Stochastic Sequential Machines Synthesis with Application to Constrained Sequence Generation

Diana Marculescu
Department of Electrical Engineering
University of Maryland, College Park, MD 20742
and
Radu Marculescu
Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis, MN 55455
and
Massoud Pedram
Department of Electrical Engineering-Systems
University of Southern California, Los Angeles, CA 90089

Categories and Subject Descriptors:

General Terms:

Additional Key Words and Phrases:

In power estimation, one is faced with two problems: 1) generating input vector sequences that satisfy a given statistical behavior (in terms of signal probabilities and correlations among bits); 2) making these sequences as short as possible so as to improve the efficiency of power simulators. Stochastic sequential machines (SSMs) can be used to solve both problems. In particular, this paper presents a general procedure for SSM synthesis and describes a new framework for sequence characterization to match designer's needs for sequence generation or compaction. Experimental results demonstrate that compaction ratios of 1-3 orders of magnitude can be obtained without much loss in accuracy of total power estimates.

1. Introduction

In the past, time, area and testability were the primary concerns of IC designers. With the growing need for low-power electronic circuits and systems, power estimation and low-power optimization have become crucial tasks that must be also addressed. It is expected that, in the forthcoming years, power issues will receive increasing attention due to the widespread use of portable applications and desire to reduce the packaging and cooling costs of high-end systems.

Power estimation techniques must be fast and accurate in order to be applicable in practice. Not surprisingly, these two requirements interfere with one another and at some point they become contradictory. General simulation techniques can provide sufficient accuracy, but the price to be payed is too high; one can extract switching activity information (and thus power estimates) by doing exhaustive simulation on small circuits, but it is unrealistic to rely on simulation results for large circuits. Probabilistic power estimation techniques were thus developed and proved their usefulness by providing sufficient accuracy with low computational overhead [1].

A major challenge in probabilistic power estimation approaches is the ability to account for internal dependencies due to the reconvergent fan-out in the circuit. This problem, which we will refer as '*the circuit problem*', is by no means trivial. Indeed, a whole set of solutions have been proposed, ranging from approaches which build the global OBDDs [2] and therefore capture all internal dependencies, to efficient techniques which partially account for dependencies in an incremental manner [3]-[5].

Recently, some authors have pointed out the importance of correlations not only inside the target circuit, but also at the circuit inputs. If one ignores these correlations, the power estimation results can be significantly impaired [6][7]. We will refer to this issue as '*the input problem*' and note that it is important not only in power estimation, but also in low-power design.

Let us consider a simple example to illustrate the significance of the input problem. Suppose that a 4-bit ripple-carry adder is fed successively by two input sequences S_1 and S_2, as shown in Fig.1a. To estimate the total power consumption of the circuit (in a gate level implementation), we may sum over all the gates in the circuit the average power dissipation due to the capacitive switching currents, that is: $P_{avg} = \frac{f_{clk}}{2} \cdot V_{DD}^2 \cdot \sum_n (C_n \cdot sw_n)$ 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. As we can see, the average switching activity per node (gate) is a key parameter that needs to be determined correctly.

FIGURE 1. The input problem

The two sequences in Fig.1a, have the same signal probability on the input lines (p = 0.5), but are otherwise very different; whilst the average switching activity per bit is 0.5 for

sequence S_1, it is 0.3 for sequence S_2. This difference in switching activity leads to: P_{S1} = 456 μW and P_{S2} = 365 μW at 20 MHz that is, about 20% difference in the values of total power consumption. On the other hand, sequences S_1 and S_3 given in Fig.1b have different lengths (S_3 is about 40% shorter than S_1) but similar statistics on the bit lines. This similarity leads to P_{S3} = 446 μW, a value which is only 2% off from P_{S1}. Since the input sequence plays such an important role in determining the average power dissipation of a circuit, the question becomes how one should select or generate the sequence of input vectors to be applied to the circuit under consideration and still get the same power values.

In many cases, the designer has some information (albeit, limited) about the statistics of the input sequence (in terms of signal or transition probabilities, inter-bit correlations, etc.). Generating a minimal-length sequence of input vectors that satisfies these statistics in not trivial. More precisely, LFSRs which have traditionally found use in testing or functional verification [8], are of little or no help here. The reason is that the set of input statistics which must be preserved or reproduced during sequence generation (for use with power simulators) is quite complex. One such attempt is [9] where authors use deterministic FSMs to model user-specified input sequences. Since the number of states in the FSM is equal to the length of the sequence to be modeled, the ability to characterize anything else but short input sequences is limited.

From a designer perspective, we may want to estimate the power consumption of the adder in Fig.1 in a context resembling as much as possible the one where this adder will be instantiated. For example, if the adder was designed to be an incrementor in the address calculation unit of a memory chip, then the primary inputs A and B will likely receive sequences similar to S_2; on the other hand, if this adder was designed to be part of a DSP system used for noise analysis, then it will likely receive random inputs and therefore S_1 is the most appropriate sequence to be used for power estimation.

To validate the design, circuit or gate-level simulation is finally invoked to measure the total power consumption. The biggest hurdle for simulation-based power estimators is the huge number of vectors which should be applied to the circuit to obtain accurate power values for the circuit. It is impractical to simulate large circuits using millions or even thousands of input vectors and therefore, the length of the sequence to be simulated is an important consideration.

In summary, a number of issues appear to be important for power estimation and low-power synthesis. The *input statistics* which must be properly captured and the *length of the input sequences* which must be applied, are two such issues. From this perspective, the present paper shifts the focus from ‘the circuit problem’ to ‘the input problem’ and improves the state-of-the-art by proposing an original solution for *constrained sequence generation*.[1]

Over the years, many important problems in sequential circuit synthesis and optimization have been approached using concepts from automata theory. Finite automata

---

1. Simply stated, any input sequence that must satisfy a set of spatial and/or temporal correlations is considered to be “constrained”.

are mathematical models for systems with a finite number of states which accept, at discrete time steps, certain inputs and emit accordingly certain outputs. Finite automata exhibit deterministic behavior, that is, the current state of the machine and the input value determine the next state and the output of the automaton. It is quite natural (and useful) to consider automata with stochastic behavior. The idea is that the automaton, when in state s_i and receiving input x, can move into any new state s_j with a positive probability $p(s_i,x)$. A practical motivation for considering probabilistic automata is that even sequential circuits which are intended to behave deterministically, may exhibit stochastic behavior because of random malfunctioning of components [10][17].

The mathematical foundation of our approach relies on the *stochastic sequential machines* (SSMs) theory and, without any loss in generality, emphasizes those aspects related to Moore-type machines. In this paper, we reveal a general procedure for SSM synthesis and describe a new framework for sequence characterization to match designer's needs for sequence generation or compaction. We focus on the basic task of synthesizing an SSM which is able to generate constrained input sequences. Such a machine can be effectively used in power estimation as it is illustrated in Fig.2.

FIGURE 2. Constrained sequence generation

To evaluate the total power consumption in a target circuit for a given input sequence of length L_0, we first derive a probabilistic model based on SSMs and then, having this compact representation, we generate a much shorter sequence L, equivalent with L_0 as far as total power consumption is concerned, which can be used with any available simulator to derive accurate power estimates.

