probabilities between two states; in area-driven state encoding, such as MUSTANG [5], JEDI [20] and PESTO [12], the weight is calculated through the adjacency matrices. Due to this reason, we simply refer to the weighted sum of Hamming distances as the "cost" in the FSM state encoding algorithms based on minimum weighted Hamming distance (MWHD). We show in Figure 3 a weighted STG of the original FSM in Figure 1.

B. Calculating the Weight

Power optimization Since the dynamic power is proportional to the total switching activity, the weight on an edge is defined as the total transition probability between its two ending states. This measures how frequently each transition occurs; if two states have frequent transitions between them, they should be assigned adjacent codes to reduce the total number of switching bits. The weight is expressed as:

$$w_{ij} = P_{ij} + P_{ji} \quad (8)$$

where P_{ij} is the transition probability from state s_i to state s_j.

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 [36]. This gives us $p_{j|i}$, the conditional probability that the next state is s_j if the current state is s_i. Then a Markov chain can be built 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 [9]. Now we can obtain the steady-state 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

$$P_{ij} = p_{j|i} P_i \quad (9)$$

Area minimization There are several different methods to define the weight. The most popular ones are the fanout oriented and fan-in oriented algorithms [5], [6], [20]. The fanout oriented algorithm is as follows:

1) For each output o build a set O^o of the present states where o is asserted. Each state p in the set has a weight OW^o_s that is equal to the number of times that o is asserted in s (each cube under which a transition can happen appears as a separate edge in the state transition graph).
2) For each next state n build a set N^n of the present states that have n as next state. Again each state s in the set has a weight NW^n_s that is equal to the number of times that n is a next state of s multiplied by the number of state bits (the number of output bits that the next state symbol generates).

3) For each pair of states k, l define the weight of the edge joining them in the weight graph as $w_{kl} = \sum_{n \in S} NW_k^n \times NW_l^n + \sum_{o \in O} OW_k^o \times OW_l^o$.

This algorithm gives a high weight to present state pairs that have a high degree of similarity, measured as the number of common outputs asserted by the pair.

The fan-in oriented algorithm is almost symmetric with the fanout oriented algorithm [33].

C. Problem Formulation

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 state the FSM re-engineering problem as:

Given an encoded FSM M and its corresponding weighted graph $\tilde{G} = (V, \tilde{E}, \{C_i\}, \{w_{ij}\})$, construct an equivalent FSM M' and encode it so that in the corresponding graph $\tilde{G}' = (V', \tilde{E}', \{C_i'\}, \{w_{ij}'\})$, the total cost reduction is maximized:

$$\sum_{(v_i, v_j) \in \tilde{E}} w_{ij} H(v_i, v_j) - \sum_{(u_i, u_j) \in \tilde{E}'} w_{ij}' H(u_i, u_j) \qquad (10)$$

The FSM re-engineering problem targets the re-construction and encoding of a functionally equivalent FSM for a better implementation. We set the optimization objective as maximizing the difference in cost between the original and re-constructed FSMs rather than minimizing the cost in the new FSM because these two objectives are essentially equivalent and it is more efficient to calculate the cost difference in our implementation. Clearly, the problem is NP-hard because it requires the best state encoding for the re-constructed FSM M', which itself is an NP-hard problem (it is NP-hard for minterm-based representations; it is worse, i.e., Σ_p^2-hard for representations based on sum of products, see [33], [35]). Furthermore, when we restrict M' to be the same as M, the problem becomes "determining a new encoding scheme to minimize the total cost", which is the 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 a better FSM encoding. In this paper, we focus on FSM re-construction and defer the state encoding problem to existing algorithms. It is also possible to extend the problem formulation to non-encoded FSMs, which would further enlarge the search space and could lead to better solutions at an higher search cost. We now give an example on how to re-engineer an FSM and explain how it can effectively lead to better solutions.

D. An Example of Re-constructing FSMs

We have already seen from Figures 1 and 2 how to add a new state to the FSM without altering its functionality. Figure 4 illustrates a systematic way to do so. We see that a new state, S', is added as a split of state S in the following way: 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'. In the rest of the paper, we will call this process *state splitting*.

To see the advantage of this non-minimized FSM, we consider a scenario where state S has a large Hamming distance to one of its previous states Sp_j and the transition from Sp_j to S contributes a lot to the total cost. In the re-constructed FSM, we can redirect the next state of this transition to S' and assign S' a code with a small Hamming distance to Sp_j.

For example, in Figure 4, no matter which code we assign to state S, it will have a Hamming distance three or larger to at least one of its previous states. (To see this, notice that both codes 11111 and 00000 are assigned to its previous states). However, in the re-constructed FSM, we can assign code 11110 and 00001 to state S and its equivalent S', respectively. This ensures that S will have Hamming distance one from all of its previous states, and S' will have Hamming distance two from S_4 and distance one from all the other previous states.

IV. FSM Re-engineering Algorithm

In this section, we elaborate the FSM re-engineering approach by showing how the state splitting technique can improve state encoding algorithms. We first propose two heuristic algorithms, based on Hamming distance, on how to select a state for splitting and how to split the selected state. We then present a genetic algorithm for state splitting to target more general cost functions that may not be based on Hamming distance. Finally, we describe an integer linear programming (ILP) method that can find the most power-efficient state encoding to evaluate our proposed FSM re-engineering approach.

A. A Generic Approach

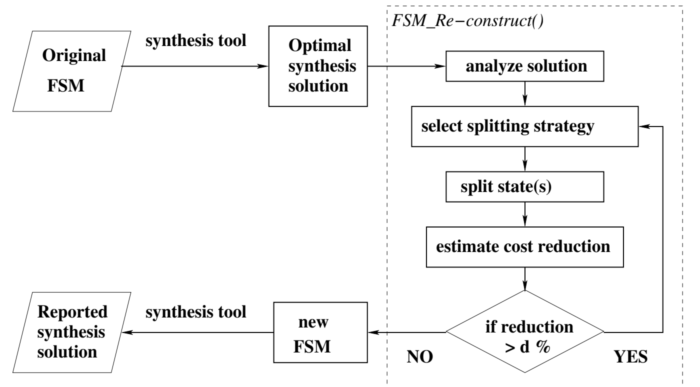


Fig. 5. FSM re-engineering flow.

