Cycle-based Decomposition of Markov Chains with Applications to Low Power Synthesis and Sequence Compaction for Finite State Machines

Ali Iranli and Massoud Pedram
Dept. of Electrical Engineering
University of Southern California
Los Angeles CA 90089

Abstract *- This paper advances the state-of-the-art by presenting a well-founded mathematical framework for modeling and manipulating Markov processes. The key idea is based on the fact that a Markov process can be decomposed into a collection of directed cycles with positive weights, which are proportional to the probability of the cycle traversals in a random walk. Two applications of this new formalism in the computer aided design area are studied. In the first application, we present a new state assignment technique to reduce dynamic power consumption in finite state machines. The technique comprises of first decomposing the state machine into a set of cycles and then performing a state assignment by using Gray codes. The proposed encoding algorithm reduces power consumption by an average of 15%. The second application is sequence compaction for improving the efficiency of dynamic power simulators. The proposed method is based on the cycle decomposition of the Markov process representing the given input sequence and then selecting a subset of these cycles to construct the compacted sequence.*

I. Introduction

Mathematical modeling is an essential task in the early steps of system design. Markov processes are one of most widely used models in describing the behavior of VLSI systems. These models are for example used to define the behavior and performance of history dependent systems, analyze different memory dependent characteristics of circuits such as power dissipation and latency, and design policies for system-level resource management including dynamic power management (DPM.) In addition, there are many system design optimization problems which strongly depend on information about the near past and/or future of the system workload and behavior e.g., dynamic resource binding and on-the-fly reconfiguration. These problems can be readily modeled by the Markov processes.

Traditionally, a Markov process has been represented by a set of *states* and a set of *transitions* (edges) among these states. This representation is simple, compact, and easy to understand. Furthermore, using this representation, one can easily acquire information about the past (fan-in states) and/or the near future (fan-out states) with respect to current state of the system. However, there are another set of design problems that require information about the past and/or future trajectory of system states. One such problem is the low power state assignment problem for VLSI circuits. In these types of problems, the customary representation of Markov processes in terms of states and transitions fails to efficiently represent the historic and/or futuristic trajectory of the system states. This paper introduces a mathematical framework for cycle representation of Markov processes. In this representation, a Markov process is represented by using a set of directed *cycles* and their corresponding *weights*. This representation translates the state trajectory of the system into a set of intra-cycle and inter-cycles transitions. This representation of a Markov process in terms of cycles and weights captures more information about the trajectory of the system states and can be used to compactly formulate and solve an important set of design optimization problems.

More precisely, in this paper, the theory of cycle representation of a Markov process is used to prove that 1) A given Markov process is probabilistically equivalent to a set of weighted cycles; 2) For a given set of cycles generated according to some ordering, there is only one set of cycle weights which have probabilistic interpretation in terms of the original Markov process. In other words, if one is not careful when generating a cycle decomposition of the Markov chain, he/she may generate a decomposition which is incorrect in the sense that there may exist some edge in the Markov chain whose weight is not equal to the summation of weights of edge-covering cycles that are in the extracted cycle set and cover that edge.

The key theoretical contribution of this paper is to show how to cost-consciously decompose a Markov process into a collection of directed cycles with positive weights that are proportional to the probability of their traversal in a typical random walk. We solve two versions of this problem. In the first version, detailed knowledge about the Markov chain itself, i.e., the state diagram, state transition probabilities and the steady-state state probabilities, exists. This problem is solved optimally by a deterministic cost-driven cycle-decomposition technique. The second version of the problem assumes no prior knowledge about the state diagram of the Markov process. This problem is solved heuristically using a probabilistic cost-driven cycle-decomposition technique.

Based on this mathematical framework, we present solutions for two “power-related” problems, 1) State assignment and, 2) Sequence compaction. To solve the low power state assignment problem, we identify most probable cycles in the FSM and encode the states on these cycles with Gray codes. The objective function is to minimize the Weighted Hamming Distance. Notice that although techniques such as those presented by Tsui and Pedram in [7] use more accurate cost functions for two and multi-level logic realization of the state machines, the

MWHD metric is still relevant and valuable. In other words, although MWHD does not exhibit high absolute accuracy, it has good relative accuracy as demonstrated in published results [5][6]. [1] In particular, during the signoff analysis, we are interested in an evaluation metric with high accuracy whereas during synthesis and optimization we are typically interested in a power or delay equation with high relative accuracy. As for the sequence compaction problem, first, the input vector sequence is modeled as a Markov chain, it is then decomposed into a set of cycles. The average power dissipation of the circuit under the applied vector sequence is calculated by simulating each decomposed cycle of the Markov process once and then calculating the average over power consumption of all cycles in the input Markov model. The mathematical framework for the proposed sequence compaction technique is different from those of [13]-[18] and results in higher compaction ratios for the same level of evaluation accuracy. The proposed technique is also clearly different from statistical techniques, which attempt to achieve the same objective through sampling strategies [19][20].

I.A Prior Work Review

State encoding/assignment, as a crucial step in the synthesis of the controller circuitry, has been extensively studied. Early research on state assignment was focused on finding a state encoding that minimizes the circuit area [2]-[4]. In the 1990's, a number of low power state-encoding techniques were presented [5]-[7]. Roy et al. was the first to address the problem of reducing switching activity of input state lines of the next state logic, during the state assignment, formulating it as a Minimum Weighted Hamming Distance (MWHD) problem [5]. Olson et al. used a linear combination of switching activity of the next state lines and the number of literals as the cost function [6]. Tsui et al. [7] used simulated annealing as a search strategy to find a low power state encoding that accounts for both the switching activity of the next state lines and switched capacitance of the next state and output logic.

On the other hand, for power estimation it is well known that the average power dissipation in a CMOS circuit is proportional to a summation over all gates of the product of the capacitive load and the switching activity of each gate [1]. This summation, which is often called the switched capacitance of the circuit, can be calculated directly by simulating the circuit. To produce a power estimate with good accuracy and high confidence, the direct simulation technique requires explicit, weighted enumeration of all allowed pairs of input vectors, which is impractical for any circuit of reasonable size and complexity. Therefore, adopting a computationally less expensive method for estimating the average power dissipation in a CMOS VLSI circuit is crucial. There exist two main classes of *Static* and *Dynamic* techniques for power estimation [9].

Static techniques are based on calculating the probabilistic behavior of internal nodes of the circuit as a function of the stochastic behavior of the input vectors (e.g., switching activity, spatio-temporal correlations). Examples of power estimation techniques in this category are [10]-[12]. These techniques generally provide sufficient accuracy with low computational overhead, however, important effects such as signal slew rates, generation and propagation of hazards/glitches cannot be properly captured by these techniques. In addition, probabilistic power estimation techniques tend to have difficulty in efficiently capturing the complete set of spatio-

[1] Absolute accuracy is a measure of the estimate of some parameter of interest compared to its actual value. In contrast, relative accuracy is a measure of the accuracy of individual estimates of the parameter when compared to other estimates of the same parameter made under different conditions. For example, a proposed power estimation metric may produce absolute errors of 30% or higher for some circuit whereas that metric may exhibit a relative error of at most 10% between estimates for two different implementations of that same circuit.

temporal correlations at the external inputs and the reconvergent fan-out structures in the circuit, which further reduces their estimation accuracy.

Dynamic techniques explicitly simulate the circuit under a "typical" input stream. They can be applied at both the circuit and gate-level. Their main shortcoming is, however, that they are very slow. Moreover, their results are highly dependent on the simulated sequence. A number of issues appear to be important for power estimation using dynamic techniques. The input statistics, which must be properly captured, and the length of the input sequences, which must be applied, are two such issues. Generating a minimal-length sequence of input vectors that satisfies these statistics is not trivial. The reason is the elaborate set of input statistics, which must be preserved or reproduced during sequence generation for use by power simulators.

I.B Paper Organization

The remainder of this paper is organized as follows. Section II provides the theoretical background for the cycle-decomposition of Markov processes. Sections III and IV present the application of this mathematical framework to two instances of computer aided design problems, i.e., low power state encoding and sequence compaction. Conclusions and summary are given in Section V.