We note that successful research on constrained sequence generation, may find application ground in three areas:

- *Sequence generation:* This represents the ability to generate input sequences, with different lengths, that satisfy a set of user-prescribed characteristics in terms of word-level transition or conditional probabilities. Basically, for a given set of input symbols $\{\alpha_1, \alpha_2,..., \alpha_n\}$ with the set of occurrence probabilities $\{p_1, p_2,..., p_n\}$, we are interested in finding a machine capable not only of generating those symbols with the specified set of probabilities, but also of preserving the *temporal order* (*sequencing*) among them. As illustrated in Fig.1, this is important especially in low-power design. This issue will be discussed in detail in Section 2.

- *Sequence compaction:* This is basically the ability to construct a *representative sequence (*short enough to be efficiently simulated) equivalent as far as the total power consumption is concerned with the original sequence that reproduces the operating context in which the circuit is intended to work. This feature can make (circuit or gate-level) simulators a viable option for power analysis even for very large circuits and therefore deserves special attention; it is discussed and supported with examples in Sections 3 and 4.

• *Probability transformation:* This represents the ability to construct machines that convert a given set of input symbols, occurring with some fixed probabilities, into another one, which may have a completely different set of probabilities. Using the aforementioned formalism, a probability transformer when fed with input symbols $\{\alpha_1, \alpha_2,..., \alpha_n\}$ which occur with probabilities $\{p_1, p_2,...,p_n\}$, generates a new set of symbols $\{\beta_1, \beta_2,..., \beta_m\}$ with different, yet prescribed, probabilities $\{q_1, q_2,..., q_m\}$. A probability transformer may take as input an uncorrelated (random) binary stream and produce a highly correlated binary stream as output [21][22]. Such a probability transformer, when placed in front of the circuit under consideration, enables the use of any probabilistic power estimation approach that assumes input temporal independence as it eliminates the need for considering explicit input correlations. The input correlations will be instead captured by the structure of the probability transformer circuitry. As we shall see later, the SSM is such a probability transformer which, when excited with randomly generated inputs, produces temporally correlated data at the primary inputs of the target circuit. Thus, the product machine (*input SSM, target circuit*) (see Fig.2) can now be analyzed using probabilistic power estimation techniques such as [23].

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 a first step toward reducing the gap between the simulative and probabilistic techniques commonly used in power estimation. Finally, the concept of SSMs may find useful applications in other CAD-related problems.

The paper is organized as follows: Section 2 introduces some basic definitions from SSM theory and gives the main decomposition theorems used in SSM synthesis. Section 3 discusses the constrained sequence generation problem while Section 4 gives a practical procedure for sequence compaction. Section 5 is devoted to practical considerations and experimental results. Finally, we conclude by summarizing our main contribution.

2. Synthesis of SSMs

In this section, we review the concept of SSM and describe a basic procedure for synthesizing SSMs from their mathematical models. In what follows, we use the formalism and notations introduced in [17].

2.1 Stochastic machines: basic definitions

Definition 1: A Mealy-type SSM is a quadruple $M = (S, X, Y, \{A(x, y)\})$ where S, X, and Y are finite sets (the internal states, inputs, and outputs respectively), and $\{A(x, y)\}$ is a finite set containing $|X| \times |Y|$ square stochastic matrices of order $|S|$ such that $a_{ij}(y|x) \geq 0$ for all i and j, and

$$\sum_{y \in Y} \sum_{j=1}^{|S|} a_{ij}(y|x) = 1 \qquad \text{where} \qquad A(y|x) = [a_{ij}(y|x)] \quad (1)$$

Interpretation: Let π be any $|S|$-dimensional vector. If the machine begins with an initial distribution π over the state set S and is fed with a word x, it outputs the word y and moves on to the next state. The transition is controlled by the *transition matrices* $A(y|x)$ where $a_{ij}(y|x)$ is the *conditional probability* of the machine going to state s_j and producing the symbol y, given it had been in state s_i and fed with symbol x.

Definition 2: Let M be an SSM, $u = x_1x_2...x_k$ an input sequence and $v = y_1y_2...y_k$ an output sequence. By definition, $A(v|u) = [a_{ij}(v|u)] = A(y_1|x_1)\cdot A(y_2|x_2)\cdot...\cdot A(y_k|x_k)$; it follows from the interpretation of the values of $a_{ij}(y|x)$ that $a_{ij}(v|u)$ is the probability of machine going to state s_j and producing the sequence v, having been in state s_i and fed sequentially the sequence u.

Definition 3: A Moore-type SSM is a quintuple $M = (S, X, Y, \{A(x)\}, \Lambda)$ where S, X, and Y are as in Definition 1, $\{A(x)\}$ is a finite set containing $|X|$ square stochastic matrices of order $|S|$ and Λ a deterministic function from S into Y.

Interpretation: The value $a_{ij}(x)$ ($A(x) = [a_{ij}(x)]$) is the probability of the machine moving from state s_i to s_j when fed with the symbol x. When entering state s_j, the machine outputs the symbol $\Lambda(s_j) \in Y$.

Definition 4: Let M be an SSM. Let $A(u) = [a_{ij}(u)] = A(x_1)\cdot A(x_2)\cdot...\cdot A(x_k)$; it follows from the above interpretation that $a_{ij}(u)$ is the probability of the machine going from state s_i to state s_j when fed the word u. The output word v depends on the sequence of states through the machine passed when scanning the input word u.

As we can see from the above definitions, Mealy and Moore stochastic machines generalize the corresponding definitions of deterministic machines. Since the stochastic machines are more elaborate in structure than the deterministic ones, other generalizations are possible [17]. On this line, we should note that Mealy-Moore equivalence is still valid for stochastic machines, that is every Moore-type SSM has a Mealy-type equivalent and vice versa.

2.2 The synthesis procedure

Without loss of generality, in what follows the machines are assumed to be of Moore-type. The objective of this section is to build a SSM which generates an output sequence with given characteristics. The basic procedure involves synthesis of combinational circuits and construction of information sources with prescribed probability distributions. It can be simplified by means of the following important result:

Theorem 1 [11]: Any $m \times n$ stochastic matrix A can be expressed in the form $A = \sum p_i U_i$ where $p_i > 0$, $\sum p_i = 1$, and U_i are degenerate stochastic matrices (that is, matrix elements are 0 or 1 only), and the number of matrices U_i in the expansion is *at most* $m \cdot (n-1) + 1$.

Proof: p_1 is taken to be $\min_i \max_j [a_{ij}]$ and elements of U_1 satisfy $u^1_{ij} = 1$ if $a_{ij} = max_k[a_{ik}]$ and 0 otherwise. The procedure is then applied recursively to the newly constructed stochastic matrix $[1/(1-p_1)]\ [A-p_1U_1]$. ■

The theorem we provide in the following is a very important result from a practical point of view; as we will see later, it gives the basis to efficiently apply Theorem 1 on large matrices which may arise in practice.

Theorem 2: The sequence $\{p_i\}_{i \geq 1}$ is monotonically non-increasing and strictly positive.