Figure 5 outlines the proposed FSM re-engineering approach. This three-phase approach can be used to improve the performance of FSM synthesis tools for different design objectives. First, we apply an existing synthesis tool to obtain an "optimal" solution (i.e., the best based on the tool we use) for the given FSM. The second phase is FSM re-construction based on this synthesis solution. In the third and last phase, the re-constructed FSM is re-synthesized using the same synthesis tool to obtain a new synthesis solution.

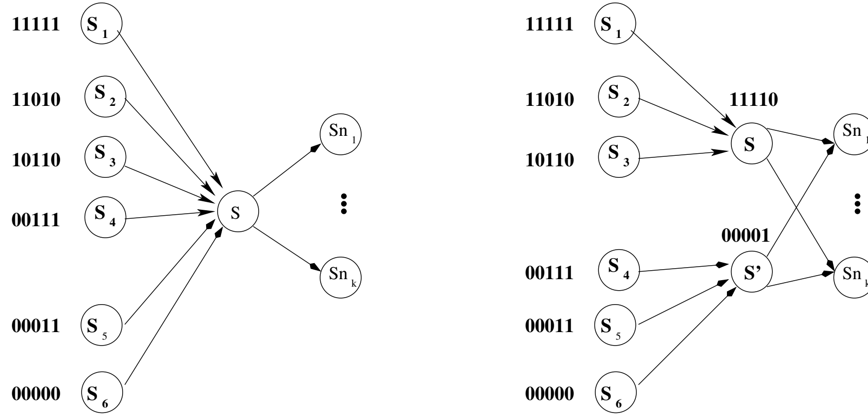


Fig. 4. Re-constructing an FSM by splitting state S and its incoming edges.

The FSM re-construction phase starts with solution analysis where we evaluate the solution based on a given cost function. Such cost function depends on the synthesis objective and one can compute or estimate it for a given (encoded) FSM. For example, in Section 3.2, we have shown that both power minimization and area minimization can be formulated as a minimum weighted Hamming distance problem. The cost function in this case can be defined as the total switching activity and the total state adjacencies, respectively. It is crucial to have a cost function that can model the design objective accurately and can be computed efficiently. We study the feature of the current FSM contributing most to the cost function and re-construct the FSM accordingly. For this purpose, we can utilize options inherently existing in the FSM such as compatible sets in an incompletely specified FSM and the don't care conditions on state transitions. We can also explore the class of FSMs that are functional equivalent to the original FSM. This process can be performed iteratively as well to locate the most promising re-constructed FSM. Figure 5 illustrates this phase by the state splitting technique. In the rest of this section, we will illustrate the three key steps of this technique:

1) select the best candidate state for splitting.
2) decide how to split the selected state.
3) estimate the (maximum) cost reduction after the state splitting.

We will use power optimization as an example to describe the algorithms. Since the cost function of both power and area optimization can be formulated as minimizing a weighted sum of Hamming distances, the algorithms can be easily adopted for area minimization by modifying the weight definition.

Finally, we mention that the strength of FSM re-engineering is to improve the performance of FSM synthesis and optimization tools/algorithms. This can be seen from Figure 5 as we use the same algorithm, which gives us the input encoded FSM, to encode the re-constructed FSM and produce the encoded FSM. In our simulation, POW3 developed by Benini and De Micheli [3] is used as the power-driven state encoding scheme; and JEDI developed by Lin and Newton [20] is used as the area-driven state encoding scheme.

B. Heuristic for Selecting States to Split

As shown in Figure 4, state splitting enables us to assign different codes to the original state and the new split state. However, as it has also been implied in Section 3, choosing an optimal state splitting strategy is also NP-hard. The reason is that it is necessary to encode the split states optimally first to determine whether a state splitting strategy is optimal. This necessary condition itself is already known as NP-hard.

Therefore we describe heuristic algorithms to select and split states. Intuitively, states with large (average) Hamming distance from their previous states will benefit from state splitting because they will have fewer previous states in the re-constructed FSM, which allows the encoding scheme to find a better code to reduce the Hamming distance. Because both the selected state and its split state will be connected to all the next states to preserve the FSM's functionality, the codes of the next states play a less important role in selecting which state to split. However, the next states will impact the codes for the selected state and its split states when we encode the re-constructed FSM.

For each state s_i, we define:

$$r(s_i) = \sum_{(v_j,v_i)\in E} H(v_i, v_j)/\text{indgree}(v_i) \qquad (11)$$

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

This value measures the average Hamming distance between state s_i and all its previous states. We split one state at a time and each time we select the state according to the following rules:

1) select the state with the largest r-value;
2) if there is a tie, select the state with fewer previous and/or next states;
3) if the tie still exists, break it by selecting a state randomly.

Rule 1) helps us to locate the state(s) such that state splitting can give us a large gain in reducing Hamming distance. Rule 2) helps the encoding engine to select a code for the split states that minimizes the average Hamming distance from its previous states. It also helps to reduce the additional switching activities created between the split state and its next states.

C. Heuristic for Splitting a Selected State

We now present a heuristic algorithm that splits the selected state. Ideally, we want to split 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 split a state to minimize switching activity locally.

More specifically, let s be the state we select for splitting, PS and NS be the sets of previous states and next states of s respectively in the original FSM. The state splitting 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 makes them as the previous states for s and s' in the new FSM. The goal of such local state splitting 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.