II. Cycle Decomposition of a Markov Process (CDMP)

Let us start with a simple example of a cyclic process modeling the motion of a particle on a closed curve. Let's focus on the particle's motion through p points of this curve at moments that are one unit of time apart; cf. Figure 1a. This leads us to a discretization of the curve into an infinite sequence of points $C = (v_1, v_2,..., v_p, v_{p+1}=v_1, v_{p+2}=v_2,...)$ called a directed cycle with period p. If no disturbance occurs, the passing of the particle through (v_i, v_{i+1}) can be codified by an infinite binary sequence,

$$y_{v_i,v_{i+1}} = \ldots 0\underbrace{100\ldots0}_{p}100\ldots010\ldots \tag{1}$$

where 1 (0) means that the particle is (is not) passing through (v_i, v_{i+1}). The sequence is understood as a *non-random* sequence in the context of *Kolmogorov's theory* of complexity since both 1 and 0 appear periodically after every p steps [21].

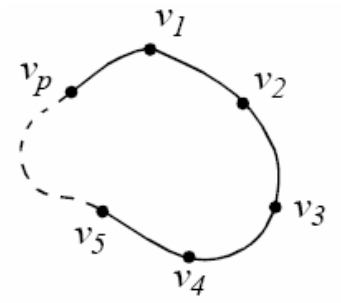
Consider a set of possibly overlapping cycles $\{C_1,...,C_r\}$ where each cycle C_i $(i \le r)$ is associated with some positive number $W(C_i)$; (cf. Figure 1b). Imagine that at some instance of time, the particle appears at some point v that is common to t cycles, say $C_1,...,C_t$ $(t \le r)$. The particle may continue its way to another point v', which is the intersection point of m cycles (out of t cycles that had point v as an intersection point) say $C_1,...,C_m$ $(m \le t)$. A natural measure of the particle's transition when moving from v to v' can then be defined as:

$$\frac{W(C_1)+W(C_2)+\cdots+W(C_m)}{W(C_1)+W(C_2)+\cdots+W(C_t)} \tag{2}$$

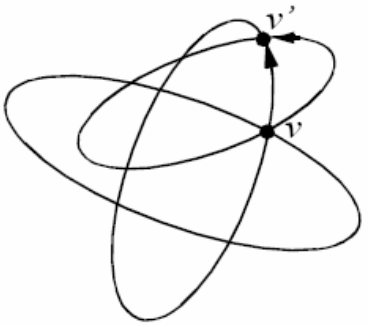
Accordingly, the binary sequence $y_{v,v'}$ codifying the transition of the particle from v to v' is given by a *chaotic* sequence. Furthermore, since expression (2) provides transition probability from v to v' of a Markov process ξ that behaviorally models the particle's movement, it can be concluded that: *"A collection of cycles, $\mathbb{C}$, along with some weights assigned to each cycle, defines a Markov process ξ."*

II.A CDMP: Problem Statement

Motivated by the above example, we proceed with a formal definition of directed cycles and cycle-based representation of Markov processes.



a. Particle's motion on a closed curve



b. Collection of overlapping cycles

Figure 1. Super imposing cycles

Definition 1. A directed cycle over a countable set of states S is a periodic function *C* from the set $\mathbb{Z}$ of integers into *S*. Furthermore *C(i)* is called a vertex of the cycle and (*C(i), C(i+1)*) is called a directed edge of the cycle.

Each cycle *C* belongs to an equivalence class of cycles $\mathbb{C}$ where $\mathbb{C}=\{C' \mid \forall i \in \mathbb{Z},\ C'(i)=C(t(i))\}$, where *t* is a translation function over $\mathbb{Z}$. Two cycles belonging to the same equivalence class are called *equivalent*.

Definition 2. For a cycle *C*, passage function J_C is a binary function defined over the set of states *S* as follows:

$$J_C(v)=\begin{cases}1 & \exists i \in \mathbb{Z}, C(i)=v \\ 0 & \text{otherwise}\end{cases} \tag{3-a}$$

The second order passage function can be defined as:

$$J_C(v,v')=\begin{cases}1 & \exists i \in \mathbb{Z}, C(i)=v \text{ and } C(i+1)=v' \\ 0 & \text{otherwise}\end{cases} \tag{3-b}$$

Definition 3. Let *S* be a countable set of states. Sequence $\xi=(X_n)\in S,\ n\geq 0$ of random variables on the probability space Ω is said to be a homogeneous Markov process with state space *S* if for any n≥0 and $Y_0, Y_1, \ldots, Y_{n+1}\in S$, we have

$$P(X_{n+1}=Y_{n+1} \mid X_n=Y_n,\ X_{n-1}=Y_{n-1},\ \ldots,\ X_0=Y_0)=P(X_{n+1}=Y_{n+1} \mid X_n=Y_n) \tag{4-a}$$

Moreover, the Markov process ξ is called recurrent exactly if

$$\exists n\geq 1:\ p_{ii}^{(n)}>0 \quad ; \quad \forall i \tag{4-b}$$

where $p_{ij}=P(X_{n+1}=j \mid X_n=i)$ and $p_{ij}^{(n)}$ are the single and n-step transition probabilities from state *i* to state *j*.

Based on aforementioned definitions, Kalpazidou proposes the following general theorem for cycle decomposition of Markov processes (cf. [21] pages 130-140.)

Theorem 1. Let *S* be a finite set of states. Consider a homogeneous recurrent |*S*|-state Markov process ξ defined over a probability space with common invariant probability distribution

p_i, $i \le |S|$; then there exists a finite set of weighted cycles $\mathbb{C}$ such that superposing the cycles will define ξ; i.e.,

$$p_i = \sum_{C \in \mathbb{C}} W(C).J_C(v_i)$$
$$p_{ij} = \frac{\sum_{C \in \mathbb{C}} W(C).J_C(v_i, v_j)}{p_i} \qquad (5)$$

where $W(C)$ is positive weight assigned to cycle C. p_i, and p_{ij} are steady state probability of state v_i and conditional probability of transition from v_i to v_j, respectively.

Proof. Given in reference [21]. ♦

Based on Theorem 1, the Markov process decomposition problem can be stated as follows.

Cycle Decomposition of a Markov Process (CDMP) Problem: Given a homogeneous recurrent $|S|$-state Markov process, find a set of weighted cycles $\mathbb{C}$ and their weights $W(C_i)$, such that their superposition defines the Markov process.

Solutions to the CDMP problem can be classified as probabilistic or deterministic solutions depending on whether or not the weights on cycles are subjected to probabilistic interpretation or not. In the following sections, one solution from each class will be presented.

II.B A Probabilistic Solution to the CDMP Problem

In [24], Qian and Qian presented a probabilistic approach for performing cycle decomposition. This approach is based on the fact that if we take an infinitely long random walk on the states of the Markov process, all possible cycles are identified. The weights for these cycles are then calculated by solving the set of equations in (5). An infinitely long random walk starting from a random state of the Markov process is initiated. The algorithm *generate_cycles*, shown in Figure 2, is then used to find all the cycles along the generated random walk. Given the walk V on the states of Markov process ξ, all the states are traversed one by one. If a previously visited state is encountered, a cycle is identified (cf. line 4). The sequence of states on the walk that makes up the cycle C are then replaced by the single state, s. State s is the boundary state of cycle C, which means that cycle C is started and returned to state s along the given walk V of the Markov process ξ. The algorithm continues until all cycles are identified.

```
Algorithm Prob_CD (V)
begin
1.  ℂ=Φ;
2.  foreach state s on V do
3.      if s is already visited then
4.          C = states on V between the two occurrences of s;
5.          ℂ=ℂ ∪ {C};
6.          Replace set of states C on V by the single state s;
7.      end if
8.  end for
9.  Solve set of linear equations (5) for set ℂ of cycles;
10. return ℂ;
end
```

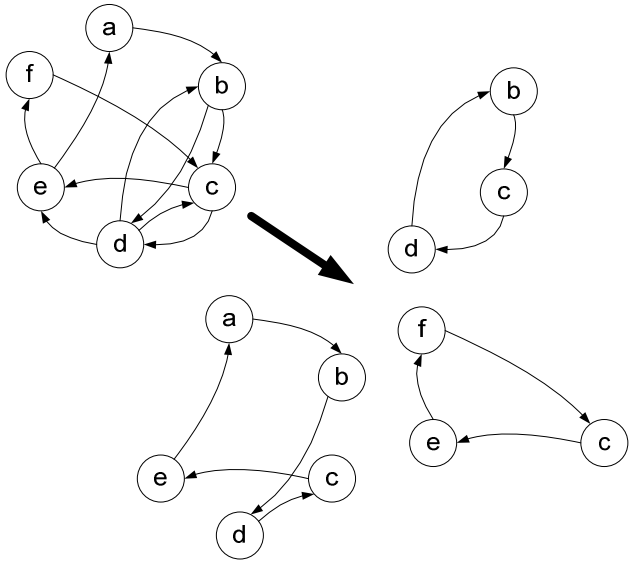
Figure 2. Probabilistic cycle decomposition algorithm