Proof: It suffices to show that $p_1 \geq p_2 > 0$ due to the recursive manner in which matrices U_i are generated. According to the definition, $p_1 = \min_i \max_j [a_{ij}]$ and $p_2 = (1-p_1)q_2$ where $q_2 = \min_i \max_j [a^1_{ij}]$ ($A_1 = [a^1_{ij}]$). Since $A_1 = [1/(1-p_1)]\ [A-p_1U_1]$, the inequality becomes $\min_i \max_j [a_{ij}] \geq \min_i \max_j [a_{ij}-p_1u^1_{ij}]$ where $U_1 = [u^1_{ij}]$. But for any fixed i, j, $a_{ij} \geq a_{ij}-p_1u^1_{ij}$ (the elements of matrix U_1 are either 1 or 0 and p_1 is positive); hence $\max_j [a_{ij}] \geq \max_j [a_{ij}-p_1u^1_{ij}]$ for any fixed i and $\min_i \max_j [a_{ij}] \geq \min_i \max_j [a_{ij}-p_1u^1_{ij}]$ thus concluding our proof. ■

Let A be a stochastic matrix which can be expressed in the form $A = \sum p_i U_i$ (i = 1, 2,..., t) according to the above result. This means that either A has been decomposed using exactly t matrices U_i, or considering only the first t matrices in this decomposition has been satisfactory for a given level of accuracy. We allow therefore limited precision in our calculations, not only because this limitation is sufficient in practice, but also because it may substantially simplify the decomposition process based on Theorem 1 (see Section 4).

Let $\Sigma = \{\sigma_1,\sigma_2,...,\sigma_t\}$ be an auxiliary alphabet with t symbols, one for each matrix U_i in the expansion of A, and let P be a single information source over Σ emitting the σ_i with probability p_i. We give in Fig.3 a simplified block diagram of the network which synthesizes such a machine.

FIGURE 3. The general structure of the SSM

The combinational logic is constructed such that its output is s_j for input (x_l, σ_m, s_k), if and only if the entry of matrix U_m in the row corresponding to (s_k, x_l) and the column corresponding to s_j equals 1. The output logic box is a combinational logic implementing the function Λ.

Interpretation: Theorem 1 states in fact that any SSM can be decomposed into a finite number of deterministic sequential machines. The behavior of the SSM is thus "simulated" by selecting one of these deterministic machines based on the values of the auxiliary inputs.

Example 1: Let $M = (S, X, Y, \{A(x)\}, \Lambda)$ with $S = \{0, 1\} = X = Y$, $\Lambda(0) = 1$, $\Lambda(1) = 0$, and $A = \begin{bmatrix} A(0) \\ A(1) \end{bmatrix}$ with $A(0) = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{3}{4} \end{bmatrix}$ and $A(1) = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$. Applying Theorem 1, we get

$$A = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} + \frac{1}{4}\begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} + \frac{1}{6}\begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} + \frac{1}{12}\begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

and thus $\Sigma = \{\sigma_1,\sigma_2,\sigma_3,\sigma_4\}$ and $P = \{p(\sigma_1), p(\sigma_2), p(\sigma_3), p(\sigma_4)\} = (1/2, 1/4, 1/6, 1/12)$. Encoding the symbols in Σ with 2 bits (w_1,w_2) as 00, 01, 10, 11 respectively, we get the following transition table.

FIGURE 4. The behavior of the SSM in Example 1

Using standard Karnaugh approach, we obtain the circuit which synthesizes the given SSM, as shown in Fig.5.

FIGURE 5. A possible implementation of the SSM in Example 1

Words Σ on the auxiliary input (w_1,w_2) must be supplied with the probability distribution P as resulted from the decomposition of matrix A. For this purpose, it is sufficient to generate a set of numbers uniformly distributed on the interval [0, 1] and divide the interval into four subintervals as follows: [0, 1/2), [1/2, 1/2 + 1/4), [1/2 + 1/4, 1/2 + 1/4 + 1/6) and [1/2 + 1/4 + 1/6, 1]. Each subinterval corresponds to a particular word (w_1,w_2) on the auxiliary input: if the number generated lies in some subinterval, the corresponding word is generated. Clearly, this procedure will generate all auxiliary inputs according to the given probability distribution.

In summary, the basic synthesis procedure based on Theorem 1, involves essentially the synthesis of a combinational circuit with feedback, and construction of information sources with prescribed probability distributions.

3. Constrained Sequence Characterization

In this section, we give a precise characterization of sequences in terms of their transition matrices. In addition, we present some theoretical results that demonstrate the possibility of using different input sequences while still having the same total power consumption in the target circuit.

3.1 Sequence equivalence

In what follows, we associate with every Moore SSM its output sequence (of length $L \geq 1$), generated during its normal operation, and we will interchangeably refer to both SSM and its output sequence. The general problem of equivalence between stochastic machines is very complex and for an in-depth presentation the reader is referred to [17]. For practical purposes, we restrict our attention only to reduced stochastic machines of Moore-type.

Definition 5: Two reduced stochastic machines M and M^* (as in Definition 3) are *output-equivalent* if the following conditions are satisfied:

1) The state spaces S and S^* have the same cardinality, that is $|S| = |S^*|$;

2) The output spaces Y and Y^* are the same, that is $Y = Y^*$;

3) For every state s_i of M there corresponds a state s_j of M^*, and vice versa, such that $\Lambda(s_i) = \Lambda^*(s_j)$ for every input u with $L(u) \geq 1$.

Interpretation: If the isomorphism relationship between state spaces S and S^* is given by the function $h : S \rightarrow S^*$, then we can represent the output-equivalence relationship between machines M and M^* as follows:

FIGURE 6. Output equivalence

Basically, S is isomorphically mapped to S^* such that the output spaces coincide. These considerations translate into a definition for sequence equivalence as follows:

Definition 6: The output sequence Y generated by machine M is ε-*equivalent* with the output sequence Y^* produced by M^* if $\|A - A^*\| < \varepsilon$, where the norm is defined as $\|A\| = max|a_{ij}|$. (We note the particular case $\varepsilon = 0$, when $A = A^*$, which corresponds to exact equivalence).

Differently stated, two output sequences are ε-equivalent if they are generated by SSMs characterized by nearly the same average transition probabilities, that is $|a_{ij}(x) - a_{ij}^*(x)| < \varepsilon$, for any input x. In practice, having a reference sequence produced by M, we need to know how close is the sequence generated by M^* to the original one. To this end, we have to investigate the effect of errors (perturbations) that may appear in A[1] on the statistical behavior of the output sequence generated according to A^*.

1. This may appear as a side effect if, for instance, we apply Theorem 1 with limited precision.

3.2 Perturbation analysis for generated sequences

For our particular application, the SSM to be synthesized has no external inputs (that is, $X = \varnothing$). In addition, we assume that the output function Λ is a one-to-one mapping from S to Y and thus, A represents the stochastic matrix associated to the output sequence, i.e. $a_{ij} = p(v_j | v_i)$, where v_i, v_j are two consecutive vectors. Having a reference sequence, to produce an equivalent one we should preserve the word-level transition probabilities. This essentially becomes the problem of preserving both conditional and state probabilities because $p_{i \to j} = p_i \cdot a_{ij}$, where p_i is the state probability of vector v_i and $p_{i \to j}$ represents the transition probability of going from vector v_i to v_j. Assuming stationarity conditions, if $p = [p_i]$ denotes the state probability vector, then from Chapman-Kolmogorov equations [18] we have $A^T \cdot p = p$. (In other words, p is the eigenvector that corresponds to the eigenvalue $\lambda = 1$ in the general equation $A^T \cdot p = \lambda \cdot p$.)