Local Algorithm to Split a State

```
/* Split state s */
1.  for each pair s_i and s_j in PS, the previous states of s
2.     compute the Hamming distance H(s_i, s_j);
3.  pick s_1 and s_2 s.t. H(s_1, s_2) = max_{s_i,s_j ∈ PS} {H(s_i, s_j)};
4.  c_1 = s_1; c_2 = s_2;
5.  do
6.    PT_1 = {c_1}; PT_2 = {c_2};
7.    for each state t ∈ PS
8.       if (H(t, c_1) < H(t, c_2))
9.            PT_1 = PT_1 ∪ {t};
10.      else PT_2 = PT_2 ∪ {t};
11.   c_1 = center of PT_1; c_2 = center of PT_2;
12.   H_total = Σ_{t∈PT_1} H(t, c_1) + Σ_{t∈PT_2} H(t, c_2);
13. while (H_total is decreasing);
14. for each state t ∈ PT_1
15.    add t as a previous state of state s;
16. for each state t ∈ PT_2
17.    add t as a previous state of state s';
18. for each state t ∈ NS, the next state of s
19.    add t as a next state of state s';
```

Fig. 6. Pseudo code: Split a State

We propose a greedy heuristic to partition the previous states in PS into two clusters, which is shown in Figure 6. The algorithm first puts the two states s_1 and s_2 with the largest Hamming distance into clusters PT_1 and PT_2, respectively (line 3-4). For each of the other states $t \in PS$, we assign t to PT_1 if it is closer to s_1 in terms of Hamming distance, or PT_2 if it is closer to s_2 (lines 6-9). We define the *center* of a cluster to be the code that has the minimum total Hamming distance from all the states in the cluster. Suppose that a cluster has n states and the code for the i-th state is $c_{1i}c_{2i}\ldots c_{ki}$, then the center is $c_{1c}c_{2c}\ldots c_{kc}$, where $c_{jc} = MAJ(c_{j1}, c_{j2}, \ldots, c_{jn})$ and MAJ is the majority function [1]. For example, the center for codes '00', '01' and '11' will be '01'. After we partition the previous states into two clusters, the centers c_1 and c_2 of the two clusters (line 11) are computed. We then re-partition set PS based on these new centers and continue if the new partition results in reduced total Hamming distance (line 13).

The following lemma upper bounds the run-time complexity of this algorithm.

Lemma 1: The run-time complexity of the procedure in Figure 6 is linear in kn, where k is the size of set PS and n is the encoding length.

[Proof]. For the loop (lines 6-12) to be repeated, it is necessary that the total Hamming distance is reduced by at least 1. Therefore, this loop will stop after being repeated a finite number of times, upper bounded by H_{total}. Furthermore, the largest Hamming distance from s (or its equivalent s') to any state in PS is n, that is the encoding length. If there are k states in PS, then the loop will not be executed more than kn times.

This procedure is illustrated in Figure 4. In the original FSM, state S has six encoded previous states. S_1 and S_6 have the largest Hamming distance and are put into two subsets. The remaining states from S_2 to S_5 are partitioned according to line 7 to 10 in Figure 6. Then the center in the first subset (that contains S_1) is calculated as "11110"; the center in the second subset (that contains S_6) is "00001". Then all the states are assigned again based on their Hamming distance to each center. In this round, state S_4 is moved from the first subset to the second subset because it has a smaller Hamming distance, that is 2, to the center in the second subset. There will be no partition afterwards because the total Hamming distance has reached a minimum.

D. A Genetic Algorithm for Selecting and Splitting States

As the technology scales down and the number of transistors on chip rises so fast, power has become the major design concern in CMOS circuits design and synthesis. Since the total dynamic power includes both sequential power and combinational power, minimizing the total switching in state registers alone may not be optimal in power. As the power in combinational part is proportional to area, we want to modify our cost function such that total switching is minimized with area under control. To do this, we simply combine the weight function for power and area minimization using a linear model: $w_{ij} = \alpha * w_{ij}^{sw} + (1-\alpha) * w_{ij}^{ar}$.

[1]Suppose that one partition has k states with codes $\{x_{i1}x_{i2}\cdots x_{in} : i = 1,2,\cdots,k\}$ whose next state will be s 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 $\{x_{ij} : i = 1,2,\cdots,k\}$.

Figure 7 depicts the proposed genetic algorithm that searches for a good state splitting strategy. Since splitting a state with only one previous state does not help in reducing the Hamming distance between that state and its previous state, we eliminate all the states with a single previous state from the queue of states to be split (line 1-3). For the 5-state FSM in Figure 1(a), the candidate queue for state splitting is $\{S1, S3, S4, S5\}$.

A state splitting scheme is represented by a boolean vector of the same length as the above candidate queue. A bit '1' at the ith position of the vector indicates that the ith candidate state is split and a bit '0' means that the scheme chooses not to split this state. For example, the 6-state FSM in Figure 1(b), where state $S1$ is split, corresponds to vector 1000. Each vector is referred as a *chromosome*.

According to each *chromosome*, we split the states (lines 7-9) and calculate its *fitness* (line 10), which is defined as the total switching activity according to that *chromosome*. The smaller the total switching activity, the better the *chromosome*. We start with an initial population of N randomly generated chromosomes (line 5). Children are created by the *roulette wheel* method in which the probability that a *chromosome* is selected as one of the two parents is proportional to its fitness (line 13). With a certain ratio, *crossover* is performed among parents to produce children by exchanging substrings in their *chromosomes*. A simple *mutation* operation flips a bit in the *chromosome* with a given probability known as *bit mutation rate* (line 14). When the population pool is full, i.e., the number of new *chromosomes* reaches N, the algorithm stops to evaluate fitness of each individual for the creation of next generation. This process is repeated for MAX_GEN times and the best *chromosome* gives the optimal state splitting strategy.

Genetic Algorithm

```
/* Traverse STG and split states. */
 1. for each state in STG
 2.    if it has more than one incoming edge
 3.       put it in candidate queue;
 4. chromosome_length = the size of candidate queue;
 5. initialize N random vectors;
 6. while generation < MAX_GEN
 7.    for each chromosome v_k and for each i
 8.       if v_k[i] == 1
 9.          split the i^th candidate state;
10.       v_k.fitness = total cost;
11.    do
12.       sort chromosome by non-decreasing fitness;
13.       roulette wheel selection to select parents;
14.       crossover & mutate to create children;
15.    until number of new chromosomes = N
```

Fig. 7. Pseudo code: State splitting via genetic algorithm

We will discuss how to calculate the *fitness* of each *chromosome*, that is, the total cost for a new FSM with certain states split.

A way to calculate the cost in each step after state splitting is to encode the re-constructed FSM and to compute its total cost as discussed in Section 3. This gives the actual gain in total cost reduction by splitting a set of states. When it is too expensive to apply the state encoding algorithm on the entire FSM, we use the following alternative: we assign locally to the new state the "best" code (notice that it might not be feasible) and calculate the lower bound for its cost. Here we trade accuracy for efficiency, since a lower bound may not always lead to the optimal solution.