For example, consider a trajectory of ξ given by (a, b, c, d, b, d, c, e, f, c, e, a, …), applying the *generate_cycle* algorithm extracts a cycle set as shown in Figure 3. When all cycles are extracted, the set of linear equations (5) are solved by using any linear programming approach such as Gauss-Seidel method (cf. [25]) to find the corresponding weights for each cycle.

n	1	2	3	4	5	6	7	8
ξ_n	a	b	c	d	b	d	c	e
V	a	a,b	a,b,c	a,b,c,d	a,b,c,d,b	a,b,d	a,b,d,c	a,b,d,c,e
C					[b,c,d,b]			
$\mathbb{C}$	ϕ	ϕ	ϕ	ϕ	{[b,c,d,b]}	{[b,c,d,b]}	{[b,c,d,b]}	{[b,c,d,b]}

n	8	9	10	11	12
ξ_n	e	f	c	e	a
V	a,b,d,c,e	a,b,d,c,e,b,f	a,b,d,c,e,f,c	a,b,d,c,e	a,b,d,c,e,a
C			[c,e,f,c]		[a,b,d,c,e,a]
$\mathbb{C}$	{[b,c,d,b]}	{[b,c,d,b]}	{[b,c,d,b], [c,e,f,c]}	{[b,c,d,b], [c,e,f,c]}	{[b,c,d,b], [c,e,f,c], [a,b,d,c,e,a]}

(a)



(b)

Figure 3. Cycle extraction using *generate_cycle* algorithm,
a) cycle decomposition steps b) the Markov Model and its cycles

It is obvious that this approach to cycle decomposition requires non-polynomial time to find all possible cycles of a given Markov process, since the number of simple cycles is combinatorial in a number of states in the Markov process. It is worth noting that cycle weights found using this

approach are unique and independent of the ordering in which the cycles were generated. The weights have a probabilistic interpretation, in the sense that W(C) is equal to the expected number of times that C appears along an infinitely long sample path. In the next section, we will propose a deterministic approach to cycle generation with polynomial time complexity in a number of states in the Markov process.

II.C Deterministic Solutions to the CDMP Problem

In [21], Kalpazidou presented a deterministic approach to the CDMP problem as follows. Consider a homogenous and recurrent Markov process ξ. Pick an arbitrary state v_i. Since the process is recurrent, there exists at least one state v_j, such that the transition probability from *i* to *j*, P_{ij}, is non-zero. Pick this new state v_j and repeat the procedure. Since the number of states is finite, a cycle will finally be created. Set the weight for this cycle to the minimum probability of any transition on the cycle, and decrease the probability of each transition on the cycle by this weight. Proceed in a similar manner to process other cycles until no nonzero probability transition is left.

Theorem 2: Let ξ be a homogenous and recurrent Markov process and $\mathbb{C}$ be the set of weighted cycles generated using the above-mentioned deterministic approach. The set of cycles in $\mathbb{C}$ is a decomposition of ξ that satisfies Theorem 1.

Proof: Following the cycle generation procedure mentioned above, in each iteration, after finding a cycle, the total transition probability of the edges on that cycle is reduced such that the total transition probability of the minimum-weight edge becomes zero. Now, if the minimum-weight edge happens to be edge ij, then $p_i.p_{ij} - w(ij) = 0$, and *ij* will be removed from the Markov chain. However, more likely, the minimum-weight edge is some other edge, say *kl*, and thus, the weight of edge *ij* after extracting this cycle will be $p_i.p_{ij} - w(kl) > 0$. After extracting a number of cycles, the weight of target edge *ij* is finally reduced to zero. In this way, the total transition probability of each edge is distributed between different cycles that are extracted in each iteration, and therefore, we have,

$$p_i.p_{ij} = \sum_{\forall C:(v_i,v_j)\in C} W(C)$$

Now,

$$p_i.p_{ij} = \sum_{\forall C:(v_i,v_j)\in C} W(C) = \sum_{\forall C} W(C).J_C(v_i,v_j)$$

$$p_i = \sum_{\forall v_j} \sum_{\forall C:(v_i,v_j)\in C} W(C) = \sum_{\forall C:v_i\in C} W(C)$$

$$= \sum_{\forall C} W(C).J_C(v_i)$$

♦

Notice that the cycles generated by the deterministic approach are not unique and depend on the policy for selecting the next state. Moreover, even if the same set of cycles is generated, the weight for each cycle will depend on the order in which the cycles were generated.

Theorem 3: Let ξ be a homogenous and recurrent Markov process and $\mathbb{C}$ be the set of weighted cycles generated using the above-mentioned deterministic approach, then $|\mathbb{C}|$ is of $O(|S|^2)$ where $|S|$ is the number of states in ξ.

Proof: Because in each iteration of the deterministic approach, after extracting a cycle, weight of at least one edge becomes zero, $|\mathbb{C}|$ is upper bounded by the number of edges in the Markov process which is $O(|S|^2)$, Thus, the algorithm can have $O(|S|^2)$ iterations in the worst case. ♦

To summarize, the cycle decomposition of Markov processes can be done in two ways: 1) a probabilistic approach that requires the enumeration of all simple cycles in the given Markov process. This enumeration requires exponential time in terms of the size of Markov process, and results in cycle weights that are equal to the probability of visiting the cycles in any random walk along the given Markov process. 2) a deterministic approach that produces a polynomial size set of cycles in polynomial time in terms of the size of Markov process. In this approach, cycle weights no longer carry the probabilistic meaning, but satisfy the requirements of theorem 1.

This deterministic solution is a generic procedure without any cost function to optimize. To utilize the cycle representation of FSMs for design optimization, it is important to develop a cost-aware deterministic cycle decomposition algorithm. More precisely, we must augment Kalpazidou's algorithm to select the "right" set of cycles for the target application. One such specialization is presented in Figure 4.

```
Algorithm Dtr_CD (ψ)
begin
1.  ℂ = ∅;
2.  while max. edge weight > 0 do
3.      pick (v_i, v_j) w/ max. weight;
4.      ω = ∅;
5.      repeat
6.          ω = ω + (v_i, v_j);
7.          pick (v_i, v_j) w/ max. weight in (E - ω);
8.      until there is a cycle in ω;
9.      C = cycle_of (ω);
10.     W(C)=min. {w_ij | J_C(v_i,v_j)=1};
11.     ℂ = ℂ ∪ {C};
12.     foreach i,j: J_C(v_i,v_j)=1 do
13.         w_ij = w_ij – W(C);
14.     end for
15. end while
16. return ℂ;
end
```

Figure 4. Deterministic cycle generation algorithm

In this algorithm, the weight of each edge is the total transition probability of that edge; i.e., $w_{ij} = p_i.p_{ij}$ $\forall i, j=1...|S|$. The algorithm starts with the edge with maximum weight in line 3 and then repeatedly picks the maximum weight edges from the set of remaining edges (line 7). This process is repeated until a cycle is generated (line 8); the cycle weight will then be set to the minimum of all the edge weights on the cycle (line 10), and the weights of all the edges on the

cycle will be decreased by the cycle weight (lines 12-14). This process is repeated until there are no more edges with non-zero weights.

When a cycle is formed by successively marking maximum-weight edges as per *Dtr_CD* algorithm, the cycle does not necessarily include all of the marked edges. Consider, for example, the following sequence of max-weight edges in a 5-vertex graph: 12, 34, 15, 35, and 25. The cycle formed here for the first time, is 1251. Clearly all edges are max-weight edges; yet only three of the five marked edges are included in the cycle; Assume that the min-weight edge is 12; Subsequently, the weight of this edge becomes zero; the weights of edges 15 and 25 are reduced by w(12) while the weight of edges 35 and 34 remain unchanged. The reason we choose edges of maximum weight may be explained as follows. During the state encoding step, the extracted cycles are sorted according to their weights. Next the cycle of maximum weight is identified and encoded; The states in this first cycle will therefore have the maximum flexibility in assuming any Gray code sequence, which means that the expected switching activity for transitions along the cycle (which have the highest occurrence probability) are minimized. This corresponds to the minimization of the MWHD during state assignment.

III. Low Power State Assignment

Finite state machines (FSM's) can be described by a six-tuple $\psi(X, Y, S, s_0, \lambda, \eta)$, where X is the set of input symbols, Y is the set of output symbols, S is the set of states, s_0 is the initial state, $\lambda: X \times S \rightarrow Y$ is the output function, and $\eta: X \times S \rightarrow S$ is the next state function.

