Vector Compaction Using Hierarchical Markov Models

Radu Marculescu, Diana Marculescu
and Massoud Pedram

CENG 97-07

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4458

October 1996

VECTOR COMPACTION USING HIERARCHICAL MARKOV MODELS

Abstract

Power estimation has become a critical step in the design of today's ICs. Power dissipation is strongly input pattern dependent and, hence, to obtain accurate power values one must simulate the circuit with a large number of vectors that typify the application data. The goal of this paper is to present an effective and robust technique for compacting large sequences of input vectors into much smaller ones such that the power estimates are as accurate as possible and the simulation time is reduced by orders of magnitude. Specifically, this paper introduces the hierarchical modeling of Markov chains as a flexible framework for capturing not only complex spatiotemporal correlations, but also the dynamic changes in the sequence characteristics such as different operating modes or power distributions. The new framework is very effective and has a high degree of adaptability: the hierarchical model is dynamically grown according to the sequence behavior and after that is used to generate a new, shorter sequence. As the experimental results show, large compaction ratios can be obtained without significant loss in accuracy (less than 5% on average) for power estimates.

1. Introduction

With the growing need for low-power electronic circuits and systems, power analysis and low-power synthesis have become primary concerns for the CAD community. Power estimation is in general a difficult problem; the key task in this process is the accurate and fast estimation of average switching activity. To date, both simulative [1]-[4] and nonsimulative approaches [5]-[10] have been tried, each one having its own advantages and limitations [11]. More specifically, general simulation techniques provide sufficient accuracy, but at high computational cost. On the other hand, nonsimulative approaches (best represented by probabilistic power estimation techniques) are in general faster, but less accurate than those based on simulation; usually, the input correlations and the reconvergent fan-out in the target circuit make things very complicated and simplifying assumptions (like input signal independence) become mandatory. As a conclusion, the *input statistics* which must be properly captured and the *length of the input sequences* are two issues that appear to be important for power estimation. Our objective in this paper is then to solve efficiently the following problem: having an initial sequence (assumed representative for the application data of some target circuit), transform that input sequence into a smaller one, such that the new body of data represents a *good approximation* as far as total power consumption is concerned.

Of course, attempts to solve this issue do exist. In [12] authors use deterministic FSMs to model user-specified input sequences, however the ability to characterize anything but short input sequences is limited. More elaborate and effective techniques were presented in [13][14] where authors succeed in compacting large sequences without significant loss in accuracy. However, both techniques may introduce new vectors in the final compacted sequence and do not adapt very well to changes in the input characteristics. To illustrate the significance of this latter issue, we will consider an example.

Example 1: Let S_1 be a 5-bit sequence as shown in Fig.1a; in Fig.1b, we represent the word-level transition graph that corresponds to this sequence. Each state in this graph corresponds to a distinct pattern in the sequence and each edge represents a valid transition between any two patterns[1] that occur in the sequence; the label of each edge captures the conditional probability of transition from the source node to the destination node. This particular set of inputs is a subset of a larger random sequence and, on its own, behaves like a pseudorandom sequence because any vector is equally likely to be followed by any other remaining pattern. The total number of bit-flippings in the whole sequence is 36; then, dividing this value by the sequence length, we get an average value of 3 transitions per time step.

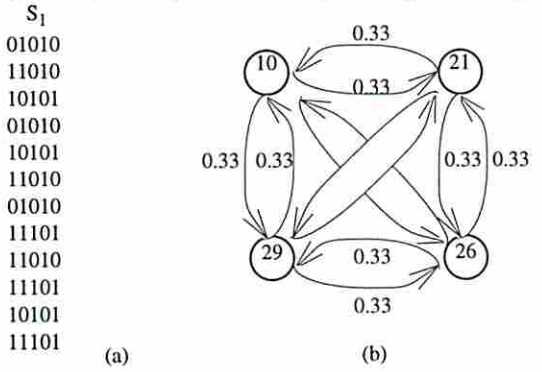

Fig. 1

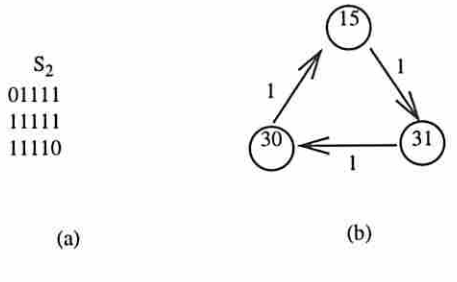

Fig.2

In Fig.2a we consider another sequence S_2, much shorter than S_1, which is a completely deterministic, highly

1. Throughout the paper, we may refer occasionally to vectors v_1, v_2,..., v_n as 'symbols', 'patterns' or 'states'.

correlated, one. It has an average value of 1.33 transitions per step, thus being 'less active' than S_1.

Suppose that we duplicate 25 times the original sequence S_1 and 100 times the sequence S_2, getting two new sequences S_1* and S_2*, respectively. These two sequences, when applied at the primary inputs of the circuit, must result in different total power consumptions. The average switching activity per bit is 0.60 and 0.27 for S_1* and S_2*, respectively, and intuitively at least, this should significantly differentiate the total power values. Indeed, applying for instance S_1* and S_2* to the benchmark C17, we get (at 20MHz) a total power consumption of 110.90uW and 38.08uW, respectively.

Assume that, based on S_1* and S_2*, we construct now a new sequence S* which is formed by concatenating S_1* and S_2* in the order $S_1^* \rightarrow S_2^* \rightarrow S_1^*$; the transition graph representation of this new 'macrosequence' is given in Fig.3.

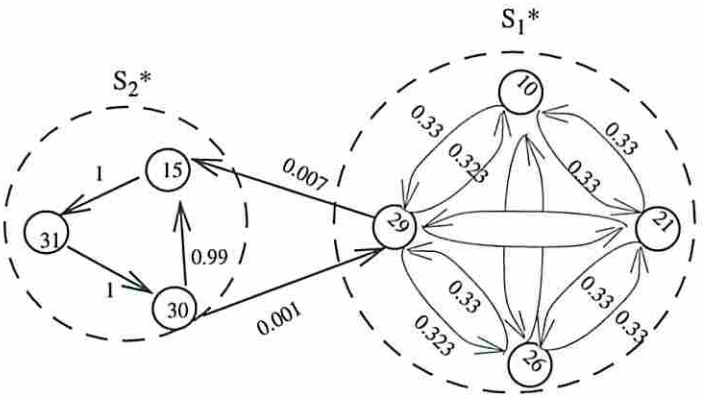

Fig.3

The question becomes now, how will the average power consumption look like as a function of time, when S* is applied to the same benchmark C17? Obviously, the circuit has now three very different modes of operation: one where a lot of activity is generated at the primary inputs, a second one where the stimuli stabilize and about one single input bit toggles at every time step, and finally, once again the circuit has high activity at its primary inputs.

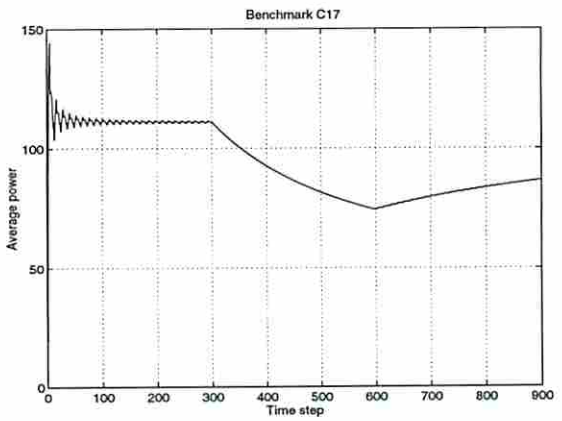

Fig.4

In Fig.4 we can see the effect of these three different regimes on average power consumption. Starting initially with S_1*, after 300 time steps the value of average power stabilizes around 110uW; after that, when the characteristics of the input sequence change, it goes down toward 70uW and finally, due to the increase of the switching activity at the primary inputs, it tends to come back towards 90uW.

This type of behavior is very common in practice. More precisely, only homogenous input sequences (which contain statistically similar vectors) will exercise the circuit such that the value of average power will converge rapidly. A typical example is a set of pseudorandom vectors where the average power value stabilizes after tens of vectors. However, in practical applications, the stimuli may contain a mixture of vectors, each one very different as far as average switching activity per bit is concerned. The approach suggested in [13] tries to model the average behavior which might result when such a mixture of vectors is present at the primary inputs of the circuit. To this end, the authors synthesize first a stochastic machine that is probabilistically equivalent to the initial sequence. After that, using randomly generated inputs, a random walk through the states of the transition graph is emulated with the stochastic machine and a new, much shorter sequence is generated. However, a compaction procedure based on random walks in graphs in which some pairs of vectors have very small transition probabilities, has the potential drawback of 'hanging' into a small subset of states. For instance, in Fig.3 it is very likely that a random walk will remain in either of the two subgraphs due to the very small probability of transitions between the two components. As a consequence, an erroneous power value can be obtained depending on which component (low or high activity) is visited. The same type of phenomenon is observed for statistical methods when the distribution is very different from a normal one (e.g. bi-modal distributions), or when one selects from the initial sequence only the first few hundreds vectors. Thus, in practice, in order to compact large sequences that contain non-homogeneous power behaviors, a technique with high adaptability is needed.