Lemma 2: Let $\{x_i : (x_{i1}x_{i2}\cdots x_{in})\}$ be the set of states that have transitions to/from state s and their codes. Let w_{x_is} be the weight between states x_i and s. The total cost is minimized at state s when it has code $c_1c_2\cdots c_n$, where

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

[Proof]. From the definition, the switching activity at the j-th bit will be $\sum_{x_i} w_{x_is}x_{ij}$ if $c_j = 0$, and $\sum_{x_i} w_{x_is}(1-x_{ij})$ if $c_j = 1$. Comparing these two values, we conclude that c_j should be assigned 1 if $\sum_{x_i} w_{x_is}(1-x_{ij}) < \sum_{x_i} w_{x_is}x_{ij}$, which yields the result as above.

E. ILP Computation of the Minimum Switching Activity

There are two reasons to determine the optimal encoding scheme for a given FSM. First, it allows 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 insights on the potential of FSM re-engineering approach in power minimization.

The power-driven state encoding problem can be formulated as follows: *find a code $x_{i1}x_{i2}\cdots x_{in}$ for each state $x_i, i = 1,\ldots,k$, of a k-state FSM, such that*

$$\sum_{l=1}^{n} |x_{il} - x_{jl}| \geq 1, \forall i \neq j \quad (12)$$

and the following (total switching activity) is minimized

$$\sum_{1\leq i<j\leq k} p_{ij} \sum_{l=1}^{n} |x_{il} - x_{jl}| \quad (13)$$

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

Equation (12) enforces that no two states have the same code. Expression (13) represents the weighted sum of Hamming distances, 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 (12) and Expression (13) can be re-written in the following linear form:

$$\sum_{l=1}^{n} d_{ij}^{(l)} \geq 1 \quad (14)$$

$$\sum_{1\leq i<j\leq k} p_{ij} \sum_{l=1}^{n} d_{ij}^{(l)} \quad (15)$$

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 that can be solved by an off-the-shelf ILP solver.

V. Experimental Results

We demonstrate the FSM re-engineering framework on state encoding of FSM benchmarks from the MCNC91 suite. Our simulation is designed to show the performance enhancement of FSM re-engineering on a given state encoding algorithm. Based on different optimization objectives, we run FSM re-engineering with a power-driven state encoding algorithm and an area-driven state encoding algorithm, respectively. We compare the cost function before and after FSM re-engineering in each optimization framework. Then we synthesize the sequential circuits in the SIS environment [27] as follows: we first read in the minimized FSM in *kiss2* format. Then, we use the 'state_assign' command to encode the FSM by one of the encoding algorithms. Next, we run the 'source script.rugged' command to further optimize the circuit and map the circuit to a general library 'lib2.genlib' using the command 'map -m -s'. Area and delay results are reported after technology mapping. The delay is computed using the average of the maximal rising and falling arrival times. Afterwards, we measure power consumption by running the command 'power_estimate -t SEQUENTIAL' in SIS.

A. Power driven optimization

We implemented the power-driven POW3 [3] algorithm for state encoding. We use the Markov model to calculate the transition probability between states (for that the FSM not only has to be a Markov chain, it also must be strongly connected, i.e., every state has to be reachable from every other state). For this reason, some of the benchmarks that have non-deterministic transitions are excluded, and the unreachable states in some benchmarks are removed or modified. We run the experiments on 25 benchmarks that can be encoded using our POW3.

To compare the performance of POW3 before and after FSM re-engineering, we use the following metrics: switching activity (calculated from Equation (7)) and sequential power (simulated in SIS). We also compare the performance enhancement of POW3 by FSM re-engineering with the reported literature on comparable cases.

Switching Activity Reduction

Table I reports the switching activity reduction by the proposed FSM re-engineering algorithms and their run-times. The first column lists the 25 FSM benchmarks from MCNC91; the second column shows the number of states in each original FSM; the third column shows the number of states split by the state splitting strategy based on the heuristic algorithm and on the genetic algorithm, respectively. We note that, for several benchmarks, the GA algorithm creates many more split states than the heuristic one. This is because the heuristic state splitting algorithm is a greedy approach that only proceeds when there is enough reduction in switching activity. On the other hand, the genetic algorithm works more globally. Therefore, if more than one state needs to be split to reduce the switching activity, the heuristic may be unable to do so, whereas the genetic algorithm may succeed in finding multiple splits. For example, in s1488, five states need to be split to achieve a 0.2% reduction in switching activity. The greedy heuristic algorithm, in this case, cannot find any single state whose splitting may lead to sufficient switching activity reduction.

The switching activities in the original FSMs encoded by POW3 are shown in the fourth column. Column 5 and 6 report the switching activity reduction over POW3 in FSMs re-engineered by our heuristic algorithm and genetic algorithm, respectively. In the genetic algorithm, we chose the population size to be 11 and the number of generations to be 30. We notice that the genetic algorithm performs worse than heuristic on lion9. This is due to the nature of the genetic algorithm. The results can be improved by increasing the population size or the number of iterations at the cost of run-times. In our experiment, we found the current setting gives us good results for most benchmarks with moderate run-times.

The next two columns list the switching activity reduction over POW3 by two non-minimal length low-power state encoding algorithms for low power reported in [24]. The asterisks in these two columns mean that the results are not available from [24]. POW3 is a state-of-the-art low power state encoding algorithm; it achieves an average of 16% switching activity reduction over non-power-driven encoding algorithms [3]. The two non-minimal length low-power state encoding algorithms proposed in [24] claim average of 2.3% and 6.6% more reduction over POW3, respectively. When we apply the FSM re-engineering framework to POW3, we see that for most of the benchmarks, we are able to further reduce the switching activity over the solution provided by POW3. The average reduction is 8.5% for the genetic algorithm and 5.7% for the heuristic algorithm.