From a probabilistic point of view, an FSM, ψ, can be described by a Markov chain in which p_i is the probability of being in state s_i and p_{ij} is the conditional probability of transition from state v_i to state v_j. The p_{ij} values are obtained from input sequence statistics and the next state function of the FSM, or they can be calculated based on output trace for simulation of typical input sequences to the FSM. The p_i values are in turn calculated by solving the Chapman-Kolmogrov equations [22][23].

The state transition graph of an FSM is a vertex/edge weighted, directed graph $G(V, E)$, where the set of vertices V and set of edges E correspond to the states of the FSM and transitions between them respectively:

$$\begin{aligned} V &= \{v_i \,|\, v_i \in S\} \\ E &= \{(v_i, v_j) \,|\, \exists x \in X : \eta(v_i, x) = v_j\} \end{aligned} \tag{6}$$

The weight assigned to each vertex v_i is the state probability p_i, and the weight assigned to each edge (v_i, v_j) is the probability p_{ij} in the Markov process. Assume states v_i and v_j are encoded using binary strings b_i and b_j, respectively. The transition from state s_i to state s_j will have a switching activity equal to d_{ij}, the hamming distance between b_i and b_j. Since dynamic power consumption is directly related to switching activity and state transition in an FSM corresponds to the switching of the state bits, state encoding will have a major effect on the power consumption. The goal is to perform the state assignment in such a way that state transitions, with higher probability, take place within a smaller switching activity on state bits. The objective function to minimize would then be:

$$p_{ave} = \sum_i p_i . \sum_{j:(v_i, v_j) \in E} p_{ij} . d_{ij} \tag{7}$$

Implementation of an FSM is usually done using D flip-flops. The input to the flip-flops would then be $D = \eta(x, v)$ where x is the input and v is the present state.

In our proposed technique for low power state encoding, the FSM is first decomposed into a set of cycles. These cycles are then encoded in order to minimize the total switching according to

the cost function in (7). Once the cycles are encoded, the entire FSM will be implemented using D flip-flops.

III.A Cycle-decomposition-based Encoding

Having described a new mathematical framework for Markov process cycle decomposition, we can now proceed with the solution to the low-power state encoding problem. As mentioned earlier, the technique for state assignment proposed here is based on the decomposition of FSM into a set of cycles. Due to the high complexity of the probabilistic method (i.e., the number of cycles in that approach can grow exponentially in the number of states of the FSM), the deterministic method will be employed here to generate the cycles.

After decomposing the FSM into a set of cycles, the *CD_based_encoding* algorithm of Figure 5 is employed for the state assignment step.

Algorithm CD_based_encoding (ψ)
// ψ is the State Transition Graph for a FSM
begin
1. $\mathbb{C}$ = Dtr_CD (ψ);
2. $\mathbb{C}^S$ = sort ($\mathbb{C}$);
3. Generate_Gray_code_table();
4. **while** $\mathbb{C}^S \neq \varnothing$ **do**
5. C = get the cycle with max. weight from $\mathbb{C}^S$;
6. Encode_cycle (C);
7. remove from $\mathbb{C}^S$ those cycles where more than t% of the states are encoded;
8. **end while**
9. U = get the remaining un-encoded states;
10. Assign_MWHD(U);
end

Figure 5. Cycle-decomposition-based state assignment algorithm

After all of the cycles are generated in line 1 of the algorithm, they are sorted according to their weights (line 2) and then a table of all Gray codes for the minimum required bit count is generated by *Generate_Gray_code_table* (line 3). The number of such Gray codes will be $2^{\lceil \lg_2 |S| \rceil}$. The cycles will then be encoded one by one according to the sorted order by using the *Encode_cycle* algorithm. This algorithm assigns the codes to the states on the cycle in such a way that the hamming distance of each state from its neighboring states is minimized. However, this is not feasible for those cycles whose states are partially encoded as part of a previously encoded cycle (line 7). In fact, when the number of previously encoded states in a cycle is sufficiently large, it makes more sense to switch from a Gray-coding scheme to a Minimum Weighted Hamming Distance algorithm (MWHD). That is precisely what *CD_based_encoding* does for the few un-encoded states in these cycles (lines 9 and 10.)

Gray codes are used to encode states in a cycle because these codes are the optimal solution with respect to the measured cost function, i.e., the minimum weighted hamming distance. Consider a table of Gray codes shown in Figure 6; codes are divided into two sets. A code is a

high-code if its MSB is 1, and is a *low-code* if its MSB is 0. A table of 2^{n+1}Gray codes can successively be constructed from a table of 2^n Gray codes applying a simple procedure:

1. Write a Gray code table;
2. Concatenate the table above with a copy of itself, written in reverse order;
3. Add a 0 as the MSB of the entries in the first half of the table and a 1 as the MSB of the second half;

When constructing a Gray code table according to this simple procedure, the low-codes are always on top whereas the high-codes are at the bottom. The line, which separates the high-codes from low-codes, is called the *middle-line*.

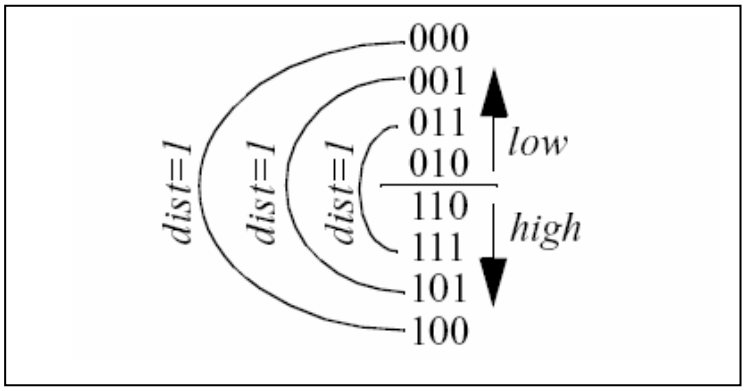


Figure 6. Gray code table

The only difference between high and low codes with equal distances from the middle-line is the MSB, thus they have a hamming distance of one. Moreover, each code differs from its neighboring codes only in one bit (this is the well-known Gray code property). Given a cycle *C*, we can encode it optimally by following a *ping-pong* movement in the Gray code table starting from the very first high-code under the middle-line and choosing the codes in high, low, low, high, high, low, ... order as shown in Figure 7.

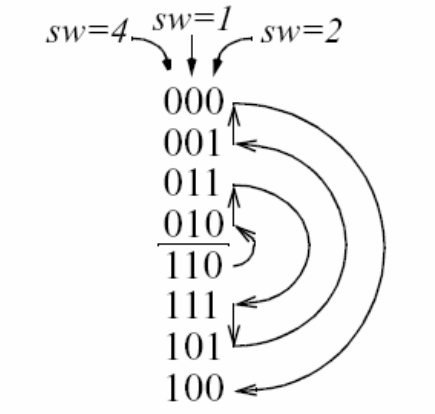


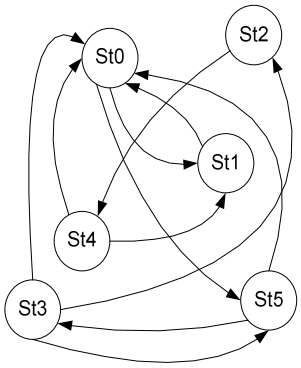
Figure 7. Ping-pong encoding and switching activity of state bits

Figure 8 shows the *Encode_cycle* heuristic algorithm for encoding cycles. For cycles with all un-encoded states, it makes no difference which state to start from for encoding. However, for cycles where some of the states have already been encoded, the start state is indeed important. In line 1, *Find_best_rotation* returns the beginning of the largest consecutive sequence of previously un-encoded states in the cycle. For example, if cycle *C* has 10 states from which two consecutive groups of size 3 and 4 states are not yet coded, *Find_best_rotation* returns the state at the start of un-encoded state sequence of size 4. The *high* and *low* codes are selected as candidate codes for the yet un-encoded state (lines 3, 4). Next *Find_best_code* is used to compare the cost for each of the two candidate codes and to pick the best one (line 5). The process is continued, traversing all states starting from the *strt* state until all un-encoded states in the cycle are visited (line 2.)

```
Procedure Encode_cycle (cycle C)
begin
1.  strt = Find_best_rotation (C);
2.  foreach un-encoded state on C starting from strt
3.      high = find first available high-code;
4.      low = find first available low-code;
5.      code(state) = Find_best_code (high, low);
6.  end for
end
```