The present paper improves the-state-of the art by providing an original solution for the vector compaction problem. The foundation of our approach is probabilistic in nature; it relies on *adaptive (dynamic) modeling* of binary input streams as first-order Markov sources of information. As distinctive feature, we use hierarchical Markov models to structure the input space into a hierarchy of macro- and micro-states: at the highest level in the hierarchy we have a Markov chain of macrostates; at the second-level, each macrostate has attached to it a Markov chain for all microstates that are grouped together within that macrostate. Our primary motivation in doing this structuring is to enable a better modeling of the different stochastic levels that are present in sequences that arise in practice. Another important property of such models is to individualize different operating modes of a circuit or system via higher levels in the hierarchical Markov model, thus providing a high adaptability to different system operating modes (e.g. active, standby, sleep, etc.). The structure of the model itself is general and, with further extensions, can allow an arbitrary number of activations of its sub-models.

Once we have constructed the hierarchy for input sequence, starting with a macrostate, a compaction procedure with a specified compaction ratio is applied to compact the set of microstates within that macrostate. When done with processing the current macrostate, the control returns to the higher-level in the hierarchy and, based on the conditional probabilities that characterize the Markov chain at this level, a new macrostate is entered and the process repeats. While the compaction procedure in [13] can be adapted to work in this new environment, we preferred to combine instead the advantages offered by the hierarchical model with a new technique, computationally more efficient, called dynamic Markov chain (DMC) modeling. The DMC technique itself was introduced very recently in the literature on data compression [15] as a candidate to solve various compression problems. From the very beginning, this technique looked very promising and indeed, in most practical situations, has been more effective than any other compression technique available to date. However, the original model introduced in [16] is not completely satisfactory for our purposes. In this paper, we thus extend the initial formulation to manage not only correlations among adjacent bits that belong to the same input vector, but also correlations between successive input

patterns.

As demonstrated and supported by practical evidence, this new framework is extremely effective in power estimation. The basic idea is illustrated in Fig.5. To evaluate the total power consumption of a target circuit for a given input sequence L_0 (Fig.5a), we derive first the hierarchical Markov model of the input sequence and after that, having this compact representation, we generate a much shorter sequence L, equivalent with L_0, which can be used with any available simulator to derive accurate power estimates (Fig.5b).

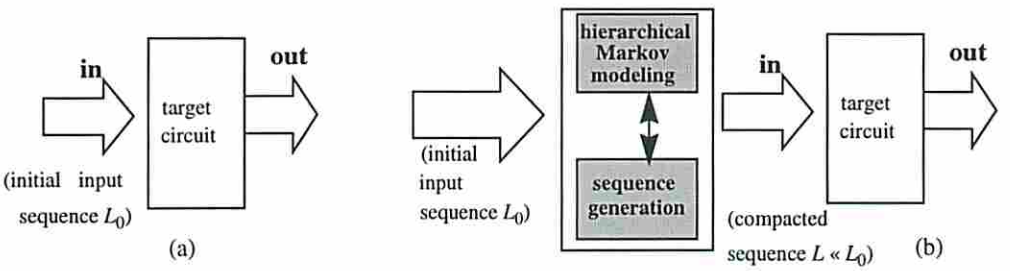

Fig.5

To conclude, both simulation-based approaches and probabilistic techniques for power estimation may benefit from this research. The issues brought into attention in this paper are new and represent an important step toward reducing the gap between the simulative and probabilistic techniques commonly used in power estimation. Finally, the concept of hierarchical Markov modeling itself may be useful in other CAD applications.

The paper is organized as follows: Section 2 introduces and characterizes the concept of hierarchy associated with an input sequence. Section 3 formalizes the power-oriented vector compaction problem and discusses parameters which make this approach effective in practice. Section 4 presents a DMC-based procedure for vector compaction. In sections 5 and 6, we give some practical considerations and experimental results, respectively. Finally, we conclude by summarizing our main contribution.

2. Hierarchical input space. Micro/macro-state modeling.

Having in mind the issues described in the Introduction, our task now is to partition the transition graph associated with a vector sequence into subgraphs that correspond to different behaviors (in particular, different power consumptions). To do this, we provide first some useful definitions and results.

Definition 1. (Lag-one Markov chain)

A discrete stochastic process $\{v_n\}_{n \geq 1}$ is said to be a lag-one Markov chain (or Markov chain for simplicity) if at any time step n:

$$p(v_n | v_{n-1} v_{n-2} \cdots v_0) = p(v_n | v_{n-1}) \tag{1}$$

where $p(v_n | v_{n-1}) = p(v_n v_{n-1}) / p(v_{n-1})$ is the conditional probability of v_n given v_{n-1}.

In other words, the probability of having a symbol depends only on the previous vector and is independent on the 'history' of the sequence.

Proposition 1.

The graph associated to a Markov chain source consists of *exactly one* strongly connected component.

Hint for proof: A state of the chain is reachable from any other state either directly or in a finite number of steps.■

In what follows, we will talk interchangeably about Markov chains and their corresponding transition graph.

Definition 2. (weighted transition graph)

A *weighted* transition graph is a directed graph where any edge between states s_i and s_j is labelled with a conditional probability *and* a weight w_{ij} associated with the transition $s_i \rightarrow s_j$.

Definition 3. ((ε, δ)-grouping)

A grouping $\{S_1, S_2,...,S_p\}$ on the set of states $\{s_1, s_2,..., s_n\}$ of the transition graph is called an (ε, δ)-grouping if:

- (ε-criterion): for any $s_i \in S_k$ and $s_j \in S_l$ we have $p(s_i s_j) < \varepsilon$ and $p(s_j s_i) < \varepsilon$;
- (δ-criterion): for any S_k there is some W_k such that $| W_k - w_{ij} | < \delta$, for any two states $s_i, s_j \in S_k$. Also, if $k < l$, W_k's are such that $W_k < W_l$.

The intuitive reason for the above definition is the following: each S_k (called from now on *macrostate*) gathers a subset of states (referred to as *microstates)*. All transitions within a macrostate have 'similar' weights. For instance, in Fig.3 microstates '10', '21', '26', '29' form the macrostate S_1* (with high activity), while '15', '30', '31' form S_2* (with low activity). However, in general, a particular microstate may appear in more than one macrostate since not only the vector itself, but also the context in which it appears is important (as seen in Definition 3, the weight value for a transition determines whether the microstates belong to the macrostate or not). So actually the *grouping* of states is done such that transitions are *clustered* according to the weight associated to them.

From what we defined so far, we are able to structure the input space hierarchically. More precisely, instead of considering the input sequence as a flat sequence of vectors we can see it as a structured multi-level discrete stochastic process called *Hierarchical Markov Model* (HMM). HMM generalizes the familiar Markov chain concept by making each of its macrostates a similar stochastic model on its own, i.e. each macrostate is a HMM as well. For instance, the graph in Fig.3 can be represented hierarchically as shown below where the macrostate S_1* identifies the high activity mode and S_2* the low activity one.

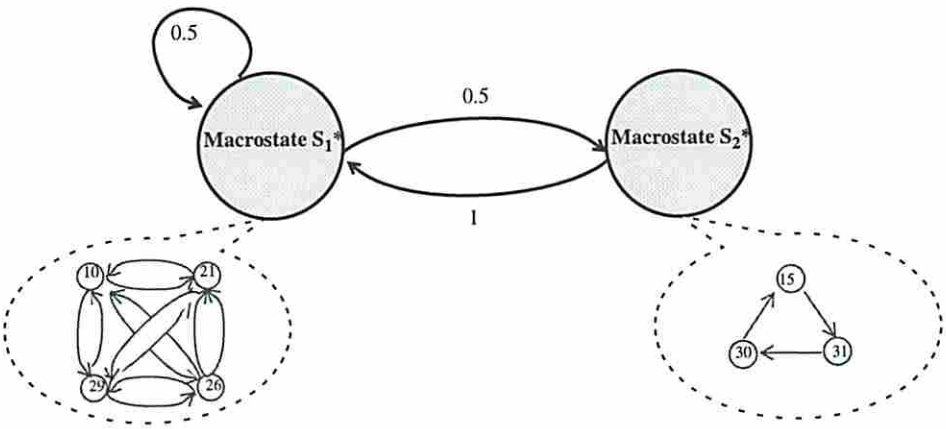

Fig.6

The initial problem of compacting a flat input sequence becomes now that of compacting a hierarchy of subsequences. Since the vectors from each macrostate are gathered using the same (ε, δ) criterion, the compaction is done now inside each macrostates preserving the high-level model. This way, the 'hang-up' problem mentioned in the introductory part is avoided, that is all macrostates are guaranteed to be visited (as their transition probabilities 'scale-up' after hierarchization). For instance, if in Fig.3 the transition probabilities between S_1* and S_2* were 0.001 and 0.007 respectively, in the hierarchical organization in Fig.6 they become 0.5 and 1, respectively.