Finally, in the last three columns in Table I, we report the run-times of POW3 and the proposed FSM re-engineering algorithms. Note that both algorithms invoke POW3 encoding in their cost estimation step (see Figure 5) in order to obtain an accurate estimation on switching activity reduction. We are able to do this because of the small run time of the synthesis tool, POW3 in this case. When the synthesis tool is time consuming, we can reduce the frequency of invoking the tool or use only the switching activity reduction estimation method we proposed earlier. Therefore, the run-times of FSM re-engineering are susceptible to the run-times of the encoding algorithm, in this case POW3. The heuristic algorithm can run all the benchmarks with average run-times less than one second; the genetic algorithm can finish most of the benchmarks within one minute.

TABLE I

SWITCHING ACTIVITIES IN THE ORIGINAL FSMS ENCODED BY POW3 (COLUMN 4) AND THE REDUCTION IN FSMS RE-ENGINEERED BY THE GENETIC ALGORITHM (COLUMN 5) AND HEURISTIC ALGORITHM (COLUMN 6), COMPARED WITH THE REDUCTION BY TWO NON-MINIMAL STATE ENCODING ALGORITHMS PROPOSED IN [24] (COLUMN 7 AND 8); THE RUN-TIMES OF POW3 ALONE AND THE TWO FSM RE-ENGINEERING ALGORITHMS THAT INVOKE POW3 ENCODING ITERATIVELY (COLUMN 9 TO 11).

FSM	# states	# splits (ga/heu)	switching activity					run-time(s)		
			pow3	ga	heu	fast[24]	greedy[24]	pow3	ga	heu
s8	5	2/1	0.22	34.4%	27.0%	-16.9%	0.0%	0.01	0.22	0.03
ex3	5	2/3	1.20	10.9%	10.4%	14.1%	19.5%	0.01	0.24	0.05
s27	6	2/1	0.89	4.3%	0.0%	0.0%	0.0%	0.01	0.26	0.02
bbtas	6	0/0	0.44	0.0%	0.0%	0.0%	0.0%	0.01	0.24	0.02
beecount	7	0/1	0.50	0.0%	0.0%	0.0%	2.1%	0.01	0.61	0.02
dk14	7	1/0	1.17	3.7%	0.0%	10.5%	10.5%	0.01	0.79	0.01
ex5	9	2/2	1.20	12.8%	12.8%	0.0%	0.0%	0.01	1.72	0.03
lion9	9	1/2	0.56	11.4%	18.8%	20.0%	20.0%	0.01	1.69	0.03
ex7	10	1/1	1.01	5.9%	5.9%	0.0%	0.0%	0.01	1.94	0.02
bbara	10	0/0	0.31	0.0%	0.0%	3.3%	6.7%	0.01	1.89	0.01
train11	11	1/1	0.55	8.2%	8.2%	0.0%	0.0%	0.01	2.01	0.04
modulo12	12	2/2	0.58	14.3%	14.3%	*	*	0.02	2.44	0.06
mark1	12	1/1	0.95	3.1%	1.6%	-4.3%	-2.2%	0.02	2.24	0.04
ex4	14	0/0	0.59	0.0%	0.0%	7.7%	7.7%	0.01	2.96	0.05
dk512	15	1/1	1.60	15.3%	11.5%	7.4%	19.6%	0.03	3.11	0.06
s208	18	0/0	0.48	0.0%	0.0%	*	*	0.04	9.02	0.06
s1	20	8/1	1.25	7.2%	4.4%	3.8%	15.8%	0.04	10.15	0.02
ex1	20	3/2	0.98	12.5%	4.7%	0.8%	2.4%	0.03	10.01	0.09
donfile	24	6/3	1.52	10.2%	8.6%	-13.6%	-6.4%	0.07	20.25	0.28
pma	24	1/0	0.91	6.8%	0.0%	*	*	0.05	18.66	0.06
dk16	27	2/2	1.92	8.6%	3.5%	1.6%	9.2%	0.08	82.61	0.31
styr	30	2/2	0.53	18.4%	1.2%	1.7%	6.8%	0.05	49.32	0.21
s510	47	4/1	0.92	12.2%	4.1%	*	*	0.15	177.12	0.35
planet	48	8/1	1.53	11.8%	5.8%	10.8%	20.0%	0.15	197.63	0.37
s1488	48	5/0	0.35	0.2%	0.0%	*	*	0.17	215.45	0.19
Average	17.8	2.2/1.1		8.5%	5.7%	2.3%	6.6%	0.04	32.50	0.10

Comparison with the Optimal Encoding Solutions

To further demonstrate the advantage of FSM re-engineering as a performance enhancement technique, we apply FSM re-engineering to seven small FSM benchmarks, where the state encoding can be obtained optimally using the technique discussed in Section 4.5. This will also allow us to quantitatively judge the quality of POW3 by comparing with the optimal solutions.

Figure 8 depicts from bottom to top: the switching activity in the re-constructed FSMs (using the genetic algorithm) encoded by the optimal algorithm(OPT) and by POW3, and the switching activity in the original FSMs encoded by POW3. These numbers are all normalized to the switching activity in the original FSMs with the optimal encoding. We see that FSM re-engineering not only reduces the switching activity in the original optimally-encoded FSMs by 2.5% on average, it also enhances the effectiveness of POW3 greatly. From Figure 8, we see that the state encodings by POW3 in the original FSMs have switching activities up to 48.6% higher than the optimal encodings, with an average 27.0%. However, when we encode the re-constructed FSMs using POW3, this gap shrinks to 6.7%. In fact, in more than half of the benchmarks after FSM re-engineering, POW3 finds encoding schemes that have lower switching activity than the optimal ones in the original FSMs. Although these results show the power of re-engineering, we cannot assume that they can be safely extrapolated to larger FSMs (the experimental bottleneck being the computation of the exact switching activity).

Impact of Circuit Implementations

Table II reports the power, area, and delay of the circuits that implement the FSM benchmarks based on POW3 and our proposed FSM re-engineering approach. The data are from SIS. A † in the entry means that the benchmark degenerates into a single state after SIS optimization and no power, area or delay data are reported from SIS.