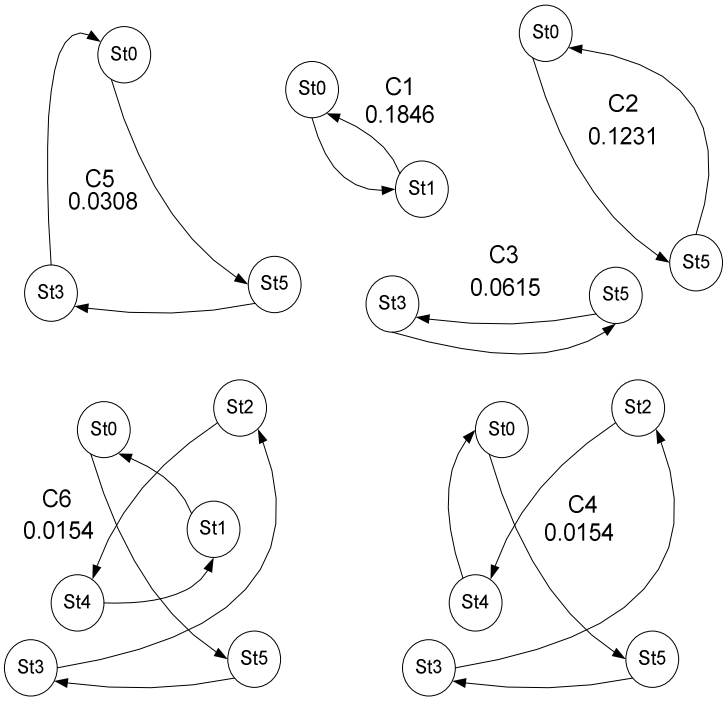
Figure 8. Cycle encoding heuristic

Example: Figure 9.a shows an FSM with 6 states and its transition probability matrix. Algorithm *Dtr_CD* chooses the edges with descending transition probability until a cycle is formed. In this example the first cycle detected is the cycle C1 (cf. Figure 9.b) since transitions St0→St1 and St1→St0 are the two most probable transitions. Next, the algorithm finds the minimum transition probability in the cycle which is St0→St1 and reduces the transition probability of all edges with that value. This would eliminate edge St0→St1 and reduces the transition probability of edge St1→St0 to 0.0154. The algorithm continues until there are no more edges left. Figure 9.b shows the set of cycles generated using this algorithm. Next, *Encode_cycle* algorithm is used to encode each cycle. First Cycle C1 is selected for encoding because it has the maximum weight; this cycle is encoded using the codes 010 and 110 (cf. Figure 7). Then Cycle C2 is selected, since in this cycle state St0 has been encoded in previous iteration, the algorithm starts from the next un-encoded state, i.e., St5 and picks the best code with respect to code of St0 to encode St5, which is 011. Next, the algorithm encodes C3, i.e. St3 is encoded with respect to St5 resulting in code 111. For Cycle C4, we note that states St0, St5, and St3 are previously encode, therefore, *Find_best_rotation* selects St2 as the start state for C4, since this is the beginning state of the largest un-encoded sequence in this cycle. The algorithm then traverses the cycle and encodes each state. St2 state is encoded as 101 and state st4 is encoded as 001. Since there are no more un-encoded states, the algorithm stops at this point. Note that the set of cycles and the order in which the cycles are being coded is very important, for example if we change the encoding order to be C5, C4, C3, C2, C1, and C6. By the time we reach cycle C1, the remaining available codes for state St1 have at least a hamming distance of two compared to the codes of state St0, but this cycle is the most probable cycle, which implies that the resulting low power code is not good. On the other hand the set of cycles generated by *Dtr_CD* are the best for this problem because this algorithm chooses those cycles which are most probable in typical operation of the FSM machine.



$$p_i \cdot p_{ij} = \begin{bmatrix} 0 & 0.1846 & 0 & 0 & 0 & 0.1846 \\ 0.20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0308 & 0 \\ 0.0308 & 0 & 0.0308 & 0 & 0 & 0.0615 \\ 0.0154 & 0.0154 & 0 & 0 & 0 & 0 \\ 0.1231 & 0 & 0 & 0.1231 & 0 & 0 \end{bmatrix}$$

a. Finite State Machine and its Transition and State probabilities



b. Finite State Machine and its Transition and State probabilities

Figure 9. Example of state assignment algorithm

III.B Experimental Results

The cycle-based encoding algorithm was implemented in C and run on an IBM IntelliStation with a 730 MHz Pentium III processor and 256 MB memory to generate the experimental results. The total weighted Hamming distances (WHD) for a number of FSM benchmark circuits and for different encoding techniques are reported in Table 1. For these results, we assumed uniform external input distribution and used equation (7) to calculate the WHD value for each state machine. The purpose of this table is to demonstrate the relative efficiency of the cycle-based encoding algorithm compared to a genetic search based algorithm proposed in [26].

The first column provides names of the FSM circuits, which are all selected from LGSynth89 or ISCAS89 benchmark sets. The largest circuit used for generating experimental results has 256

states. Column 2 shows the number of states in each FSM. Columns 3 and 4 report the average switching activity per state bit line and the runtime (in seconds) for the proposed cycle-based state assignment (i.e., the *CD_based_encoding* algorithm). All FSMs were encoded in the order of one second or less. Columns 5 and 6 report the average switching activity and runtime for a genetic search algorithm that was implemented to calculate the low power state assignment based on the minimum weighted hamming distance cost. For these results, parameter *t* (cf. line 7, Figure 5) was set to 40% for all of the above experiments. The results show a significant speed-up compared to the genetic search algorithm with nearly the same quality of results. The case of *s208* is a notable exception, where we obtain more than 70% reduction in the WHD metric. The reason for this significant improvement is that *s208* is indeed a 256 state counter. As a result we perform much better than the genetic search algorithm (the quality of GA solution may improve if it is given more computation time). Column 7 shows the WHD result of the approach presented in Reference [4] whereas column 8 shows the WHD result if the transition selection process in the deterministic cycle decomposition algorithm is performed at random instead of based on the probabilities.

FSM	# States	Cycle-based WHD	Cycle-based Time	Genetic search WHD	Genetic search time	Ref [4] WHD	Arbitrary Cycle_Gen WHD
dk16	28	1.89	0.6	1.83	220	2.23	3.54
dk512	15	1.35	0.4	1.42	130	1.33	2.34
donfile	24	1.45	0.5	1.39	165	1.55	3.43
ex1	21	0.72	0.7	0.70	155	2.55	5.12
ex2	19	1.61	0.6	1.60	100	1.78	2.45
ex5	9	1.33	0.4	1.29	42	1.62	3.43
ex7	10	1.46	0.4	1.35	56	1.37	4.23
planet	48	1.31	0.8	1.86	600	1.10	2.45
S208	256	0.50	1.6	3.55	3450	3.12	2.76
S298	218	3.12	1.2	4.50	2875	5.10	6.23
S820	25	0.49	0.5	0.55	170	0.68	3.12
S953	20	0.37	0.6	0.45	300	0.32	2.45
S1488	48	0.35	1.0	0.36	700	0.54	2.56
sand	32	0.65	0.8	0.82	390	0.89	1.99
sse	16	0.78	0.5	0.88	150	0.56	2.42
styr	30	0.57	0.7	0.61	250	0.70	1.89
tbk	32	1.05	0.8	1.17	400	2.08	3.57
train11	11	0.41	0.4	0.46	5	1.5	3.12
average		**1.07**	**0.70**	**1.37**	**564.34**	**1.61**	**3.17**

Table 1. Average Switching Activity

Table 2 shows the post-mapping area and power consumption of the FSMs using genetic search generated state codes v.s. *codes generated by CD_based_encoding* algorithm. Relative accuracy of WHD cost function can be noticed by considering tables 1 and 2 side by side.

FSM	Genetic search Power (μW)	Genetic search Area (μm^2)	Cycle-based Power (μW)	Cycle-based Area (μm^2)
dk16	1650	97852	1760	99539
dk512	429	32976	427	32864
donfile	1106	85012	1215	84711
ex1	983	72985	968	73538
ex2	997	67862	1089	68544
ex5	572	35676	507	26606
ex7	581	40227	634	41576
planet	1326	117382	1179	110112
S208	362	63120	220	54351
S298	8943	985651	8307	917559
S820	1269	134263	1137	130741
S953	1311	113176	1047	106007
S1488	1054	159124	923	153947
sand	1623	134278	1455	129910
sse	629	48319	459	43044
styr	926	134257	840	131997
tbk	2391	173216	2135	171718
train11	310	30128	240	29533
average	**1470.11**	**140305.80**	**1363.44**	**133683.20**

Table 2. Genetic v.s. cycle_encode_cycle

IV. Sequence Compaction for Power Simulation

The statistics of input vectors heavily impacts the power dissipation in a combinational circuit [1]. Therefore, to obtain accurate power estimates for a circuit, it is required to use a vector sequence, which captures the typical application data. Unfortunately, the length of such a sequence can be quite large. So it is important to find ways to construct or otherwise extract a subsequence of much shorter length that can be simulated by utilizing a low-level simulation engine to provide highly accurate power estimates of a circuit. Two approaches have been developed to address this problem: probabilistic compaction and statistical sampling.