In what follows we present some results useful for HMM characterization.

Theorem 1.

If the state probability of each macrostate and the state probabilities for all microstates within a macrostate are preserved, then the state probability distribution for the initial (flat) sequence is completely captured.

Proof: Let s_i be any state from the original sequence. Then, its probability is given by:

$$p(s_i) = \sum_{S_k} p(s_i S_k) = \sum_{S_k} p(s_i | S_k) \cdot p(S_k) = \sum_{S_k} p_k(s_i) \cdot p(S_k)$$

where $p_k(s_i)$ denotes the state probability of s_i in macrostate S_k. ■

Thus, the state probabilities are the same if they are preserved inside each macrostate and also the probability distribution for macrostates is correctly captured.

Theorem 2.

In a hierarchical model satisfying the ε–criterion, if transition probabilities of the microstates are preserved within each macrostate and if the state probabilities of the macrostates are correctly captured, then transition probabilities of the initial sequence are reproduced with an error less than or equal to ε.

Proof: The transition probability between two states s_i and s_j can be expressed as:

$$p(s_i s_j) = \sum_{\substack{S_k \\ s_i, s_j \in S_k}} p(s_i s_j S_k) + \sum_{\substack{S_k, S_l \\ s_i \in S_k, s_j \in S_l, k \neq l}} p(s_i s_j S_k S_l) = \sum_{S_k} p_k(s_i s_j) \cdot p(S_k) + \sum_{S_k, S_l} p_{kl}(s_i s_j) \cdot p(S_k S_l)$$

where $p_k(s_i s_j)$ is the transition probability between microstates s_i and s_j inside macrostate S_k and $p_{kl}(s_i s_j)$ denotes the transition probability between those states if they belong to two different macrostates. Since our assumption is that the hierarchy satisfies the ε-criterion, we have that $p_{kl}(s_i s_j) < \varepsilon$ and hence:

$$p(s_i s_j) \leq \sum_{S_k} p_k(s_i s_j) \cdot p(S_k) + \varepsilon \cdot \sum_{S_k, S_l} p(S_k S_l) = \sum_{S_k} p_k(s_i s_j) \cdot p(S_k) + \varepsilon \,.\blacksquare$$

Thus, if the macrostate probability distribution and the transition probabilities inside each macrostate are preserved, the actual transition probabilities are the same within some ε.

Definition 4. (Weight of a random walk)

The weight of a random walk in a weighted transition graph is given by:

$$W = \sum_{s_i, s_j} p(s_i) \cdot p(s_j | s_i) \cdot w_{ij} = \sum_{s_i, s_j} p(s_i s_j) \cdot w_{ij}$$

where w_{ij} is the weight associated with transition $s_i \rightarrow s_j$.

Theorem 3.

If the hierarchy satisfies the (ε, δ) property (as in Definition 3), then the weight of a random walk in the flat model satisfies:

$$W \leq \sum_{S_k} p(S_k) \cdot W_k + \varepsilon \cdot \sum_{\substack{s_i, s_j \\ s_i \in S_k, s_j \in S_l, k \neq l}} p(S_k S_l) \cdot w_{ij} + \delta$$

Proof: Let W be the weight of the initial sequence. Then, using Definition 4 and Theorem 2, we have:

$$W \le \sum_{s_i, s_j} \sum_{S_k} p_k(s_i s_j) \cdot p(S_k) \cdot w_{ij} + \varepsilon \cdot \sum_{\substack{s_i, s_j \\ s_i \in S_k, s_j \in S_l, k \neq l}} p(S_k S_l) \cdot w_{ij}$$

On the other hand, if s_i, s_j are in the same macrostate S_k and the hierarchy satisfies the δ–criterion, then there is some W_k such that $|w_{ij} - W_k| < \delta$. Hence, we also have:

$$W \le \sum_{s_i, s_j} \sum_{S_k} p_k(s_i s_j) \cdot p(S_k) \cdot (W_k + \delta) + \varepsilon \cdot \sum_{\substack{s_i, s_j \\ s_i \in S_k, s_j \in S_l, k \neq l}} p(S_k S_l) \cdot w_{ij} \text{ or equivalently}$$

$$W \le \sum_{S_k} p(S_k) \cdot W_k + \varepsilon \cdot \sum_{\substack{s_i, s_j \\ s_i \in S_k, s_j \in S_l, k \neq l}} p(S_k S_l) \cdot w_{ij} + \delta \text{ which is exactly the above claim.}\blacksquare$$

In other words, a random walk on the hierarchical Markov model *preserves* up to some error the average weight of the original sequence. The first term in the above sum represents the average weight per macrostate, while the second accounts for the weight of transitions between them.

3. Power-oriented data compaction

In this section, first we review the main concerns about preserving the input statistics to obtain accurate power estimates. Second, we present an approach for macrostates recognition based on average Hamming distance.

3.1 Problem formulation

Capturing only signal probabilities at the primary inputs of the circuit is not enough for accurate estimates therefore, for power estimation purposes, it is critical to distinguish between sequences which exhibit the same signal probabilities on different bit lines, yet showing very different spatial and temporal correlations. Assuming that a gate level implementation is available, to estimate the total power dissipation, one can sum over all the gates in the circuit the average power dissipation due to the capacitive switching currents, that is: $P_{avg} = (f_{clk}/2) \cdot V_{DD}^2 \cdot \sum_n (C_n \cdot sw_n)$, where f_{clk} is the clock frequency, V_{DD} is the supply voltage, C_n and sw_n are the capacitance and the average switching activity of gate n, respectively. From here, the average switching activity per node (gate) is the key parameter that needs to be correctly determined, mostly if we are interested in a node-by-node basis power estimation. However, this parameter is highly sensitive to the input statistics, namely it depends significantly on transition and conditional probabilities among different signal lines.

Having these issues in mind, the vector compaction problem can be formulated as follows: for a k-bit sequence of length n (consisting of vectors $v_1, v_2, ..., v_n$), find another sequence of length $m < n$ (consisting of the subset $u_1, u_2, ..., u_m$ of the initial sequence), such that the average transition probability on the primary inputs is preserved *wordwise*. More formally, for any generic input v and u (seen as a collection of bits) in the original and in the compacted sequence, respectively, the following holds:

$$\left| p(v^- = \alpha \wedge v^+ = \beta) - p(u^- = \alpha \wedge u^+ = \beta) \right| < \varepsilon \qquad (2)$$

In relation (2), v^-, v^+ (u^-, u^+) denote the current and the next vector, respectively, in the original (compacted) sequence and α, β are any two patterns that appear in the initial sequence. This condition simply requires that the joint transition probability for any group of bits is preserved within a given level of error, for any two consecutive vectors.

If the input sequence is hierarchically structured, Theorem 2 guarantees that (2) is still satisfied. Moreover, Theorem 3 guarantees that the average power value is maintained. From a practical point of view this is very important because the hierarchical model has the advantage of being *highly adaptive* as opposed to a flat processing of the input which does well 'on average'.

3.2. A Hamming distance-based criterion for microstate grouping

In practice, it is hard to determine the weight w_{ij} for each individual transition. This would mean to have detailed information about the circuit (e.g. capacity loads, internal structure) and to employ a simulation procedure (full or symbolic simulation) in order to derive the exact power consumption for each pair of vectors from the original sequence [20]. For all practical purposes, this is at least inconvenient if not impossible, and therefore we suggest a different, circuit independent, criterion to structure the input space. As suggested in Example 1, what we need is an indicator for the level of activity at the primary inputs. To this effect, we propose the average Hamming distance between two consecutive vectors because, from our investigation, it seems to be the most reliable indicator for average power consumption[1]. To support this claim, we provide in Fig.7 the variation of average Hamming distance for benchmark C17 using the same sequence S* as in Example 1. The intuition behind this close tracking between average power and Hamming distance is that the latter implicitly captures the bit changes between any two consecutive input vectors.