We see that an average 5.5% and 3.3% power reduction is achieved at the cost of 1.3% and 0.9% area increase and 1.3% and 0.8% delay increase in the circuits synthesized after FSM re-engineering using genetic algorithm and heuristic algorithm respectively. Note that in those FSMs modified by FSM re-engineering the average power reduction is 7.1%. However, on three benchmarks (namely *ex5*, *mark1*, and *planet*), we observe power increase after FSM re-engineering with genetic algorithm. This occurs when the power increase in the combinational part of a circuit (implied by the area increase) exceeds the power saving in the sequential part. The reason is that switching activity estimates the power in the sequential part, but it does not reflect the power in the combinational part of a circuit. For the same reason, we see that power reduction in Table II is less than the switching activity reduction reported in Table I, which is also true for other low power approaches based on switching activity such as [24]. With an accurate model that accounts both sequential and combinational logic

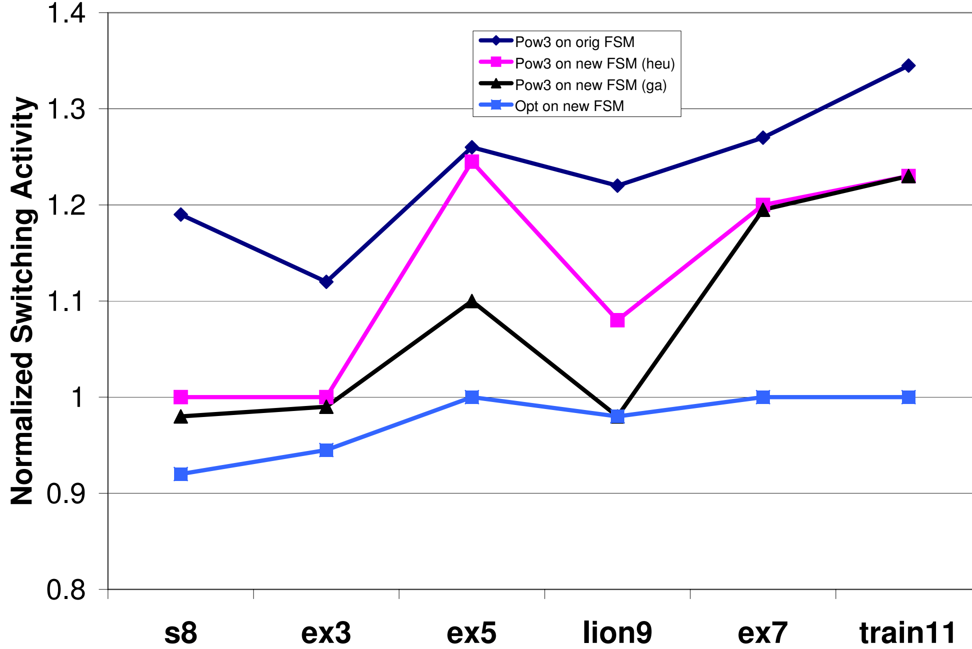


Fig. 8. Switching activities in (1) original FSMs encoded by POW3 and ILP-based optimal encoding (OPT) and (2) re-engineered FSMs encoded by POW3 and OPT; results are normalized to the switching activity in the original FSM encoded by OPT.

and a cost metric at the FSM level, we expect that the FSM re-engineering framework will be able to report always non-negative power reduction.

We have also compared our results in power reduction with the two low power non-minimal state encoding algorithms (fast and greedy) reported in [24]. Due to the lack of a common experimental setup, it is unrealistic to compare directly the absolute value of power consumption. Instead we adopt the practical method to compare the power reduction by different approaches with respect to the *POW3* algorithm. We obtained the power values from [24] and computed their reduction (in %) over POW3 for each benchmark FSM reported in [24]. Over all the 44 benchmarks, the average power reduction is 0.7% for the fast method and 3.2% for the greedy method. When restricted to the same set of benchmarks used in our paper, the average power reduction improves to 1.1% for the fast method and decreases to 1.9% for the greedy method, which are reported in the fifth and sixth column of Table II. An asterisk means that the power reduction for that benchmark is not available in [24]. Finally, on the 14 common benchmarks that are both included in our experiments and in [24], the average power reduction is 5.4% for our genetic algorithm based approach and 4.3% for our heuristic based approach.

B. Area driven optimization

We use JEDI [20], a well known area-driven state encoding algorithm, to encode the FSMs for area. In particular, we run JEDI with the fanout algorithm. Then we apply to the FSMs the re-engineering algorithm and finally we encode them by JEDI again. In the state splitting process, we consider the number of literals as the cost figure for area minimization in multi-level logic circuits. We mention that this is an approximation, since the effects of technology mapping and the cost of interconnections are not modeled; however, at FSM synthesis level, literal count is a very common and relatively effective way to estimate area.

Synthesis results of JEDI on FSMs before and after re-engineering are compared in Table III. The third column lists the number of states being split in FSM re-engineering; the fourth column shows the number of literals obtained by JEDI in the original FSMs and the fifth column shows the reduction in the number of literals after re-engineering. Only the heuristic algorithm is used in re-engineering because estimating the number of literals after each step of state splitting (see Figure 5) is computationally expensive, which makes it infeasible for the genetic algorithm.

FSM re-engineering splits state(s) in 16 benchmarks and reduces the number of their literals by 5.0% on average. In the remaining benchmarks, since no state is split, the literal counts remain the same, which accounts for an average reduction over all benchmarks of 3.5%. The reduction in the number of literals is smaller than that of switching activity in power-driven encoding. This is because the weight functions (obtained by fanout adjacency matrices) used in area minimization are not

TABLE II

POWER IN THE ORIGINAL CIRCUITS SYNTHESIZED BY POW3 (COLUMN 2) AND POWER REDUCTION BY FSM RE-ENGINEERING USING GENETIC ALGORITHM (COLUMN 3) AND HEURISTIC (COLUMN 4), COMPARED WITH POWER REDUCTION OVER POW3 BY FAST AND GREEDY STATE ENCODING ALGORITHMS IN [24] (COLUMN 5-6); AREA AND DELAY IN THE ORIGINAL CIRCUITS (COLUMN 7, 10) AND THEIR INCREASE IN CIRCUITS SYNTHESIZED WITH FSM RE-ENGINEERING (COLUMN 8, 9 AND 11, 12);'*' MEANS DATA NOT AVAILABLE IN [24] AND '†' MEANS DATA IS NOT AVAILABLE FROM SIS.