An approach for reducing the power simulation time is to compact the given long stream of bit vectors using probabilistic automata [13]. The idea is to build a Stochastic State Machine (SSM) which captures the relevant statistical properties of a given, long bit stream, and then excite this machine by a small number of random inputs so that the output sequence of the

machine is statistically equivalent to the initial one. The relevant statistical properties denote, for example, the signal and transition probabilities, and first-order spatio-temporal correlations among bits and across consecutive time frames. The procedure then consists of decomposing the SSM into a set of deterministic state machines, and realizing it through SSM synthesis with some auxiliary inputs. The compacted sequence is generated by uniformly random excitement of such inputs. Another algorithm for vector compaction is presented in [14]. The foundation of this approach is also probabilistic in nature: it relies on adaptive (dynamic) modeling of binary input streams as first-order Markov sources of information. The adaptive modeling technique itself is best known as dynamic Markov chain modeling. A hierarchical technique for compacting large sequences of input vectors is presented in [15]. The distinctive feature of this approach is that it introduces hierarchical Markov chain modeling as a flexible framework for capturing not only complex spatio-temporal correlations, but also dynamic changes in the sequence characteristics such as different input modes. Other approaches such as [16] use spatio-temporal correlation of the input sequence as a cost function for generating a new input sequence, which can estimate the average power within reasonable error bounds. Reference [17] approximates Discrete Fourier Transform (DFT) of hamming distance between consecutive vectors in the input sequence to generate a new sequence which captures the original sequence's statistical properties. Reference [18] presents a different approach based on a graph model to transform the sequence compaction problem to that of finding a heaviest weighted trail in a directed graph. The authors then present a heuristic based on min-cost flow to solve the problem.

References [19][20] use statistical sampling techniques as another approach for solving the sequence compaction problem; these techniques use an input model based on a Markov process to generate the input stream for simulation. The simulation is performed in an iterative manner. In each, iteration, a vector sequence of a fixed length (called sample) is simulated. The simulation results are monitored to calculate the mean value and variance of the samples. The iteration terminates when some stopping criterion is met. This approach suffers from two shortcomings. First, the required number of samples, which directly impacts the simulation run time, is approximately proportional to the ratio between the sample variance and the square of the sample mean value. For certain input sequences, this ratio becomes large, thus significantly increasing the simulation run time. Second, there is a general concern about the normality assumption on the sample distribution. Since the stopping criterion is based on such assumption, if the sample distribution significantly deviates from the normal distribution, the simulation may terminate prematurely.

This section presents a new sequence compaction technique to improve the efficiency of dynamic power simulation techniques. The proposed approach is based on cycle decomposition of a Markov process that models the input vector. Average power dissipation is subsequently calculated by simulating each decomposed cycle of the Markov process. The mathematical framework for the proposed sequence compaction technique is different from the abovementioned techniques and the proposed algorithm results in higher compaction ratios for the same level of evaluation accuracy.

IV.A Sequence Compaction Problem

As mentioned before probabilistic behavior of input vectors heavily impacts the power dissipation in a combinational circuit. In our model, any input combination is regarded as a state in the Markov process. Transition from one input combination to next one is modeled by a transition between the corresponding states in the Markov process. The probability of each transition in the Markov process indicates the frequency of the corresponding input change. Associated with each transition is a value that represents the power dissipation in the circuit due to input vector change. Therefore, the average power dissipation under the modeled input vector, q_{ave}, is calculated as:

$$q_{ave} = \sum_{i,j} p_{ij}.q_{ij} \tag{9}$$

where p_{ij} is the probability of a transition from state *i* to state *j* and q_{ij} is the power dissipation in the circuit due to the corresponding input change. In this section, we first show how cycle decomposition of Markov processes enables compacting a large vector sequence that is to be applied to a circuit.

Example: Consider a combinational circuit with input vectors denoted by symbols *a*, *b*, *c* and *d*. An example input vector sequence applied to this circuit is:

a b c b d c b c b c a b c b c a

A string of length *k* is a stream of *k* consecutive vectors in the original input sequence. A cover of an input sequence is a set of strings of different length such that the original input sequence can be constructed with repetition and/or concatenation of different elements of this set. Size of a cover is defined as summation of length of all strings in that cover e.g. $\{abc, bdc, bcb, ca\}$ is a cover of size 11 and *abc* is a string of length 3.

Constrained Minimum Sequence Covering Problem (CMSC): Given an input sequence and an integer L_{min} find a cover *C* of the input sequence such that:

$$\forall s \in C \quad Length(s) \geq L_{\min}$$

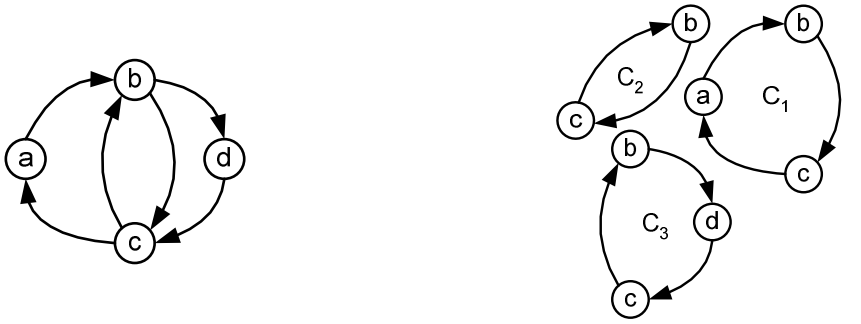
and *Size*(*C*) is minimized.

A sequence compaction problem can be formulated as a constrained minimum sequence-covering problem where L_{min} is 2, since each string should cover at least one transition in the input sequence. The average power consumption of the circuit is then calculated by:

$$P_{ave} = \sum_{\forall s \in C} N_s \cdot P_{ave_s} \tag{10}$$

where N_s is the number of times string *s* is repeated and P_{ave_s} is the average power consumption of circuit due to string s. The CMSC problem can be shown to be *NP*-complete by reducing the satisfiability problem to it.

The Markov process modeling this input sequence is shown in Figure 10. Simulation of this sequence will require 15 input transitions: $a \to b, b \to c, c \to b$, etc. Following this sample path on the Markov process, one can easily see that the whole path can be represented by summation of cycles C1, C2, and C3 where C1 is repeated twice and C2 is repeated three times. Therefore, the total power dissipation along the above input transitions is equal to the power dissipation along C3, two times C1 and three times C2, calculation of which requires only 8 transitions.



a. Markov Chain Representation b. Cycle Decomposition of the Markov Chain Model

Figure 10. Markov Chain Model of Example input Sequence

IV.B The Proposed Compaction Algorithm

Consider a sequence of input vectors applied to a combinational circuit. If we assume that the input sequence is long enough to accurately model the "typical" application data, then the Markov process generated using this input sequence is an accurate stochastic representation of the circuit inputs.

In practice, since the Markov process is generated based on the given input sequence and all the state and transition probabilities are calculated accordingly, it is assumed that the input sequence captures the stochastic behavior of Markov process in the sense that the input sequence is sufficient to provide cycle decomposition for the Markov process. Therefore, we can use the probabilistic approach described in Section II.B for the walk corresponding to the input sequence to generate the cycles that decompose the Markov process. Note that the deterministic algorithm can also be used for this purpose. However, typically only the input sequence is given for power estimation purposes and not its generating Markov chain, which is required by the deterministic algorithm. Therefore, in order to use the deterministic algorithm, one is required to solve the Chapman-Kolmogrov equations (cf. [22][23]) to get the generating Markov chain, making the probabilistic solution more appealing.

We first prove that the cycles generated along the walk corresponding to the given input sequence provide a decomposition of Markov process and then show how these cycles are used to calculate the average power consumption in the circuit. Notice that no a priori knowledge of the circuit structure is needed for the input vector compaction algorithm. This is in sharp contrast with statistical sampling techniques, which require the circuit net list.