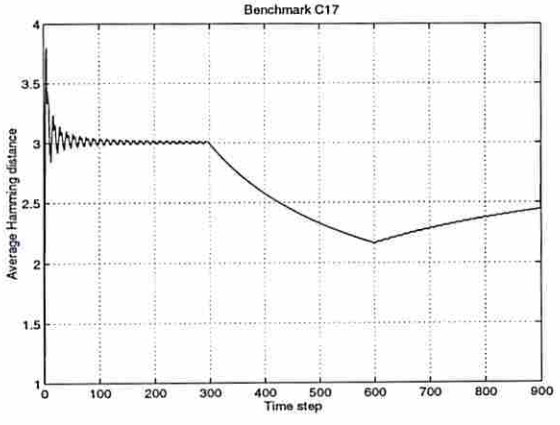

Fig.7

In this particular example, based on the Hamming distance criterion, we can roughly classify the input sequence into 'high activity' and 'low activity' macrostates (that is, if more than 3 out of 5 bits change, the we are in the high activity mode, otherwise in low activity mode). While this kind of partitioning into high and low activity modes can be always used, in practice is better to have a more refined model. For instance, if the set of all possible values for the Hamming distance is divided in three equally-sized regions that correspond to low, medium and high activity, then we can identify more than two modes of operation. A more refined model might be required if we are not only interested to preserve the total power consumption, but we are also required to use more than one mode of operation (e.g. an initialization mode, a high-impedance mode and a normal operation mode).

More precisely, having an input sequence on k bits, the interval $[0, k]$ is divided into a number of macrostates such that inside each macrostate the δ-criterion is satisfied for some value δ. To detect the changes in the input

1. However, the framework provided by our approach is also open to accept any kind of weight function which can be obtained if some simulation support is available.

sequence, a *sliding window* is used. We note that the *size* of the window should not be chosen too small (due to the fragmentation, the top-level Markov chain becomes very similar to the flat model) or too large (we may end up with a macrostate containing more than one mode of operation in it). However, our experience shows that a window size of tens of vectors works well on sequences that satisfy the (ε, δ) property.

4. A DMC-based vector compaction procedure

In this section, we introduce first the basic structures for DMC modeling and after that we describe a practical procedure for sequence compaction.

4.1 Background on dynamic Markov models

Without loss of generality, in what follows we restrict ourselves to finite binary strings, that is, finite sequences consisting only of 0's and 1's. The set of events of interest is the set S of all finite binary sequences on k bits. A particular sequence S_1 in S consists of vectors v_1, v_2,..., v_n (which may be distinct or not), each having a positive occurrence probability. Indices 1, 2,..., n represent the discrete time steps when a particular vector is applied to a target circuit. Imposing a total ordering among bits, such a sequence may be conveniently viewed as a binary tree (let's call it DMT_0 from *Dynamic Markov Tree of order zero*) where nodes at level j correspond to bit j ($1 \leq j \leq k$) in the original sequence; each edge that emerges from a node is labelled with a positive count (and therefore with a positive probability) that indicates how many times the substring from the root to that particular node, occurred in the original sequence. For clarity, let's consider the following example.

Example 2: For the following 4-bit sequence consisting of 8 non-distinct vectors: (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8) = (0000, 0001, 1001, 1100, 1001, 1100, 1001, 1100) the construction of the tree DMT_0 is shown step-by-step in Fig.8a. Obviously, the whole Markov tree that models this sequence must have four levels because the original sequence is a 4-bit sequence. Without any loss in generality, we assume a left-to-right order among bits that is, the leftmost bit in any vector v_1 to v_8 is considered as being bit number one (and consequently represented at level one in DMT_0 as shown in Fig.8a), the next bit is considered as being bit number two and so on. Every time a vector is completely scanned (that corresponds to reaching the level four in the tree), we come back to the root and start again with the next vector in the sequence. While the input sequence is scanned, the actual counts on the edges are dynamically updated (as shown in Fig.8a for the first two vectors) such that, for this particular example, they finally become as indicated in Fig.8b. The Markov tree in Fig.8b contains in a compact form all the spatial information about the original sequence v_1, v_2,..., v_8. Another approach would have been to consider a static binary tree capable to model any 4-bit sequence and just to update the counts on the edges while scanning the original sequence. By doing so, we would end up with the obvious disadvantage of having 15 instead of 9 nodes in the structure for the same amount of information; this reason alone is sufficient for considering from now on only dynamically grown models.

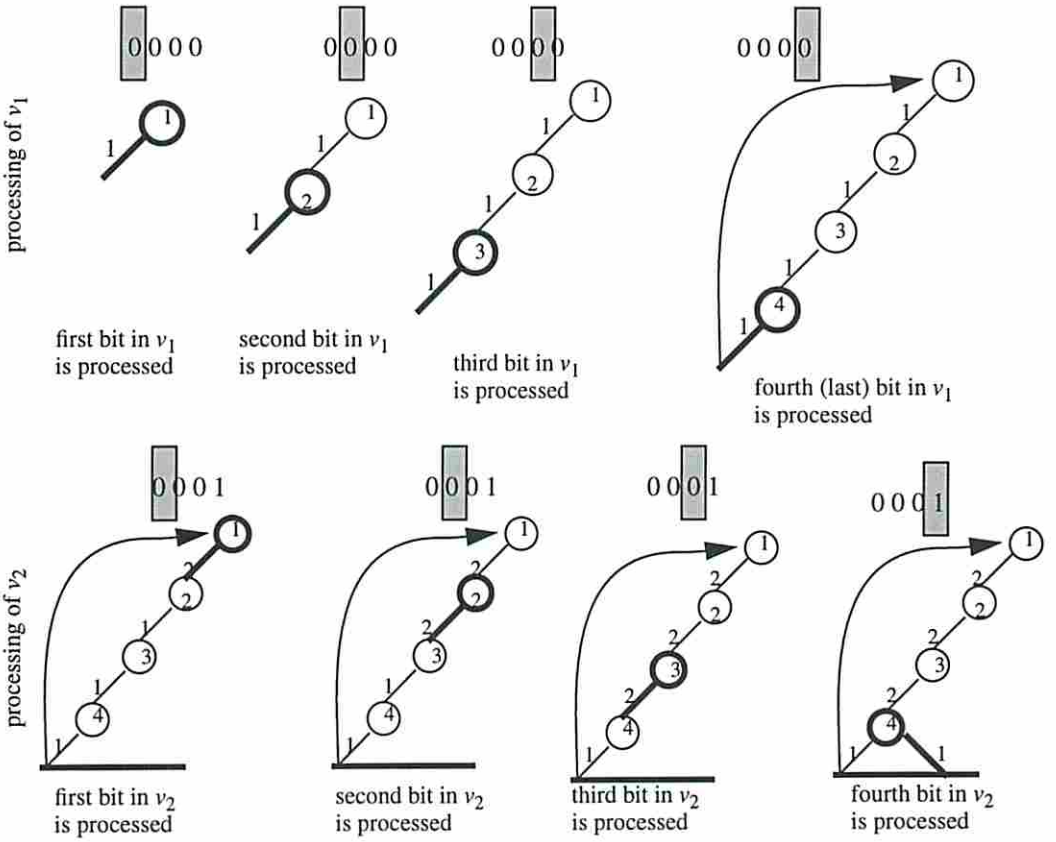

Fig.8 (a)

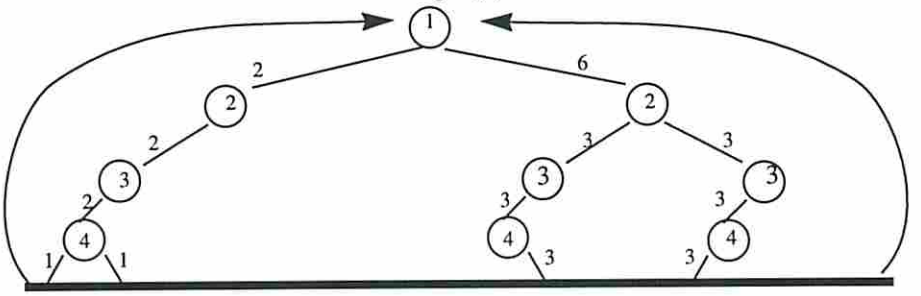

Fig.8(b)

Proposition 2. At every node in DMT_0 we have:

$$p(v) = \sum_{x \in S} p(vx) \tag{3}$$

for all v in S, where vx represents the event corresponding to the joint occurrence of the strings v and x.

The above condition, simply states that the sum of the counts attached to the immediate successors of node v equals its own value $p(v)$. As we can easily see in Fig.8, condition (3) is satisfied at every node in this representation[1]. In addition, based on the counts of the terminal edges, we may easily compute the probability of occurrence for a particular vector in the sequence. For instance, the probability of occurrence for string '1001' is 3/8 (because the

1. This is actually similar to Kirchoff's law for currents.

count on the terminal edge that corresponds to '1001' is 3 and the length of the sequence is 8) while the probability of string '1111' is zero, '1111' being a 'forbidden' vector for this particular sequence.