Based on the Perron-Frobenius theorem [19], it can be proven that every stochastic matrix has 1 as a simple eigenvalue and all other eigenvalues have absolute values less than one. Let us assume that the newly generated sequence is characterized by the matrix $A^* = [a_{ij}^*]$ where $a_{ij}^* = a_{ij} + \varepsilon_{ij}$ (ε_{ij} represents the error introduced by some perturbation of matrix A) and $|\varepsilon_{ij}| < 1$. Because A^* characterizes a sequence of vectors, it is also a stochastic matrix and therefore, it has an eigenvalue $\lambda^* = 1$. What we are interested in is the effect of perturbation of matrix A on the eigenvectors that correspond to the eigenvalue 1.

Theorem 3: For any eigenvector p of A corresponding to the simple eigenvalue $\lambda = 1$, there exists an eigenvector p^* of A^* corresponding to the simple eigenvalue $\lambda^* = 1$, such that $\|p - p^*\| = 0(\varepsilon)$ (read as '*zero of epsilon*'), where $0(\varepsilon)$ is any power series in ε (convergent for sufficiently small ε) having the form $k_1\varepsilon + k_2\varepsilon^2 + \ldots$ ■

This theorem follows from the theory of algebraic functions developed in [19]. Since $\|a_{ij} - a_{ij}^*\| = 0(\varepsilon)$, it is easy to see that:

Corollary 1: If the stochastic matrix A is properly preserved, the transition probabilities for the newly generated sequence are *asymptotically close* to the original ones, that is $\|p_{i \to j} - p_{i \to j}^*\| = 0(\varepsilon)$. ■

We have thus proved that we can asymptotically reproduce an initial sequence by preserving its matrix A. From a practical point of view, it is easy to see what are the implications of the above corollary on total power consumption in a target circuit where the input sequence is approximated by a new one.

Corollary 2: If P and P^* are the values of the total power consumption for two sequences satisfying the conditions in Corollary 1, then we have that $|P^* - P| = 0(\varepsilon)$.■

Differently stated, if the new sequence is asymptotically close to the original one, then the same holds for the corresponding total power values.

4. Constrained Sequence Compaction

In practice, we may want to generate a fixed-length sequence satisfying a certain set of constraints or, more frequently, we may have from simulation a characteristic sequence for a target circuit and want to compact it into a new one by preserving its statistics. The first situation was considered in Section 2.2 for the synthesis of the stochastic machine *M*. In this section, we focus on the second issue by considering the problem of synthesizing a SSM for a given vector sequence. In addition, we provide an exact formulation of the constrained sequence generation problem and propose a block-oriented approximation method for solving it.

4.1 The compaction procedure

Since the SSMs are assumed to be Moore type, the generated output depends only on the sequence of states traversed (and not on the inputs).

Example 2: Assume for the sake of simplicity, that the following short sequence of 20 input vectors (v_1,v_2,...,v_{20}) is representative for some target circuit:

FIGURE 7. An input sequence and its transition graph

In the right side of Fig.7, we have the transition graph corresponding to this sequence. The 'state' nodes are labelled with the values that appear in the initial sequence (decimally encoded and read from top to bottom), while the labels on the edges are conditional probabilities captured by analyzing this sequence. For instance, the word '001' (vector v_1 in the initial sequence) is always followed by the word '100' then we have $a_{14} = 1$ (and correspondingly a directed edge, labelled with probability 1, from vertex 1 to vertex 4) while the word '111' is half of the time followed by '101' and the other half by itself, thus we have $a_{75} = 0.5$ and $a_{77} = 0.5$.

Let *M* be the SSM associated with this sequence; as we can see, $S = \{0, 1, 3, 4, 5, 6, 7\}$ and then $|S| = 7$. Now, we are trying to synthesize a new machine M^*, output-equivalent with *M*, and eventually generate an equivalent (and compacted) sequence with the initial one, using *M**. To make our job easier, let's assume also that $Y = S$ and $Y^* = S^*$ (i.e., *M* and M^* are both Moore-type, and the output spaces coincide with the state spaces).

From the very beginning, just by looking at first two conditions in Definition 5, we may deduce that $|S^*| = 7$ and $Y^* = Y = S = S^*$. The corresponding stochastic matrix for the initial sequence is shown below, along with its decomposition from Theorem 1:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} + 0.5 \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= 0.5 \cdot U_1 + 0.5 \cdot U_2$$

Hence, we need a single auxiliary bit w to distinguish between the two deterministic sequential machines obtained: $w = 0$ specifies the first machine which corresponds to U_1 and $w = 1$ the second machine (both with probability 0.5) which corresponds to U_2. The transition table corresponding to this example is given below:

FIGURE 8. The transition table for the SSM in Example 2

A possible implementation for M^* (with D Flip-Flops) is given in Fig.9. This SSM can now be used as a generator for a 3-bit sequence with the same stochastic characteristics as the original one. Bit w is generated using a random number generator such that 0 and 1 are equally likely (i.e. probability 0.5).

FIGURE 9. The SSM generating the original sequence in Example 2

To generate a sequence, the SSM M^* should be initialized in the most probable state: in our case, either ‘110’, ‘101’ or ‘111’. Using a random number generator for the bit w, we get the following behavior when considering ‘111’ as the initial state:

FIGURE 10. A possible behavior of the SSM

Analyzing the next state bit lines, we get the following stochastic matrix after 10 generated vectors:

$$A^{*} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 \end{bmatrix}$$

Note: we can see that for 10 generated vectors with M^{*}, the initial stochastic characteristics are preserved exactly; indeed from Definition 6 with $\varepsilon = 0$, we have $\|A - A^{*}\| = 0$, therefore, in this case, a compaction ratio of 2 has been achieved without any loss of information.

We should note that using another initial vector (e.g. '110' or '101') as initial state of the circuit in Fig.9, we would have obtained other output sequences, but all of them still satisfying the inequality from Definition 6.

Remark: decomposing the initial matrix A (via Theorem 1), instead of directly generating the compacted sequence from the Markov chain globally characterized by A, has three important advantages:

- first, it allows the hardware synthesis of the machine M;

- second, it drastically reduces the complexity of the generation process: instead of using a separate generator for each individual state, we need only a single, unique, generator for the whole process (that is, in the worst-case, we need only $2 \cdot \log|S|$ instead of $|S| \cdot \log|S|$ bits per generator, where |S| is the number of reachable states in the Markov Chain);

- third, it allows to trade accuracy versus efficiency by keeping only a small subset of matrices U_i from the whole set that would correspond to the exact decomposition. This way, the transition probabilities that should be generated at the auxiliary inputs are more uniformly distributed over the interval [0, 1] and therefore the generation procedure is significantly simplified.

To conclude this section, the following general procedure can be used for sequence generation:

FIGURE 11. The generation procedure

4.2 Complexity issues

In practice, we may have to deal with sequences that have a large number of bits (and bit patterns) which may give rise to large number of states in the SSM. More precisely, the theoretical space complexity of matrix A is $2^n \times 2^n$ (where n is the number of bits of the input sequence) but in practice the number of distinct transitions is far less than this limit.