Theorem 4: Assume that a sequence of input vectors for a combinational circuit is given and that a Markov process ξ is generated based on the given sequence. The probabilistic approach applied to the given input vector sequence produces a cycle decomposition of Markov process ξ with the following weights:

$$W_{C_k} = \frac{N_{C_k}}{\sum_l |C_l| . N_{C_l}} \tag{11}$$

where W_{C_k} is the weight assigned to cycle C_k, N_{C_k} is the number of times cycle C_k appears along the walk, and $|C_l|$ is the length of cycle C_l, i.e., the number of transitions on that cycle.

Proof: For the set of cycles with the above weights to be a decomposition of the Markov process, we have to show that (cf. equation (5)):

$$p_i p_{ij} = \sum_{\forall k | (i,j) \in C_k} W_{C_k} \tag{12}$$

The left hand side of the above equation is equal to:

$$p_i p_{ij} = \frac{N_{(i,j)}}{\sum_l |C_l| . N_{C_l}} \tag{13}$$

where $N_{(i,j)}$ is the number of times that transition from i to j appears along the walk and the denominator is basically the length of the walk itself. The right hand side of equation (12), on the other hand, is equal to

$$\sum_{\forall k | (i,j) \in C_k} W_{C_k} = \sum_{\forall k | (i,j) \in C_k} \left(\frac{N_{C_k}}{\sum_l |C_l| . N_{C_l}} \right) = \frac{1}{\sum_l |C_l| . N_{C_l}} \cdot \sum_{\forall k | (i,j) \in C_k} N_{C_k} \tag{14-a}$$

Note that the second factor in equation (14) is exactly equal to the number of times that transition from state i to state j occurs along the walk, i.e., $N_{(i,j)}$. Therefore equation (12) holds for the cycle weights defined in (11). The second equality in equation (5) holds since we have

$$\begin{aligned} p_i &= \sum_{\forall v_j} \sum_{\forall C: (v_i, v_j) \in C} W(C) = \sum_{\forall C: v_i \in C} W(C) \\ &= \sum_{\forall C} W(C) . J_C(v_i) \end{aligned} \tag{14-b}$$

and the proof is complete. ♦

The cycles and corresponding weights depend only on the walk itself - represented by the given input vector sequence - and not the Markov process. Therefore the cycle decomposition can be obtained without having an explicitly-constructed Markov process. Now that the cycles and their weights are generated, we show how the average power consumption can be calculated.

Theorem 5: Consider a combinational circuit and a sequence of input vectors applied to this circuit. Let C be a cycle decomposition based on the given input sequence with weights assigned as in equation (11). The average power dissipation of the circuit under the given input sequence is then equal to:

$$q_{ave} = \sum_{i,j} q_{C_k} . W_{C_k} . |C_k| \tag{15}$$

where q_{ave} is the average power dissipation and q_{C_k} is the average power dissipation in cycle C_k per transition.

Proof: To provide a proof, we reformulate the right hand side of equation (15) as a summation over transitions rather than cycles.

$$\begin{aligned} q_{C_k} . |C_k| &= \frac{\sum_{(i,j) \in C_k} q_{ij}}{|C_k|} \times |C_k| \\ &= \sum_{(i,j) \in C_k} q_{ij} \end{aligned} \tag{16}$$

Therefore,

$$\begin{aligned}\sum_k q_{C_k}.W_{C_k}.|C_k| &= \sum_k \sum_{(i,j)\in C_k} q_{ij}.W_{C_k} \\ &= \sum_{i,j} \sum_{\forall k|(i,j)\in C_k} q_{ij}.W_{C_k} \\ &= \sum_{i,j} q_{ij}.\sum_{\forall k|(i,j)\in C_k} W_{C_k} \\ &= \sum_{i,j} q_{ij}.(p_i.p_{ij}) \\ &= q_{ave}\end{aligned} \quad (17)$$

♦

It should be noted that $W_{C_k}.|C_k|$ indicates the probability of being in cycle C_k. Since each transition in the Markov process can be repeated along the walk and may be part of several cycles, the probability of being in a specific cycle is not the same as the summation of transition probabilities over all transitions on that cycle. The probability of being in a cycle depends on the number of times that cycle is repeated and on the length of cycle, i.e.,

$$W_{C_k}.|C_k| = \frac{|C_k|.N_{C_k}}{\sum_l |C_l|.N_{C_l}} = p_{C_k} \quad (18)$$

The numerator in the above equation is the number of transitions along the walk that are part of cycle C_k whereas the denominator is the total length of walk. Therefore, the ratio gives the probability of being in cycle C_k. One can now easily verify that $\sum p_{C_k} = 1$ and that the average power dissipation may be calculated as $\sum q_{C_k}.p_{C_k}$.

Based on the above theorems, a vector compaction algorithm for efficient dynamic power evaluation is proposed in Figure 11. In line 1, all the cycles corresponding to the given input Vector *V* are generated using the probabilistic approach. In line 2, some of the cycles are dropped by the *Prune_cycles* algorithm to achieve a higher compaction ratio, then the average power consumption is calculated using equation (3).

According to equation (5), eliminating some of the cycles changes the probabilities in the Markov process that models the input sequence. Reference [14] shows that by bounding perturbations in Markov process probabilities, one can bound the error in average power estimation. Dropping cycles with small weights minimizes perturbations in the Markov process probabilities, and thereby, minimizes the estimation error. To achieve this in algorithm Prune_cycles (cf. Figure 11), first all cycles are sorted by their descending weights (line 2). Next based on this ordering, cycles are selected until the compaction ratio reaches a user specified level, *r* (lines 6-9). After dropping some of the cycles, weights of the other cycles are updated in lines 10-12. This is done because cycle elimination is equivalent to dropping some of the transitions from the given input sequence, which in turn influences the cycle weights according to equation (18). Finally, the average power consumption is calculated from the updated weights.

Algorithm Compact(V, r)
begin
1. C=Prob_CD(V);
2. $\tilde{C}$ =Prune_cycles(C, r);
3. $q_{ave} = \sum_{c \in \tilde{C}} q_c.W_c.|c|$;

end

Algorithm Prune_cycles(C, r)
begin
1. OrigLen = $\sum_{c \in C} N_c.|c|$;
2. C_sort=Sort_cycles(C);
3. i=0;
4. $\tilde{C} = \varnothing$;
5. PruneLen=|C_sort[i]|;
6. **while** PruneLen/OrigLen <= r **do**
   begin
7. $\tilde{C} = \tilde{C} \cup C\_sort[i]$;
8. i++;
9. PruneLen + =|C_sort[i]|;
   end
10. **foreach** cycle c in $\tilde{C}$ **do**
11. $W_c = \frac{N_c.|c|}{\sum_{c \in \tilde{C}} N_c.|c|} \times W_c$;
12. **endfor**
13. **return** $\tilde{C}$;

end

Figure 11. Vector compaction algorithm

IV.C Experimental Results

The proposed algorithm is implemented in SIS framework. Average power consumption calculation is performed using an in-house gate-level logic simulator under SIS. To make a fair comparison with reference [14], we used the same input vector sequence (4000 vectors) to generate the results reported in Table 3. Column 1 shows the name of the circuit. Each circuit was first optimized by using *script.rugged* and then mapped to a 0.25μm technology library. The average power consumption for all 4,000 vectors is reported in the third column. The proposed compaction algorithm was then used to achieve the results in columns 4, 5 and 6 with compaction ratios of 2, 5 and 10, respectively. The average error for the compaction algorithm using hierarchical model presented in reference [14] is also reported for the sake of comparison. It is interesting to note that skipping the pruning step and considering all the cycles has resulted in a compaction ratio of 2 with no error in average power estimation. More precisely this means that the number of transitions after cycle decomposition and compaction is already two times smaller than the original number of transitions. Therefore, if we do not go through the cycle pruning step,

then we will not incur any error in calculation of the power consumption and yet will achieve a compaction ratio of 2.

Another set of experimental results is shown in Table 4 for an in-house generated sequence of length 100,000, which is a counter sequence with the sequence restarting at a random number after a random number of vectors. The procedure for generating the results was the same as the procedure used to generate the first set of results.

The results clearly show the effectiveness of the proposed method in estimating the average power consumption of a circuit. The exact power estimate can be obtained with compaction ratio of two. Maximum error is less than 3% and average error is less than 2% when the compaction ratio is 5. For the case where the compaction ratio is 10 the maximum and average errors are bounded by 5% and 3% respectively. Actual error values for different circuits are shown in Table 5.