Spatiotemporal correlations that characterize a particular sequence are the key factor in power estimation. Differently stated, not only a particular vector v_i in a given sequence is important, but also its relative position in that sequence matters. More precisely, different interleavings among the vectors belonging to the same initial set $(v_1, v_2,..., v_n)$ (e.g.$(v_1,...,v_i, v_j, v_k,...,v_n)$, $(v_1,...,v_i, v_k, v_j,...,v_n)$ or $(v_1,...,v_k, v_i, v_j,...,v_n)$) define completely different input sequences. We observe that DMT_0 alone cannot capture this property; we say that DMT_0 has *no memory* and therefore the relative order of vectors in the initial sequence is irrelevant for DMT_0's construction. In Fig.8b for instance, the value of 3/8 is the probability to see the particular string (state) '1001' in the original sequence but this gives us no indication at all about the sequencing of this vector relative to another one, say '0001'.

To solve properly the compaction problem, we refine now the above structure by incorporating in it *first-order memory effects*. Specifically, we consider a more intricate structure, namely a tree called DMT_1 (*Dynamic Markov Tree of order 1*), where from the node representing any vector v there is an emergent arc to each value x connecting v to the successor node, associated with the string vx.

Example 3: For the same sequence in Example 2, suppose we want to construct its corresponding tree DMT_1. We begin as in DMT_0 and for each leaf that represents a valid combination in the original sequence, we construct a new tree (having the same depth as DMT_0) which is meant to preserve the *context* in which the next combination occurs. For instance, the vector v_2 = 0001 follows immediately after v_1 = 0000; consequently when we reach the node that corresponds to v_1 (the leftmost path in Fig.9a), instead of going back to the root (and therefore 'forgetting' the context), we start to build a new tree (rooted at the current leave of DMT_0) as indicated in Fig.9a. Basically, we added a new path which corresponds to '0001'. The newly constructed tree will preserve the context in which v_2 = 0001 occurred that is, immediately after v_1 = 0000 (denoted by $v_1 \rightarrow v_2$). After processing the pair (v_1,v_2), we come back to the root and continue with (v_2,v_3) as shown in Fig.9b; v_2 alone leads us to the second leftmost edge of DMT_0 from where, to construct DMT_1, we have to add the path '1001' which corresponds to v_3. In this way, we indicate the sequencing between v_2 and v_3 that is, $v_2 \rightarrow v_3$.

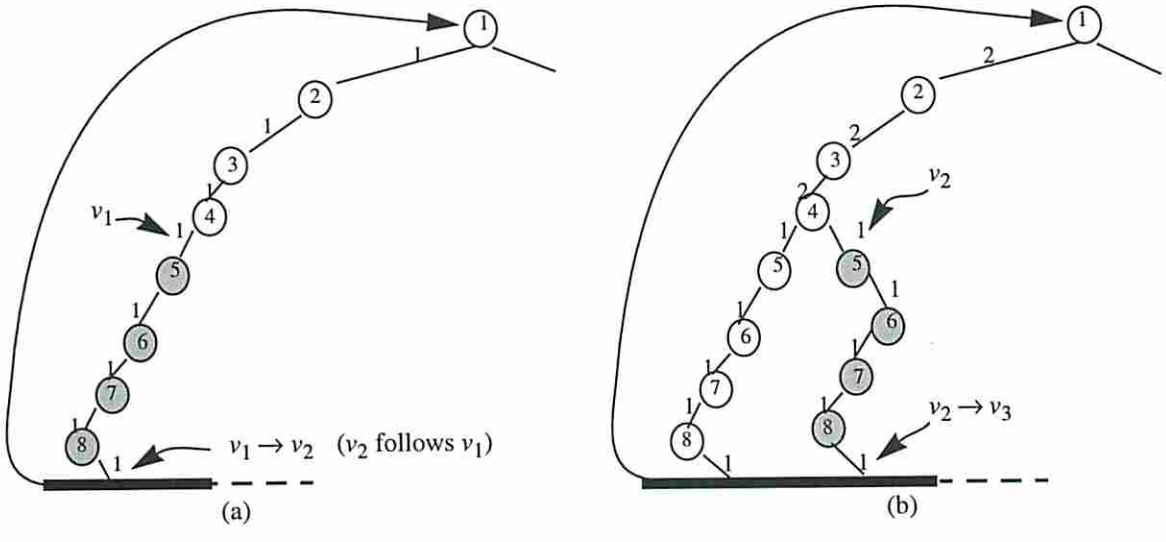

Fig.9

What is important to note here, is that *all* vectors in the original sequence are processed that is, *none of them is skipped* during the construction of DMT_1. This is the theoretical basis for accurate modeling of the input sequences as first-order Markov sources of information. Similarly, continuing this process for all leaves in DMT_0 in Fig.9b, we end up by building the whole tree DMT_1 as shown in Fig.10.

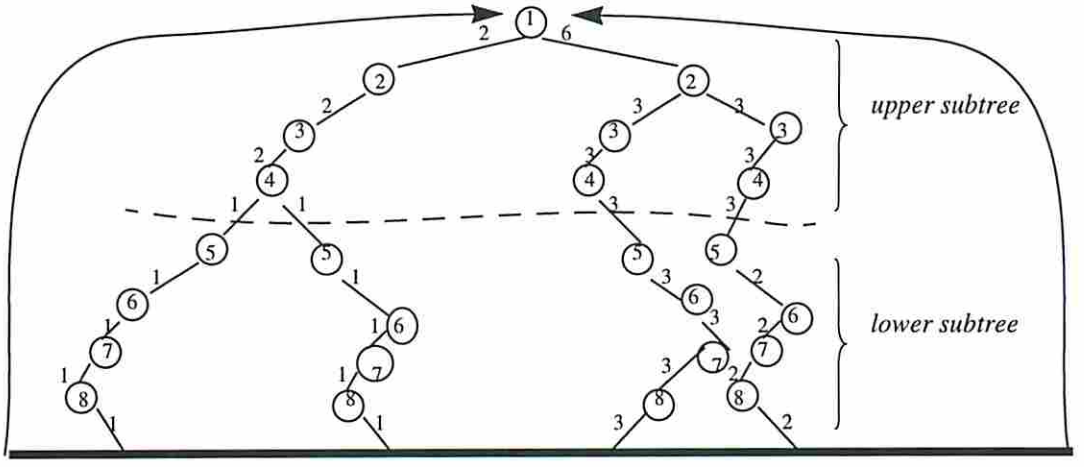

Fig.10

In Fig.10, we separated by a dashed line the two subtrees that constitute DMT_1. The upper subtree (levels 1 to 4) represents DMT_0, that is, it sets up the state probabilities for the sequence; the lower subtrees (levels 5 to 8), give the actual sequencing between any two successive vectors. To keep the counts in these subtrees consistent, while we traverse the lower subtrees and update the counts on their edges, we also accordingly increment the counts on the paths in the upper subtree (in fact, all vectors except the first and the last are processed exactly twice, once in the upper DMT_0 and next in the lower DMT_0).[1]

Obviously, DMT_1 provides more information than DMT_0. To give an example, string '1001' can follow only after '0001' or '1100', information that cannot be gathered by analyzing DMT_0 alone.

Proposition 3 [17]. We write the probability of a vector string $v = v_1 v_2 ... v_n$ as follows:

$$P(v) = P(v_1) \cdot P(v_2 | v_1) \cdot \ldots \cdot P(v_n | v_1 v_2 \ldots v_{n-1}) \tag{4}$$

This property, used in connection with the counts on the edges, allows a quick calculation of the transitions probabilities that characterize a particular sequence. For example, if we want to calculate the transition probability '1001' $\rightarrow$ '1100' we have from Proposition 3 $P(v) = P(v_1 v_2) = P(v_1) \cdot P(v_2 | v_1) = 3/8$ which is exactly the count on the path '10011100' in the tree DMT_1 divided by the sequence length.

Theorem 4. Any sequence in S can be modeled as a first-order Markov source using the structure DMT_1 and its parameters. We call this process Dynamic Markov Chain (DMC) modeling.

Hint for proof: If $v = v_1 v_2$ is a string in the structure DMT_1 such that v_1 is in the upper tree and v_2 is in the lower tree, then $P(v_2 \mid v_1) = P(v) / P(v_1)$. Thus, the parameters stored on the edges of DMT_1 structure provide the conditional probabilities that characterize the lag-one Markov chain for the sequence in S. ■

1. For simplicity, we also link the last vector in the sequence with the first.

4.2 A practical procedure for sequence generation

A practical procedure to construct DMT_1 and generate the compacted sequence is given in Fig.11a. First, vectors are assigned to macrostates during a one-pass traversal of the input sequence based on average Hamming distance as explained in Section 3.2.