As a consequence, the sparse matrix representation technique is of real help to handle complexity. However, if the number of states is still too large, the manipulation of matrix *A* becomes prohibitive. To handle such cases, we suggest to apply the above procedure in a *block-oriented* fashion, that is first partition the whole sequence of *n* bits, into *b* smaller groups of at most $\left\lfloor \frac{n}{b} \right\rfloor$ bits, and after that apply the procedure to each block, one at a time. By doing so, we lose some accuracy by ignoring dependencies across the block boundaries, but greatly increase our ability to work with sequences with a large number of bits.

We decide whether a set of bits are in the same block or not, by considering only correlations between pairs of bits. For instance, the set of 4 bits $\{x_1, x_2, x_3, x_4\}$ may be partitioned in two groups of two bits each by looking only at pairwise transition probabilities (e.g. (x_1, x_2), (x_1, x_3),..., (x_3, x_4)). Note that the exact procedure would require analysis of joint transition probabilities of 3 or 4 bits (e.g. (x_1, x_2, x_3), (x_1, x_2, x_4),..., (x_1, x_2, x_3, x_4)) which is exponential in the number of bits. More formally, given a set of bits $\{x_i\}_{1 \le i \le n}$ to be partitioned into *b* groups $G_1, G_2, ..., G_b$, we construct a complete graph on *n* vertices where each vertex corresponds to one of the bits and each edge corresponds to the pairwise correlation between the corresponding bits. The edge weights are defined by: $cost(x, y) = \sum_{i, j, k, l = 0, 1} \left| p(x_{i \to j} y_{k \to l}) - p(x_{i \to j}) \cdot p(y_{k \to l}) \right|$ for every edge (x, y) in the graph. Next, we have to find an assignment of each vertex to some group such that $\sum_{1 \le p < q \le b} \sum_{\substack{x \in G_p \\ y \in G_a}} \alpha(x, y) \cdot cost(x, y)$ is minimized, where $0 \le \alpha(x, y) \le 1$ is a weighting coefficient that represents the correlation between inputs *x* and *y* in the circuit involved in the compaction process. The α coefficients are computed as the inverse of the shortest topological distance (from primary inputs up to the point of reconvergence) between the inputs of the circuit involved in compaction process. If two inputs do not reconverge, their topological distance is considered infinite and they are considered uncorrelated ($\alpha = 0$). The above formulation involving the cost function is in fact a *min-cut partitioning* problem which is NP-complete in the general case [13]. Fortunately, excellent heuristics are available to solve this problem [14][15].

Example 3: Let us consider the input sequence in Fig.7, and let *x*, *y*, *z* be the 3 bits for representing $v_1, v_2, ..., v_{20}$ that feed the simple circuit in Fig.12a. Using the definition of the cost function above, we get *cost* $(x, y) = 0.68$ and similarly for the other two pairs: *cost* $(y, z) = 0.74$ and *cost* $(x, z) = 0.48$. We can now build the corresponding *bit-dependency graph* (Fig.12c) assigning to each bit a vertex and weighting the edges with the cost function (α coefficients are $\alpha(x,y) = \alpha(y,z) = 1/2$, $\alpha(x,z) = 1/4$).

FIGURE 12. An example of a bit-dependency graph

As we can see, bits *y* and *z* are the most dependent ones. If a 2-way min-cut partitioning is desired, the solution to the problem is shown above: put *x* in one block and *y, z* in the other one. Bits in different blocks will thus be considered to be independent.

5. Practical Considerations and Experimental Results

As stated previously, we will restrict our attention to the application of SSMs to sequence generation and compaction although their applicability goes beyond these. When power becomes a factor in designing digital circuits, the problem of sequence characterization and reproducibility of experiments plays an important part. In addition, with a much higher practical impact, input sequence compaction can significantly decrease the design cycle time by drastically reducing the simulation time. Let us analyze all these issues in more detail.

The problem of sequence compaction is related to that of sequence generation. Because the latter is contained as a step in the compaction process, we will address the generation problem through the compaction problem.

FIGURE 13. The experimental setup

Our strategy is depicted in Fig.13 and follows the steps of the algorithm in Section 4. Basically, we verified our ability to generate and compact constrained input sequences which may be also used as power benchmarks in the design process. In all experiments, we target *lossy compression* [16], that is the process of transforming an input sequence into a smaller one, such that the new body of data represents a *good approximation* of the original data as far as power consumption is concerned. If there was an initial sequence of length L_0 and it turns out that $L < L_0$, 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 $r = L_0/L$ was achieved.

Starting with an *n*-bit input sequence of length L_0, we extract the initial set of statistics and based on it, if the number of bits *n* is too large to be handled as a whole, the set of input bits is partitioned into *b* subsets (blocks) using the Kernighan-Lin heuristic as described in Section 4. To each block, we then associate a stochastic machine (SSM_1, SSM_2,...,SSM_b in Fig.14b). This is similar to approximating a single source on a large number of bits with many independent sources, each one having a smaller number of bits.

FIGURE 14. Two possible strategies

We note that, as a side effect, this strategy may introduce new vectors (that is, vectors that were absent the original sequence) in the final compacted sequence.

Once a partition is obtained, we simply apply the algorithm in Section 4 to each group of bits, that is, we build the matrix *A* (by preserving exactly the transition probabilities) and after that, decompose it into a set of degenerate matrices as stated in Theorem 1. A

deterministic sequential machine is then constructed for each degenerate matrix in this decomposition. The next step does the actual generation of the output sequence (of length L): the resulting auxiliary inputs are excited by a random number generator satisfying the probability distribution from the decomposition process. From our experience, this strategy works well (less than 5% relative error on average) for pseudorandom and moderately biased input sequences. If the sequence to be compressed is a highly correlated one, then this approach will result in an error level of about 5-10% on average. In such cases, the global modeling of the SSM, if possible, (as depicted in Fig.14a) can be used to improve accuracy.

Finally, a validation step is included in the strategy; using an in-house gate-level logic simulator (which accounts for spurious activity in the circuits) developed under the SIS environment, the total power consumptions of some ISCAS'85 and ISCAS'89 benchmarks are 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 our results for real-delay model and type 1 sequences of length L_0=100,000 compacted with different compaction ratios (namely r = 100 and 1000) using the strategy in Fig.14b; this type of sequence was obtained by randomly applying a set of logic operators (AND, OR, XOR) between the bits of a random sequence. This increases the correlations among the bits because it changes not only the signal and transition probability of each bit, but also the pairwise transition probabilities between bits. For each value of the compaction ratio, different sizes were allowed for the number of bits per block (k = 4, 6). For instance, the circuit C1355 has 41 inputs which means a number of 11 blocks with a maximum number of k = 4 bits per block (and accordingly 11 stochastic machines SSM_1, SSM_2,..., SSM_{11} in Fig.14b), or 7 blocks with a maximum number of k = 6 bits per block. For two of the sequential circuits (i.e., s298 and s386) the partitioning step was unnecessary due to the small number of input bits (3 and 7, respectively).

TABLE 1. Total power (μW@20MHz) for sequences of type 1 (real delay)

As we can see, the quality of results is very good even when the length of the initial sequence is reduced by 3 orders of magnitude. Thus, for C880 in Table 1, instead of simulating 100,000 vectors with an exact power of 5990.40 μW, one can use only 1000 vectors with an estimate of 5976.60 μW (k = 6) or just 100 vectors with a power consumption estimated as 6117.60 μW (k = 4).