Circuit	**PI #**	**Actual Power**	**Est. Pow. r=2**	**Est. Pow. r=5**	**Est. Pow. r=10**
C432	36	677.02	677.02	636.19	619.00
C499	41	1623.31	1623.31	1566.11	1553.83
C880	60	1614.61	1614.61	1599.97	1593.10
C1355	41	1623.31	1623.31	1566.11	1553.83
C1908	33	2005.98	2005.98	2013.12	2041.29
C3540	50	4967.17	4967.17	4905.58	4836.25
C5315	178	8349.15	8349.15	8246.21	8284.50
C6288	32	13128.22	13128.22	12384.74	12459.62
C7552	207	10047.56	10047.56	9596.88	9364.54
Average Error			0.00	3.03	3.94
Ref. [14] Average Error			2.67	3.55	4.74

Table 3. Average power consumption (μW @ 50MHz)

Circuit	**PI #**	**Pow. Err. % (r=5)**	**Pow. Err. % (r=10)**
9symml	9	0.37	0.86
b9	41	0.24	4.02
C432	36	1.33	2.84
C499	41	1.10	4.46
C880	60	2.60	4.96
C1355	41	1.10	4.46
C1908	33	0.79	1.24
C3540	50	1.00	2.00

C5315	178	1.10	1.20
C6288	32	1.91	2.27
C7552	207	2.79	3.04
duke2	22	0.22	0.53
misex3	14	0.47	5.03

Table 4. Average power error bounds

Circuit	**PI #**	**Actual Power**	**Est. Pow. r=2**	**Est. Pow. r=5**	**Est. Pow. r=10**
9symml	9	754.29	754.29	751.45	747.78
b9	41	392.07	392.07	391.09	407.85
C432	36	617.93	617.93	626.19	600.34
C499	41	1835.80	1835.80	1856.11	1753.83
C880	60	1710.66	1710.66	1755.76	1614.43
C1355	41	1835.80	1835.80	1856.11	1753.83
C1908	33	1643.36	1643.36	1630.26	1663.89
C3540	50	5853.80	5853.80	5912.35	5736.34
C5315	178	7497.08	7497.08	7580.18	7407.20
C6288	32	20247.15	20247.15	20635.12	19787.40
C7552	207	8265.32	8265.32	8496.22	8013.76
duke2	22	864.00	864.00	865.97	868.54
misex3	14	1061.16	1061.16	1066.16	1120.11
Average Error			0.00	1.16	2.93

Table 5. Average power consumption (μW @ 50MHz)

V. Conclusions

This paper presented a sound mathematical framework for Markov Models. The new formalism is based on the cycle decomposition of the Markov processes. The key result is that a Markov process can be decomposed into a collection of directed cycles with positive weights proportional to the probability of their traversal in a typical random walk. In one application, we proposed a new state assignment technique to reduce dynamic power consumption in FSMs. The proposed encoding algorithm reduces power consumption by an average of 15%. In a second application, we studied the vector compaction problem using the new mathematical framework. The key result was that a Markov process can be decomposed into a collection of directed cycles and therefore the input compaction ratio can be improved by an order of magnitude or higher by exercising these cycles exactly once, and then calculating the total power consumption by using the corresponding weights of those cycles. Experimental results on a large number of vector sequences and test-bench circuits were presented to demonstrate the effectiveness of the proposed approach. We envision a number of other applications in network and system design and analysis

that can benefit from the proposed mathematical formalism. Examples include data aggregation in distributed sensor networks and low power on-chip bus encoding.

References

[1] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. on Design Automation of Electronic Systems*, vol. 1, pp. 3-56, Jan. 1996.

[2] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal State Assignment of Finite State Machines," *IEEE Trans. on CAD*, vol. 4, pp. 269-285, Jul. 1985.

[3] S. Yang and M. Ciesielski, "On the Relationship Between Input Encoding and Logic Minimization," *Proc. of 23rd. Hawaii Int'l Conf. System Sciences*, vol. I, pp. 377-386, Jan. 1990.

[4] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations," *IEEE Trans. on CAD*, vol. 9, pp. 905-924, Sep. 1990.

[5] K. Roy and S. Prasad, "Syclop: Synthesis of CMOS Logic for Low-Power Application," *Proc. of Int'l Conf. on Computer design*, pp. 464-467, Oct. 1992.

[6] E. Olson and S. M. Kang, "Low-Power State Assignment for Finite State Machines," *Proc. of Int'l Workshop on Low Power Design*, pp. 63-68, April 1994.

[7] C. Y. Tsui, M. Pedram and A. M. Despain, "Low-Power State Assignment Targeting Two- and Multilevel Logic Implementation," *IEEE Trans. on CAD*, vol. 17, pp. 1281-1291, Dec. 1998.

[8] X. Wu, J. Wei, Q. Wu, and M. Pedram, "Low-Power Design of Sequential Circuits Using a Quasi-Synchronous Derived Clock," *Int'l Journal of Electronics*, Taylor and Francis Publishing Group, vol. 88, no. 6, pp. 635-643, Jun. 2001.

[9] E. Macii, M. Pedram, F. Somenzi, "High level power modeling, estimation and optimization," *Proc. of 34th Design Automation Conf.* Jun. 1997, pp. 504-510.

[10] C-S. Ding, C-Y. Tsui and M. Pedram, "Gate-level power estimation using tagged probabilistic simulation", *IEEE Trans. on Computer Aided Design*, vol. 17. no. 11, Nov. 1998, pp. 1099-1107.

[11] R. Marculescu, D. Marculescu and M. Pedram, "Probabilistic modeling of dependencies during switching activity analysis", *IEEE Trans. on Computer Aided Design*, Vol. 17, no. 2, Feb. 1998, pp. 73-83.

[12] F. N. Najm, "Transition Density: A New Measure of Activity in Digital Circuits", *IEEE Trans. CAD*, vol. 12, pp.310-323, Feb. 1993.

[13] R. Marculescu, D. Marculescu and M. Pedram, "Stochastic sequential machine synthesis with application to constrained sequence generation", *ACM Trans. on Design Automation of Electronic Systems*, vol. 5, no. 3, Jul. 2000, pp. 658-681.

[14] R. Marculescu, D. Marculescu and M. Pedram, "Vector compaction using dynamic Markov models", *IEICE Trans. Fundamentals of Electronics, Comm. and Comp. Sci.*, vol. E80-A, no. 10, Oct. 1997.

[15] R. Marculescu, D. Marculescu and M. Pedram, "Sequence compaction for power estimation: theory and practice", *IEEE Trans. on CAD*, vol. 18. no. 7, Jul. 1999, pp. 973-993.

[16] C-Y. Tsui, R. Marculescu, D. Marculescu and M. Pedram, “Improving the efficiency of power simulators by input vector compaction”, *Proc. of 33rd Design Automation Conf.* Jun. 1996, pp. 165-168.

[17] A. Macii, E. Macii, M. Poncino, R. Scarsi, “Stream Synthesis for Efficient Power Simulation Based On Spectral Transforms,” *ACM/IEEE Intl. Symp. on Low Power Electronics and Design*, CA, Aug. 1998.

[18] A. Pinar and C.L. Liu, “Compacting sequences with invariant transition frequencies,” *ACM Transactions on Design Automation of Electronic Systems* (TODAES), Volume 8 , Issue 2, 2003, Pages: 214 – 221.

[19] R. Burch, F. N. Najm, P. Yang and T. Trick, “A Monte Carlo Approach for Power Estimation”, *IEEE Trans. VLSI Sys.*, vol. 1, no. 1, pp. 63-71, Mar. 1993

[20] C-S. Ding, Q. Wu, C-T. Hsieh and M. Pedram, “Stratified random sampling for power evaluation”, *IEEE Trans. on Computer Aided Design,* vol. 17, no. 6, Jun. 1998, pp. 465-471.

[21] S. L. Kalpazidou, *Cycle Representations of Markov Processes*, Springer-Verlag, 1995.

[22] C-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M. Despain and B. Lin, "Power estimation in sequential logic circuits," IEEE Trans. on VLSI Systems, Vol. 3, No. 3 (1995), pp. 404-416.

[23] A.T. Freitas and A.L. Oliveira, “Implicit resolution of the Chapman-Kolmogorov equations for sequential circuits: an application in power estimation,” Design, Automation and Test in Europe Conference and Exhibition, 2003, Pages:764 – 769.

[24] M.P. Qian, M. Qian, “Decomposition into a detailed balance and a circulation part of an irreversible stationary Markov chain”, Sci. Sinica, Special issue II, 69-79. (Chinese and English.)

[25] http://mathworld.wolfram.com/

[26] S. Chattopadhyay and P.P. Chaudhuri, “Genetic algorithm based approach for integrated state assignment and flipflop selection in finite state machine synthesis,”, Proceedings of Eleventh International Conference on VLSI Design, 4-7 Jan. 1998, Page(s):522 - 527