During the second traversal of the original sequence (when we extract the bit-level statistics of each individual vector $v_1,v_2...,v_n$ and also those statistics that correspond to pairs of consecutive vectors (v_1v_2), (v_2v_3),...,$(v_{n-2}v_{n-1})$,$(v_{n-1}v_n)$) we grow simultaneously the trees DMT_1 inside each macrostate (the low-level of the hierarchy) and also the DMT_1 tree for the sequence of macrostates (the top-level of the hierarchy). The vectors within each macrostate are sequenced together in the same DMT_1. If the input sequence satisfies the (ε, δ) criterion, the transitions introduced this way do not change much the characteristics (average weight and transition probabilities) of the model. We continue to grow the trees at both levels of hierarchy as long as the Markov model is smaller than a user-specified threshold (*model_size*), otherwise we just generate the new sequence up to that point and discard (flush) the model (we will see later why these flushes are necessary). A new Markov model is started again and the process is continued up to the end of the original sequence. The *generate_seq* procedure called by the DMC program is detailed in Fig.11b. Each generation phase is driven by the user-specified compaction parameter *ratio* that is, in order to generate a total of $m = n/ratio$ vectors, we have to keep the same compaction ratio for every dynamically grown Markov model. We also note that this strategy does *not* allow 'forbidden' vectors that is, those combinations that did not occur in the original sequence, will not appear in the final compacted sequence either. This is an essential capability needed to avoid 'hang-up' ('forbidden') states of the circuit during simulation process for power estimation.

```
procedure DMC (input_file, ratio, model_size) {
  M = assign_μstates_to_Mstates (input_file);
  while (!EOF (input_file)) {
    μstate = read_input (input_file);
    if (number_of_nodes < model_size) {
      Mstate = determine_Mstate (mstate);
      update_tree (μstate, low_level (model, Mstate));
      if (Mstate != previous_Mstate)
        update_tree (Mstate, top_level (model));
    }
    else {
      generate_seq (model, M);
      flush_model (model);
    }
  }
  generate_seq (model, M);
}
```

(a)

```
procedure generate_seq (model, M) {
  do {
    Mstate = generate_symbol (top_level (model));
    m = length_to_be_generated (model, Mstate);
    generate_μ_seq (model, Mstate, m);
  }
  until M macrostates are generated;
}
```

```
procedure generate_μ_seq (model, Mstate, m) {
  do
    μstate = generate_symbol (low_level (model, Mstate));
  until m microstates are generated;
}
```

(b)

Fig.11

For simplicity, in what follows, we give an example which shows the DMT_1 construction for a flat model. The extension to the hierarchical model is straightforward.

Example 4: Assume that we are given the following 3-bit sequence consisting of 17 non-distinct vectors: (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}) = (001, 100, 001, 110, 111, 111, 101, 110, 011, 000, 101, 001,

100, 000, 110, 110, 011); our objective is to compact this sequence with a compaction ratio of 2.

We start building the Markov model that characterizes the initial sequence. For clarity, the construction of the tree DMT_1 is shown in Figs.12-13 for two different scenarios. First, in Fig.12, we assume that the parameter *model_size* is set by the user to the value 35; this means that the model can be grown dynamically (without any need for flushing) until this limit is reached.

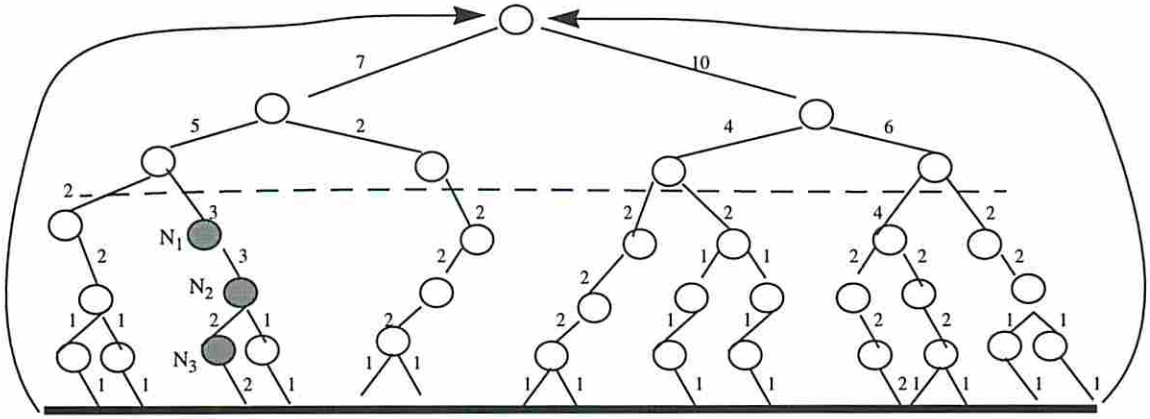

Fig.12

Once we built the Markov tree in Fig.12, we start the procedure *generate_seq* with parameter *ratio* = 2 and generate a subset of 8 vectors which best approximate the original sequence. To this effect, we use a modified version of the *dynamic weighted selection algorithm* [18]. In that approach, a similar structure with DMT_0 is built; more precisely, a full tree having on the leaves the symbols that need to be generated. The counts on the edges are dynamically updated and the symbols are generated according to their probability distribution. For this, a single random number generator is required in order to divide the interval [0,1] into subintervals that correspond to symbols probabilities. At each level, the random number is compared to the left probability: if lower, a zero value is generated; if greater, a one value is generated and the number is decreased by the left probability. In our case, the upper tree is simultaneously parsed from the root to the leaves according to the bits generated in the lower subtree. This way we ensure that all pairs of vectors $((v_1, v_2), (v_2, v_3),...)$ are used to build the model. The procedure is then resumed until the needed number of vectors is generated.

In the second scenario, illustrated in Fig.13, the *model_size* parameter is set by the user as being 30 therefore the tree in Scenario 1 cannot be grown as such because the limit of 30 nodes is reached before the whole sequence is scanned. As a consequence, once we reach this limit (this actually happens immediately after processing the subsequence $v_1, v_2,..., v_9$), we stop growing the tree and call *generate_seq* procedure with parameter *ratio* = 2 (Fig.13a). This will produce a subsequence of 4 vectors which best approximate the first 'segment' $(v_1, v_2,..., v_9)$ of the original sequence. After that we flush the model (keeping only the very last processed vector v_9) and start a new Markov tree as shown in Fig.13b. When the whole sequence is exhausted, based on this new Markov tree, we generate a new subset of 4 vectors which best approximate the second 'segment' $(v_{10}, v_{11},..., v_{17})$ of the original sequence.

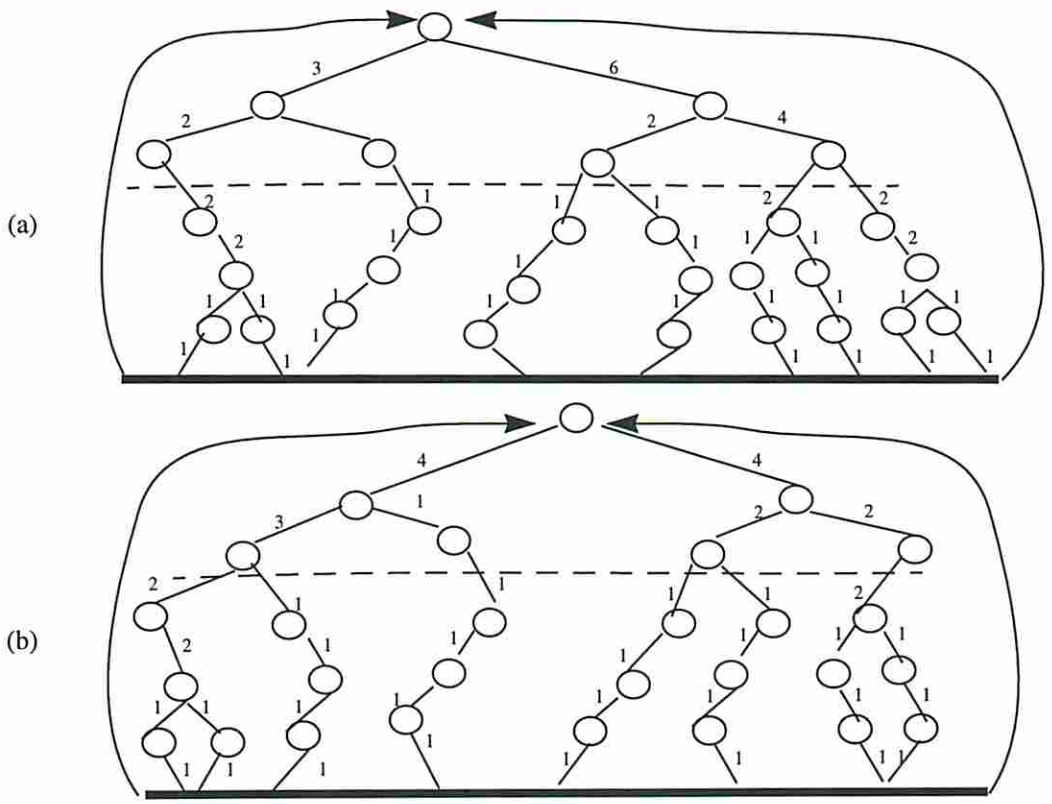

Fig.13

In general, by alternating the generation and flush phases in the DMC procedure, the complexity of the model can be effectively handled. The issue of accuracy in the context of these repeated flushes is discussed in the subsequent section.