This reduction in the sequence length has a significant impact on speeding-up the simulative approaches for power estimation where the running time is proportional to the length of the sequence which must be simulated. It should be pointed out that in the real cases where millions of vectors are applied, compaction ratios of more than 1000 may be safely used.

On the efficiency side, in Table 2 we report the running times obtained in each step of the process on a Sun SPARC 20. As the results show, the most time consuming step in our proposed approach is the time it takes to do sequence generation. These values were

obtained using a threshold of $\varepsilon = 0.001$ (that is, every probability is calculated with 3 exact digits). Setting this to a smaller value (e.g. $\varepsilon = 0.01$) will dramatically reduce the running time. Differently stated, by varying ε, one can trade-off accuracy vs. efficiency (as guaranteed by Theorem 2) if this is satisfactory from a practical point of view.

TABLE 2. CPU time (sec.) for sequences in Table 1

In Tables 3-4, we provide only the real-delay gate-level simulation results for a set of highly biased sequences (type 2 and type 3) obtained from industry. Type 2 sequences have a length of 4,000 and were compacted using the strategy illustrated in Fig.14a for two compaction ratios (r = 5 and 10). Sequences of type 3, having a length of 200,000, were compacted with the same strategy as above for three compaction ratios (r = 50, 100 and 200); the results are presented in Table 4.

TABLE 3. Total power for sequences of type 2

TABLE 4. Total power for sequences of type 3

As reported in Tables 3-4, results are still good, the average relative error being around 5% and 3% on average, respectively. As an important observation, we note that the values in the initial transition matrix themselves are important in the decomposition process: some distributions of transition probabilities tend to favor a small number of degenerate matrices, as opposed to others which result in much longer decompositions. In these cases, the decomposition becomes the critical step as far as running time is concerned.

In our analysis, we chose a gate-level simulator but our results were consistently good for a more accurate simulator such as Power Mill [20]. Moreover, under a more detailed scenario where *node-by-node* power values were extracted, the results are again very good. To support this claim, in Table 5 we present the results obtained for sequences of type 3 and compaction ratio r = 200.

TABLE 5. Node-by-node sw_act analysis for sequences of type 3

To derive the values in Table 5, we compared the switching activity estimates for the compacted sequences against those obtained for the initial sequences considering each internal node and primary output for every circuit. We report here the usual measures for accuracy: maximum error (MAX), mean error (MEAN), root-mean square (RMS) and standard deviation (STD); we exclude deliberately the relative error from this picture, due to the misleading prognostic it gives for small values.

To summarize, huge compaction ratios (3 or more orders of magnitude) can be obtained in a short amount of time with a small loss in accuracy for total power prediction,

either for combinational or sequential circuits (zero- vs. real-delay). From this perspective, simulative approaches will significantly benefit from these results.

6. Conclusion

In this paper, we addressed the problem of stochastic machines synthesis targeting constrained sequence generation or compaction. Shifting the attention from the ‘circuit problem’ to the ‘input problem’, we proposed an original approach to generate input sequences (which must satisfy a set of constraints) and to compact an existing sequence into a much shorter equivalent one.

The mathematical foundation of this approach relies in probabilistic automata theory and based on this, a general procedure for SSM synthesis is revealed. After that, these machines can be used in a stand-alone mode for sequence generation or compaction. The issues brought into attention on this paper are new to the CAD community and represent a first step to reduce the gap between simulative and probabilistic techniques which are currently the norm.

7. References

[1] M. Pedram, 'Power Minimization in IC Design: Principles and Applications', in *ACM Trans. on Design Automation of Electronic Systems,* vol.1, No.1, pp. 1-54, Jan.1996.

[2] 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. 270-275, June 1992.

[3] F. N. Najm, 'Transition Density, A Stochastic Measure of Activity in Digital Circuits', in *Proc. ACM/IEEE Design Automation Conference*, pp. 644-649, June 1991.

[4] 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.

[5] T. L. Chou, K. Roy, and S. Prasad, 'Estimation of Circuit Activity Considering Signal Correlations and Simultaneous Switching', in *Proc. IEEE/ACM Intl. Conference on Computer Aided Design,* pp. 300-303, Nov. 1994.

[6] F. N. Najm, 'Feedback, Correlation and Delay Concerns in the Power Estimation of VLSI Circuits', in *Proc. ACM/IEEE Design Automation Conference,* pp. 612-617, June 1995.

[7] R. Marculescu, D. Marculescu, and M. Pedram, 'Probabilistic Modeling of Dependencies During Switching Activity Analysis', in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* vol.17, No.2, pp. 73-83, Feb. 1998.

[8] P. H. Bardell, W. H. McAnney, and J. Savir, 'Built-in Test for VLSI: Pseudorandom Techniques', J. Wiley & Sons Inc. 1987.

[9] 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.

[10] J. Von Neumann, 'Probabilistic Logics and Synthesis of Reliable Organisms from Unreliable Components', in *Annals of Mathematics Studies,* Vol.34, pp.43-98, Princeton Univ. Press, Princeton, New Jersey 1956.

[11] A. Davis, 'Markov Chains as Random Input Automata', in *American Mathematical Monthly,* Vol.68, pp. 264-267, 1961.

[12] M. Rabin, 'Probabilistic Automata', in *Information and Control,* Vol.6, pp. 230-245, 1963.

[13] M. Garey, and D. Johnson, 'Computers and Intractability', New York: Freeman, 1979.

[14] B. Kernighan and S. Lin, 'An Efficient Heuristic Procedure for Partitioning Graphs', in *Bell Systems Technical Journal,* Vol.49, No.2, pp.291-307, 1970.

[15] C. Fiduccia and R. Matheyses, 'A Linear-Time Heuristic for Improving Network Partitions', in Proc. *ACM/IEEE Design Automation Conference*, pp. 175-181, June 1982.

[16] J. Storer, 'Data Compression: Methods and Theory', Chapter 1, Computer Science Press, 1988.

[17] A. Paz, 'Introduction to Probabilistic Automata', Academic Press 1971.

[18] A. Papoulis, ‘Probability, Random Variables, and Stochastic Processes’, McGraw-Hill Co., 1984.

[19] J.H.Wilkinson, ‘The Algebraic Eigenvalue Problem’, Clarendon Press, 1988.

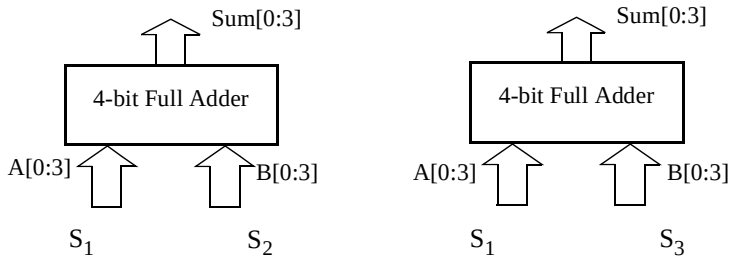
[20] C. X. Huang, B. Zhang, A.-C. Deng, and B. Swirski, ‘The Design and Implementation of PowerMill’, in *Proc. Intl. Workshop on Low Power Design*, pp. 105-110, April 1995.