Circuit	Power (μW) reduction					Area (mm^2) increase			Delay (ns) increase		
	pow3	ga	heu	fast[24]	greedy[24]	pow3	ga	heu	pow3	ga	heu
s8	†	†	†	*	*	†	†	†	†	†	†
ex3	314	10.0%	6.8%	5.2%	7.0%	46400	8.0%	9.2%	14.68	3.1%	4.1%
s27	188.6	5.0%	-2.7%	*	*	37584	2.0%	0.8%	5.96	1.0%	0.6%
bbtas	107.2	0.0%	0.0%	4.1%	1.0%	35264	0.0%	0.0%	5.74	0.0%	0.0%
beecount	161.3	0.0%	0.0%	-8.1%	3.0%	33872	0.0%	0.0%	7.69	0.0%	0.0%
dk14	587.9	6.8%	0.0%	5.0%	7.0%	88160	2.1%	0.0%	22.16	5.2%	0.0%
ex5	405.2	-15.5%	-15.5%	0.0%	0.0%	70852	13.3%	13.3%	9.32	4.8%	4.8%
lion9	178.3	7.1%	10.9%	2.1%	-3.5%	38976	16.7%	18.5%	4.22	8.1%	10.2%
ex7	405.8	29.1%	29.1%	11.1%	13.6%	78416	-11.7%	-11.7%	13.28	-2.3%	-2.3%
bbara	350.3	0.0%	0.0%	5.8%	5.8%	60320	0.0%	0.0%	15.91	0.0%	0.0%
train11	212.3	19.0%	19.0%	12.3%	24.5%	47792	2.9%	2.9%	5.12	4.3%	4.3%
modulo12	†	†	†	*	*	†	†	†	†	†	†
mark1	280.7	-12.9%	-9.1%	-4.2%	-21.3%	94656	5.4%	3.7%	16.01	4.7%	3.9%
ex4	433.9	0.0%	0.0%	-31.7%	-21.3%	79101	0.0%	0.0%	13.69	0.0%	0.0%
dk512	430.1	5.1%	2.9%	3.3%	11.7%	79344	2.3%	1.1%	19.99	0.2%	0.0%
s208	630.3	0.0%	0.0%	*	*	101781	0.0%	0.0%	14.22	0.0%	0.0%
s1	1388.7	12.9%	10.8%	*	*	321088	-2.3%	-1.9%	24.34	-1.7%	-0.2%
ex1	744.9	13.6%	4.6%	6.0%	-10.0%	234784	-8.9%	-6.5%	36.77	-3.0%	-2.2%
donfile	†	†	†	*	*	†	†	†	†	†	†
pma	987.2	7.3%	0.0%	*	*	180200	6.3%	0.0%	43.88	4.4%	0.0%
dk16	1547.3	13.3%	11.5%	4.8%	9.6%	282122	-9.7%	-9.7%	53.21	-5.7%	-5.7%
styr	1347.6	10.0%	1.1%	*	*	407856	3.3%	2.9%	41.73	0.9%	0.6%
s510	923.1	13.4%	5.7%	*	*	302046	-6.3%	-4.4%	33.57	-1.8%	-1.1%
planet	2042.1	-7.9%	3.1%	*	*	504832	1.8%	0.6%	37.72	1.7%	0.5%
s1488	7516.3	4.6%	0.0%	*	*	949344	3.9%	0.0%	48.33	5.3%	0.0%
Average		5.5%	3.3%	1.1%	1.9%		1.3%	0.9%		1.3%	0.8%

TABLE III

NUMBER OF LITERALS IN THE ORIGINAL FSMS ENCODED BY JEDI (COLUMN 4) AND ITS REDUCTION IN THE RE-ENGINEERED FSMS (COLUMN 5); AREA OF THE CIRCUITS SYNTHESIZED BY JEDI (COLUMN 6) AND ITS REDUCTION IN THE RE-ENGINEERED FSMS (COLUMN 7); POWER AND DELAY IN CIRCUITS SYNTHESIZED BY JEDI (COLUMN 8, 10) AND THEIR INCREASE IN THE RE-ENGINEERED FSMS (COLUMN 9, 11); THE RUN-TIMES OF JEDI AND FSM RE-ENGINEERING (COLUMN 12-13); '†' MEANS DATA IS NOT AVAILABLE FROM SIS.

FSM	# states	# splits	# literals		Area (mm^2)		Power (μW)		Delay (ns)		Run-time (s)	
			JEDI	re-eng	JEDI	re-eng	JEDI	re-eng	JEDI	re-eng	JEDI	re-eng
s8	5	0	†	†	†	†	†	†	†	†	†	†
ex3	5	1	85	9.0%	37120	2.0%	403.9	2.0%	13.79	-2.0%	0.01	0.01
s27	6	1	63	11.0%	29232	7.0%	136.2	-2.0%	5.66	-5.0%	0.02	0.01
bbtas	6	1	34	2.9%	35264	0.1%	122.1	12.9%	9.45	-0.5%	0.02	~
beecount	7	0	29	0.0%	26912	0.0%	144.7	0.0%	6.73	0.0%	0.02	0.01
dk14	7	1	362	4.0%	76560	2.0%	545.9	-1.0%	25.87	-1.0%	0.04	0.03
ex5	9	1	89	12.0%	65424	8.0%	399.1	-5.0%	9.22	-2.0%	0.06	0.07
lion9	9	1	23	-4.0%	32944	-1.0%	149.6	-3.0%	4.03	3.0%	0.05	0.03
ex7	10	1	111	6.0%	67744	7.0%	413.1	10.0%	6.73	-2.8%	0.16	0.08
bbara	10	0	126	0.0%	70852	0.0%	176.4	0.0%	13.63	0.0%	0.1	0.04
train11	11	1	40	-5.0%	43152	-14.0%	215.6	6.0%	4.69	7.0%	0.14	0.08
modulo12	12	0	†	†	†	†	†	†	†	†	†	†
mark1	12	1	269	10.0%	89088	5.0%	438.9	1.0%	15.03	-1.0%	0.27	0.11
ex4	14	2	114	6.0%	78416	13.0%	460.4	-15.0%	13.57	-4.0%	0.38	0.13
dk512	15	0	99	0.0%	68672	0.0%	493.9	0.0%	17.02	0.0%	0.88	0.02
s208	18	1	187	9.0%	101616	20.0%	633.9	-6.0%	14.16	-14.0%	1.38	0.25
s1	20	1	1047	3.2%	168432	8.1%	753.2	5.0%	19.75	-5.0%	1.17	0.53
ex1	20	1	1132	0.0%	208836	0.0%	916.2	0.0%	35.03	0.0%	1.75	0.76
donfile	24	0	†	†	†	†	†	†	†	†	†	†
pma	24	1	567	2.9%	178640	-9.1%	1000.2	0.1%	43.23	-9.0%	1.05	1.44
dk16	27	1	626	5.0%	227824	7.3%	1414.6	8.0%	48.88	1.3%	1.5	1.38
styr	30	0	1855	0.0%	750752	0.0%	4687.1	0.0%	33.55	0.0%	2.44	1.1
s510	47	0	1009	0.0%	276544	0.0%	2248.2	0.0%	32.46	0.0%	4.37	0.99
planet	48	1	2676	4.2%	949344	4.0%	7971.4	-5.0%	48.33	-2.0%	10.55	22.41
s1488	48	0	1979	0.0%	841332	0.0%	4997.1	0.0%	34.23	0.0%	6.93	1.01
Average				3.5%		2.7%		0.4%		-1.8%	1.51	1.45