The above discussion can be easily extrapolated for a two-level hierarchical model. The only remaining issue is to determine how many vectors must be generated inside each macrostate before a transition to another macrostate is performed. In general, if a subsequence of length L_i is assigned to macrostate S_i, after compaction with ratio r has to be reduced to L_i/r. We note that inside all macrostates the *same compaction ratio* should be used, otherwise the composition of the sequence (as far as power consumption is concerned) may be totally different than that of the initial one. On average, each macrostate S_i should be visited $(p(S_i) \cdot M)$ times where M is the length of the 'macrosequence' (i.e. the length of the initial sequence of macrostates). Thus, each time a macrostate S_i is visited we need to generate a number of $\left(\frac{L_i}{r \cdot p(S_i) \cdot M}\right)$ vectors. Since we do compaction only at the microstate level, the length of the macro-sequence is preserved (the generation procedure stops when M macrostates are obtained).

5. Practical considerations

5.1 Complexity related issues

The DMC modeling approach offers the significant advantage of being a *highly adaptive technique*. Starting with an initial empty tree DMT_1, while the input sequence is scanned incrementally, both the set of states and the transition probabilities change dynamically making this technique highly adaptive.

Input sequences having a large number of bits k are very common in practice; the success of DMC models for sequence compaction when k is large is based on two key observations:

• The larger the value of k is, the sparser the structure of DMT_1 will be.

To motivate this, assume a finite input sequence of length n ($n \ll 2^k$). Intuitively, in a worst-case scenario when DMT_1 is completely skewed (that is, all vectors are distinct), DMT_1 will have a number of nodes proportional to $2nk$ (in all other cases, due to the sharing of paths among nondistinct vectors, the number of nodes will be smaller). On the other hand, the corresponding full tree (statically constructed) with the same depth, will have a number of nodes proportional with 2^{2k}. Therefore the sparsity of the tree DMT_1 (compared to the corresponding full tree) will increase with k as: $Sparsity \propto \frac{2nk}{2^{2k}}$. Assuming for instance an input sequence on 60 bits having a length of 100,000 vectors, then the sparsity of DMT_1 is about 10^{-29}. The DMC modeling technique exploits this observation by starting with an initially empty model and dynamically growing the Markov tree that characterizes the input sequence. By doing so, one can expect to build much smaller trees than the ones otherwise obtained by using a static model based on an initial full tree. Indeed, in practice the dynamic growing of the Markov model performs very well and the experimental results presented in the next section will support this claim.

• Biased sequences which usually occurs in practice as candidates for power estimation, contain a relatively small number of distinct patterns which arise in many different contexts in the whole sequence therefore a probabilistic model is ideally suited for modeling them.

We point out that both these observations can be efficiently exploited only by a probabilistic technique such as DMC modeling; a deterministic technique (e.g. [12]) has no such inherent capability and therefore cannot avoid all the difficulties that arise from this type of complexity.

However, a natural question still remains: when should this growing process be halted? If it is not halted, there is no bound on the amount of memory needed. On the other side, if it is completely halted we lose the ability to adapt if some characteristics of the source message change. A practical solution is to set a limit on the number of states in the DMC [16] as we actually did in Example 4. When this limit is reached, the Markov model is flushed and a new model is started. Although this solution may appear as too drastic, in practice it performs very well. The intuition behind this property is the capability of DMC model to adapt very fast to changes that occur while the input is scanned. A less extreme solution to limit model growing is also possible; we can keep a backup buffer that retains the last p vectors emitted by the source and whenever the model should be discarded, we may reuse this information to avoid starting the new model from the scratch.

5.2 Accuracy related issues

To see how the flushing technique affects the accuracy, let's assume that an input sequence of length n is modeled by the *DMC* approach. Suppose that during the building of the Markov model, flushing occurs after the first n_1

vectors, then after the next n_2 vectors, and so on. If the number of flushes is f, then $n_1 + n_2 + ... + n_f = n$ and the following result holds:

Theorem 5.

The error ε in estimating transition probabilities with a model that supports flushing satisfies:

$$\varepsilon = \frac{\sum_{i=1}^{f} n_i \cdot \varepsilon_i}{n} \leq max(\varepsilon_i) \tag{5}$$

where ε_i, n_i are the error and the number of vectors processed when the sequence between the i-th and $(i+1)$-th flushes is generated.

Therefore, as long as the partial DMC models accurately the transition probabilities for the initial subsequences, the transition probabilities for the entire sequence are preserved up to some ε. Differently stated, we do not have to worry about the number of flushes needed to manage complexity if the individual Markov models capture accurately the characteristics of the subsequences.

6. Experimental results

The overall strategy is depicted in Fig.14.

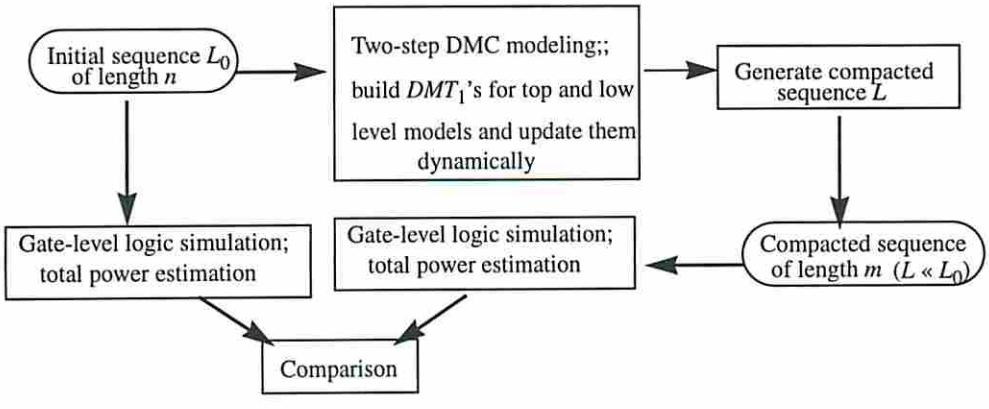

Fig.14

Basically, we verified our ability to compact large input sequences which may also be used as power benchmarks in the design process. We assume that the input data is given in the form of a sequence of binary vectors. While this is a valid assumption at the logic level, it requires some justification at the architectural level. An instruction stream at the architectural level can be converted to a binary stream using the information about opcodes and dynamic instruction traces that resolve memory references and ambiguities. The obtained binary stream can be then subjected to any compaction technique developed for bit-level specifications.

Starting with an k-bit input sequence of length n, we perform a one-pass traversal of the original sequence to assign microstates to macrostates. Next, we simultaneously build the trees DMT_1 for the entire hierarchy (macro- and microstates); during this process, the frequency counts on DMT_1's edges are dynamically updated.

The next step in Fig.14 does the actual generation of the output sequence (of length m). As explained in Section 4, to generate the new sequence we use a modified version of the dynamic weighted selection algorithm presented in [18]. If the initial sequence has the length n and the new generated sequence has the length $m < n$ then the outcome of

this process is a compacted sequence, equivalent to the initial one as far as total power consumption is concerned; we say that a *compaction ratio* of $r = n/m$ was achieved.

Finally, a validation step is included in the strategy; we resorted to an in-house gate-level logic simulator developed under SIS. The total power consumption of some ISCAS'85 benchmarks has been measured for the initial and the compacted sequences, making it possible to assess the effectiveness of the compaction procedure (under both zero- and real-delay models).

In Table 1, we provide only the real-delay results for a set of highly biased sequences (of length 4,000) which contains three modes of operation: a high activity sequence duplicated 4 times, followed by a low activity sequence, and finally by a pseudorandom one. As shown in Table 1, the initial sequences were compacted with three different compaction ratios (namely $r = 2$, 5 and 10); we give in this table the total power dissipation measured for the initial sequence (column 3) and for the compacted sequence (columns 4-9). The time in seconds (on a Sparc 20 workstation with 64 Mbytes of memory) necessary to read and compress data with DMC modeling was below 5 seconds in all cases, either for the flat or for the hierarchical model. Since the compaction with DMC modeling is linear in the number of nodes in the structure DMT_1, this value is far less than the actual time needed to simulate the whole sequence. During these experiments, the number of states allowed in the Markov model was 20,000 on average.