[21] A. Gill, ‘Synthesis of Probability Transformers’, *J. Franklin Inst.*, 274, pp. 1-19, July 1962.

[22] A. Gill, ‘On a Weight Distribution Problem with Application to the Design of Stochastic Generators’, *Journal of ACM*, 10, pp. 110-121, Jan. 1965.

[23] C.-Y. Tsui, J. Monteiro, M. Pedram, S. Devadas, A. M. Despain, and B. Lin, ‘Power Estimation Methods for Sequential Logic Circuits’, in *IEEE Trans. on VLSI Systems*, vol.3, No.3, pp. 404-416, Sept. 1995.

List of Figures



S_1	S_2
0000 0001	0111 0110
1000 0000	0111 0111
0100 0000	0111 1000
1010 0000	0111 1001
1101 0000	0111 1010
1110 1000	0111 1011
0111 0100	0111 1100
0011 1010	0111 1101
0001 1101	0111 1110
1000 1110	0111 1111
1100 0111	1000 0000
1110 0011	1000 0001
0111 0001	1000 0010
1011 1000	1000 0011
1101 1100	1000 0100
1110 1110	1000 0101
1111 0111	1000 0110
0111 1011	1000 0111
0011 1101	1000 1000
0001 1110	1000 1001

(a)

S_1	S_3
0000 0001	1100 0111
1000 0000	1110 0011
0100 0000	0111 0001
1010 0000	1011 1000
1101 0000	1101 1100
1110 1000	1110 1110
0111 0100	1111 0111
0011 1010	0111 1011
0001 1101	0011 1101
1000 1110	0001 1110
1100 0111	0000 0001
1110 0011	1000 0000
0111 0001	
1011 1000	
1101 1100	
1110 1110	
1111 0111	
0111 1011	
0011 1101	
0001 1110	

(b)

FIGURE 1. The input problem

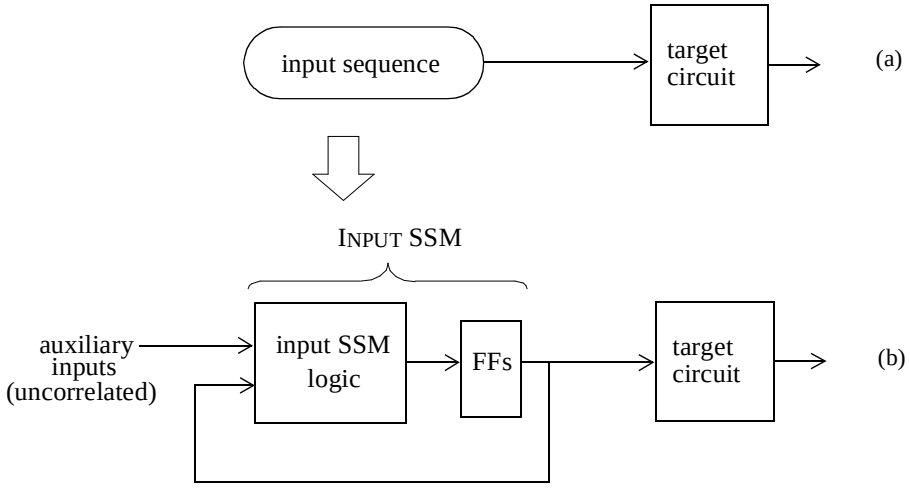


FIGURE 2. Constrained sequence generation

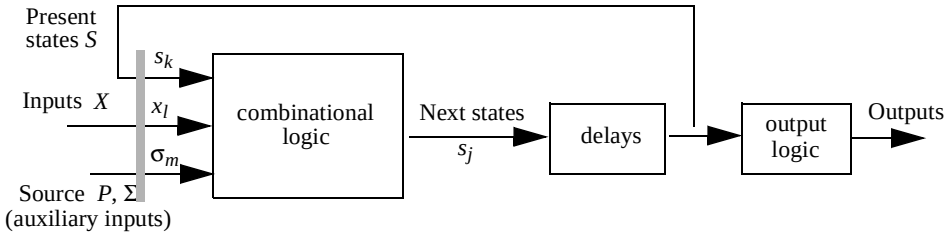


FIGURE 3. The general structure of the SSM

w_1 w_2	x	$s_1^{(n)}$	$s_1^{(n+1)}$	y
0 0	0	0	0	1
0 0	0	1	1	0
0 0	1	0	0	1
0 0	1	1	0	0
0 1	0	0	1	1
0 1	0	1	0	0
0 1	1	0	1	1
0 1	1	1	1	0
1 0	0	0	1	1
1 0	0	1	1	0
1 0	1	0	0	1
1 0	1	1	1	0
1 1	0	0	1	1
1 1	0	1	1	0
1 1	1	0	1	1
1 1	1	1	1	0

FIGURE 4. The behavior of the SSM in Example 1

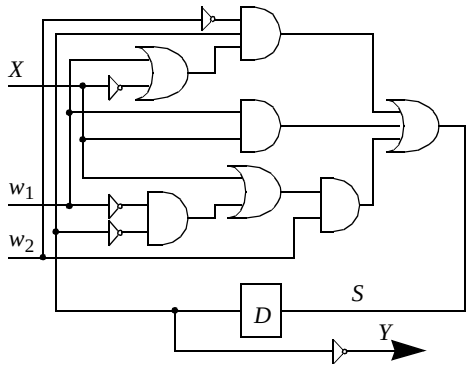


FIGURE 5. A possible implementation of the SSM in Example 1

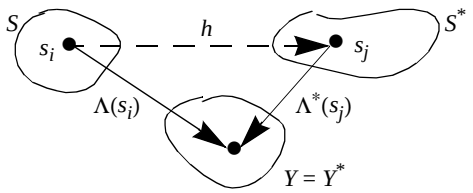


FIGURE 6. Output equivalence

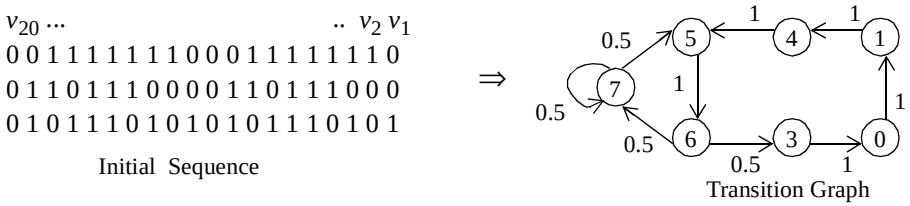


FIGURE 7. An input sequence and its transition graph

w	$y_1^{(n)}$	$y_2^{(n)}$	$y_3^{(n)}$	$y_1^{(n+1)}$	$y_2^{(n+1)}$	$y_3^{(n+1)}$
0	0	0	0	0	0	1
0	0	0	1	1	0	0
0	0	1	0	–	–	–
0	0	1	1	0	0	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	1
1	0	0	0	0	0	1
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	1	1