as accurate in estimating the number of literals as the state transition probabilities in estimating the switching activities in power-driven synthesis. We also noticed that the number of states being split never exceeds two and when the original number of states is a power of 2, the re-engineering algorithm always chooses not to split any state. This implies (1) state splitting for area minimization is not as effective as it is for power; (2) increasing the number of state bits rarely helps to reduce area.

After synthesis, we map the FSMs to sequential circuits in SIS. The sixth column shows the area in the circuits implemented from the original FSMs; the seventh column shows the area reduction in the circuits implemented from the re-engineered FSMs. We see that 11 circuits had an area reduction after FSM re-engineering averaging 2.7%. In benchmark *ex1*, one state was split with no area reduction (or a reduction smaller than 0.004%). We also notice that a reduction in the number of literals results in area increase in benchmark *pma*. This is caused by the discrepancy between the real area and the approximation using the number of literals. The effectiveness of FSM re-engineering highly depends on the accuracy of the indicator for the optimization objective, because the FSM re-engineering algorithm in phase II needs to analyze the quality of solution based on the cost model to decide how to re-construct the FSM.

The next four columns show the power and delay in the original FSMs by JEDI and their increase in the re-engineered FSMs. We see that delay always decreases as the circuit's area decreases; power is also reduced (negative figures in the table) whenever there is a large reduction in area, because fewer gates are dissipating power in the combinational part.

The last two columns in Table III report the run times of JEDI and of the heuristic FSM re-engineering algorithm.

VI. 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 FSMs may lose the optimal solutions, or make harder to find them, for many FSM related optimization problems. To keep the discussion concrete, we propose a state splitting based FSM re-engineering technique for power-driven synthesis and area-driven synthesis, respectively. In order to demonstrate its strength in enhancing the performance of any given state encoding algorithms, we apply this technique on MCNC benchmarks using POW3 as the power-driven state encoding algorithm, and JEDI as the area-driven state encoding algorithm. Experimental results show that POW3's effectiveness in reducing circuit's total switching activity has almost been doubled by the proposed FSM re-engineering approach; on the other hand, the performance of JEDI in reducing circuit's area has been improved by up to 20%. Moreover, the FSM re-engineering framework has a small run-time overhead and can be applied to most of the FSM encoding tools or algorithms.

References

[1] D.B. Armstrong, "A Programmed Algorithm for Assigning Internal Codes to Sequential Machines," *IRE Transaction on Electronic Computers*, pp. 466-472, 1962.

[2] 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.

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

[4] 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.

[5] 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.

[6] X.Du, G.D. Hachtel, B.Lin, and A.R.Newton, "MUSE: a MUltilevel Symbolic Encoding Algorithm for State Assignment," *IEEE Transaction on Computer-Aided Design,* pp. 28-38, 1991.

[7] R.M. Fuhrer and S.M. Nowick, "OPTIMIST: State Minimization for Optimal 2-Level Logic Implementation," *Proceedings of International Conference of Computer-Aided Design,* pp. 308-315, Nov. 1997.

[8] 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.

[9] 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.

[10] 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.

[11] J. Hartmanis and R.E. Stearns, "Some Dangers in State Reduction of Sequential Machines", *Information and Control,* pp252-260, Sept, 1962.

[12] B. Holmer, "What are the Ingredients for a Good State Assignment Program?", *Technical Report No. CSE-95-002,* EECS Department, Northwestern University, April 1995.

[13] 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.

[14] L. Jozwiak, "Efficient Suboptimal State Assignment of Large Sequential Machines", *Proceedings of Euro DAC*, pp. 536-541, 1990.

[15] T. Kam, T. Villa, R. Brayton and A. Sangiovanni-Vincentelli, "Synthesis of FSMs: Functional Optimization", *Kluwer Academic Publishers,* 1997.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] S. Malik, L.Lavagno, R.Brayton, and A.Sangiovanni-Vincentelli, "Symbolic Minimization of Multilevel Logic and the Input Encoding Problem," *IEEE Transaction on Computer-Aided Design*, pp. 825-843, 1992.

[22] G. De 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 .

[23] G. De 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] J.H. Tracey, “Internal State Assignments for Asynchronous Sequential Machines”, *IEEE Transactions on Electronic Computers*, EC-15:551 560, August 1966.

[30] 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.

[31] 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.

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

[33] T. Villa, T. Kam, R. Brayton and A. Sangiovanni-Vincentelli, “Synthesis of FSMs: Logic Optimization”, *Kluwer Academic Publishers,* 1997.

[34] T. Villa, A. Saldanha, R. Brayton and A. Sangiovanni-Vincentelli, “Symbolic Two-Level Minimization,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* pp. 692-708, Vol. 16, N. 7, July 1997.

[35] C. Umans, T. Villa and A. Sangiovanni-Vincentelli, “Complexity of Two-Level Logic Minimization,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* pp. 1230-1246, Vol. 25, N. 7, July 2006.

[36] 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.