As we can see, the quality of results is very good even when the length of the initial sequence is reduced by one order of magnitude. Thus, for C432 in Table 1, instead of simulating 4,000 vectors with an exact power of 1810.02 uW, one can use only 800 vectors with an estimate of 1888.42uW or just 400 vectors with a power consumption estimated as 1906.84 uW. This reduction in the sequence length has a significant impact on speeding-up the simulative approaches where the running time is proportional to the length of the sequence which must be simulated. On the other hand, if one use the flat model, for the same benchmark the relative errors in power prediction are 18% and 32%, respectively. The main reason for this inaccuracy is the lack of adaptability which characterizes the flat model when is applied to multi-modal sequences (this is not true for 'well-behaved' sequences, that is uni-modal sequences, for which the flat model performs very well).

Table 1: Total power consumption for ISCAS'85 circuits

			Flat model			Hierarchical model		
Circuit	Number of inputs	Exact power	$r = 2$	$r = 5$	$r = 10$	$r = 2$	$r = 5$	$r = 10$
C432	36	1810.02	1473.30	1491.58	1230.77	1875.22	1888.42	1906.84
C499	41	4390.79	3931.30	5341.74	6126.58	4477.09	4497.01	4591.46
C880	60	3788.22	4337.18	4504.40	2803.14	3921.61	3851.42	4006.19
C1355	41	3783.35	4300.08	3065.71	4333.81	3828.45	3910.45	3933.25
C1908	33	6352.07	4947.08	4565.87	7094.39	6127.82	6145.49	6493.44
C3540	50	14471.46	17153.07	9005.19	3527.65	14797.30	15056.43	15021.08
C6288	32	104158.25	86586.18	81100.59	82652.47	101407.12	98112.01	96295.36
		Avg.% err.	16.40	23.64	31.45	2.67	3.55	4.74

Finally, we compare our results generated by HMM with simple random sampling of vector pairs from the original sequences [19]. In simple random sampling, we performed 1,000 simulation runs with 0.99 confidence level and 5% error level on each circuit[1]. We report in Table 2 the maximum and average number of vector pairs needed for total power values to converge [11]. We also indicate the percentage of error violations for total power values, using

1. This means that the probability of having a relative error larger than 5% is only 1%.

as thresholds 5%, 6% and 10%. Using different seeds for the random number generator (and therefore choosing different initial states in the sequence generation phase), we run a set of 1,000 experiments for the HMM technique. In Table 3, we give the results obtained with the hierarchical model, for the same thresholds as those used in simple random sampling.

Table 2: Results obtained for Simple Random Sampling

	Number of vector pairs		Error violations (%)		
Circuit	Max.	Avg.	> 5%	> 6%	>10%
C432	3300	2176	1.1	0.7	0.4
C499	1500	862	1.4	1.3	0.2
C880	3990	2705	1.8	0.4	0.7
C1355	1380	814	1.7	1.0	0.2
C1908	1620	846	1.9	1.3	0.2
C3540	2340	1446	2.0	1.3	0.4
C6288	7470	5422	1.4	1.4	0.3

Table 3: Results obtained for HMM Approach

		Error violations (%)		
Circuit	Number of vectors	> 5%	> 6%	>10%
C432	800	2.6	0.6	0.0
C499	800	0.1	0.0	0.0
C880	800	2.8	0.9	0.0
C1355	800	0.1	0.0	0.0
C1908	800	0.1	0.0	0.0
C3540	1200	1.6	0.3	0.0
C6288	3600	1.5	0.0	0.0

Once again, the results obtained with HMM modeling technique score very well and prove the robustness of the present approach. As we can see, using fewer vectors, the accuracy of HMM is higher than the one of simple random sampling in most of the cases. Noteworthy examples are benchmarks C499, C1908, C6288 where less than 60% of the maximal number of vector pairs needed in random sampling for convergence are sufficient for HMM to achieve higher accuracy and confidence levels.

Conclusion

In this paper, we addressed the vector compaction problem from a probabilistic point of view. Structuring the input space as a two-level hierarchy, we proposed an original approach to compact an initial sequence into a much shorter equivalent one, which can be used after that with any available simulator to derive power estimates in the target circuit.

A major contribution of this paper is that it introduces the hierarchical modeling of Markov chains as a flexible framework for capturing not only complex spatiotemporal correlations, but also the dynamic changes in the sequence characteristics such as different circuit operating modes or varying power distributions. The results obtained on standard benchmarks, show that using this hierarchical model, large compaction ratios can be obtained without much loss in accuracy in total power estimates.

The issues brought into attention represent an important step to reduce the gap between simulative and nonsimulative techniques which are currently the norm.

References

[1] S.M.Kang, 'Accurate Simulation of Power Dissipation in VLSI Circuits', in *IEEE Journal of Solid State Circuits,* 21 (5), pp. 889-891, Oct.1986.

[2] C.X.Huang, B.Zhang, A.-C.Deng, and B.Swirski, 'The Design and Implementation of PowerMill', in *Proc. Intl. Workshop on Low Power Design,* pp. 105-110, April 1995.

[3] B.J.George, D.Gossain, S.C.Tyler, M.G.Wloka, and G.K.Yeap, 'Power Analysis and Characterization for Semi-Custom Design', in *Proc. Intl. Workshop on Low Power Design,* pp.215-218, April 1994.

[4] F.N. Najm, 'A Monte Carlo Approach for Power Estimation', *IEEE Transactions on VLSI Systems*, Vol.1, No.1, pp. 63-71, Mar.1993.

[5] A. Ghosh, S. Devadas, K. Keutzer, and J. White, 'Estimation of Average Switching Activity in Combinational and Sequential Circuits', in *Proc. ACM/IEEE Design Automation Conference,* pp. 253-259, June 1992.

[6] F. N. Najm, 'Transition Density: A New Measure of Activity in Digital Circuits', *IEEE Transactions on CAD*, Vol. 12, No.2, pp. 310-323, Feb.1993.

[7] C.-Y. Tsui, M. Pedram, and A. M. Despain, 'Efficient Estimation of Dynamic Power Dissipation with an Application', in *Proc. IEEE/ACM Intl. Conference on Computer Aided Design,* pp. 224-228, Nov. 1993.

[8] A. Chandrakasan, et. al, 'HYPER-LP: A System for Power Minimization Using Architectural Transformation', in *Proc. IEEE/ACM Intl. Conference on Computer Aided Design,* pp. 300-303, Nov.1992.

[9] P. Landman, J. Rabaey, 'Power Estimation for High Level Synthesis', in *Proc. European Design Automation Conference*, pp. 361-366, Feb.1993.

[10] M. Nemani and F. Najm, 'ITowards A High-Level Power Estimation Capability', in *IEEE trans. on Computer-Aided Design of Integrated Circuits,* vol. 15, No. 6, June 1996.

[11] J. Rabaey, and M. Pedram, 'Low Power Design Methodologies', Kluwer Academic Publishers, 1996.

[12] J. Monteiro and S. Devadas, 'Techniques for Power Estimation of Sequential Logic Circuits Under User-Specified Input Sequences and Programs', in *Proc. Intl. Workshop on Low Power Design,* pp. 33-38, April 1994.

[13] D. Marculescu, R. Marculescu, and M. Pedram, 'Stochastic Sequential Machine Syntesis Targeting Constrained Sequence Generation', in *Proc. ACM/IEEE Design Automation Conference*, pp. 696-701, June 1996.

[14] C.-Y. Tsui, R. Marculescu, D. Marculescu, and M. Pedram, 'Improving the Efficiency of Power Simulators by Input Vector Compaction', in *Proc. ACM/IEEE Design Automation Conference*, pp. 165-168, June 1996.

[15] T. Bell, J. Cleary and I. Witten, 'Text Compression', Prentice Hall, 1990

[16] G.V.Cormack and R.N.Horspool, 'Data Compression Using Dynamic Markov Modelling', in *Computer Journal*, Vol. 30, No. 6, pp. 541-550, 1987.

[17] A. Papoulis, 'Probability, Random Variables, and Stochastic Processes', McGraw-Hill Co., 1984.

[18] J.W.Green and K.J.Supowit, 'Simulated Annealing without Rejected Moves', in *Digest. of Intl. Conference on Computer Design,* pp. 658-663, Oct. 1984

[19] I.R. Miller, J.E. Freund and R. Johnson, 'Probability and Statistics for Engineers', Prentice Hall, 1990.

[20] S.-Y. Huang, K.-C. Chen, K.-T. Cheng, and T.-C. Lee, 'Compact Vector Generation for Accurate Power Simulation', *Proc. ACM/IEEE Design Automation Conference,* pp. 161-164, June 1996.