FIGURE 8. The transition table for the SSM in Example 2

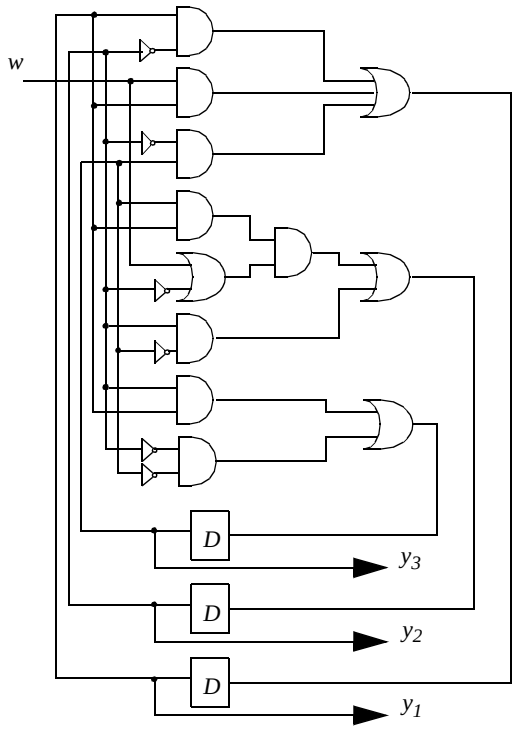


FIGURE 9. The SSM generating the original sequence in Example 2

step	*w*	$y_1^{(n)}$	$y_2^{(n)}$	$y_3^{(n)}$	$y_1^{(n+1)}$	$y_2^{(n+1)}$	$y_3^{(n+1)}$
1	0	1	1	1	1	0	1
2	1	1	0	1	1	1	0
3	1	1	1	0	1	1	1
4	1	1	1	1	1	1	1
5	0	1	1	1	1	0	1
6	1	1	0	1	1	1	0
7	0	1	1	0	0	1	1
8	0	0	1	1	0	0	0
9	0	0	0	0	0	0	1
10	1	0	0	1	1	0	0

FIGURE 10. A possible behavior of the SSM

```
procedure Generate_Sequence ()
begin
   A = Construct_Matrix (); /* either user specified or from another sequence */
   /* for a given level of accuracy ε, do decomposition of matrix A */
   while | Σp_i -1 | < ε do
      Decompose_Matrix (); /* recursive procedure implementing Theorem 1 and
Theorem 2 */
   end while;
   /* let t be the number matrices U_i in the decomposition */
   {σ_i} = Encode_Symbols (t);
   /* construct the transition table for the SSM and synthesize the circuit */
   Table = Construct_Table ({U_i});
   Circuit = Construct_Circuit (Table);
   /* generate the new sequence providing corresponding values for the auxiliary
inputs w_i */
   Gen_Sequence (Circuit);
end Generate_Sequence;
```

FIGURE 11. The generation procedure

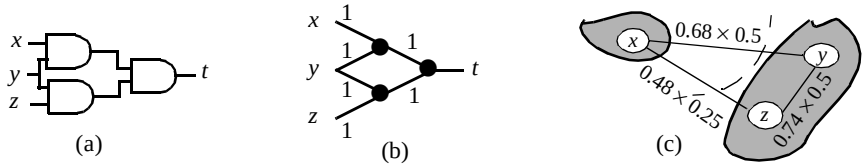


FIGURE 12. An example of a bit-dependency graph

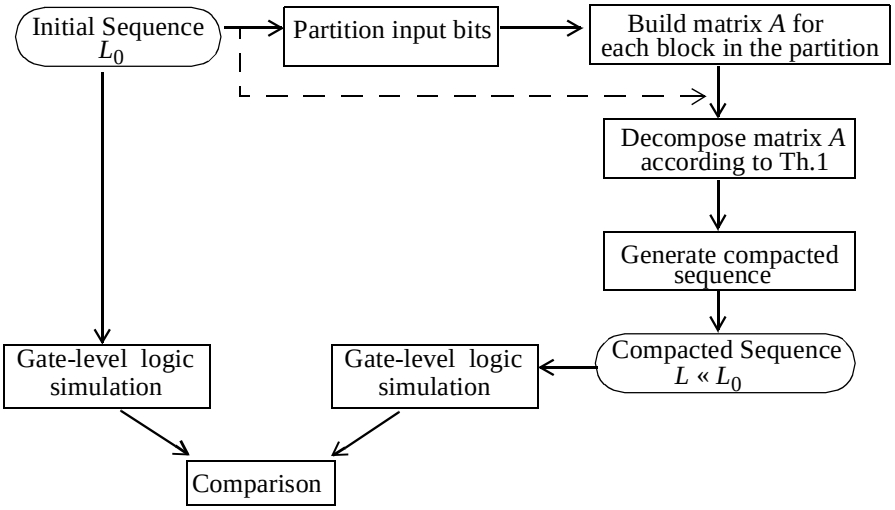


FIGURE 13. The experimental setup

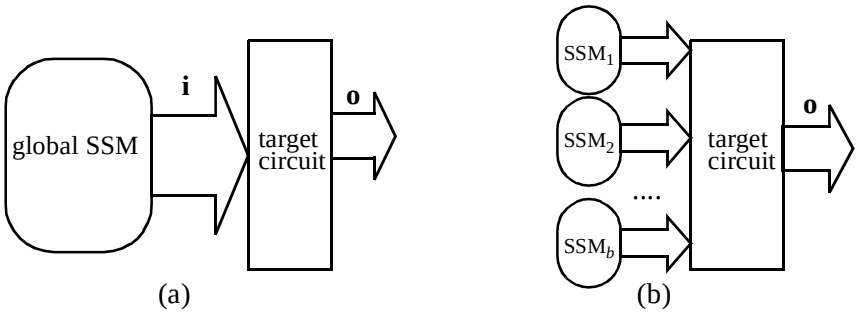


FIGURE 14. Two possible strategies

List of Tables

TABLE 1. Total power (μW@20MHz) for sequences of type 1 (real delay)

			***r* = 100**		***r* = 1000**	
Circ.	**#Inp.**	**Exact Power**	***k* = 4**	***k* = 6**	***k* = 4**	***k* = 6**
C1355	41	4218.00	4276.80	4236.00	4336.80	4134.60
C1908	33	6990.00	6894.00	6966.60	6597.60	6921.00
C3540	50	19603.20	19497.00	19102.20	19626.60	18646.20
C432	36	3070.80	3065.40	3024.60	3024.00	3052.20
C499	41	5374.20	5431.20	5397.00	5530.80	5235.60
C6288	32	347886.0	354933.60	349987.20	359386.80	336701.40
C880	60	5990.40	6084.00	5976.60	6117.60	5871.00
s1196	14	7698.60	7594.20	7452.60	7914.60	6989.40
s344	9	1814.40	1851.60	1865.40	1975.80	2062.80
s641	35	2908.80	2805.00	2806.20	2674.80	2713.20
s838	34	1551.00	1554.60	1503.60	1551.60	1438.20
s9234	36	21693.60	21679.20	21042.60	21064.20	21875.40
	Average Error (%)		1.05	1.81	3.50	3.32
s298	3	975.00	972.60		954.60	
s386	7	1996.80	2023.20		2030.40	
	Average Error (%)		0.78		1.89	

TABLE 2. CPU time (sec.) for sequences in Table 1

	Partition		**Decomposition**		**Generation** r **= 100**		**Generation** r **= 1000**	
Circ.	k **= 4**	k **= 6**	k **= 4**	k **= 6**	k **= 4**	k **= 6**	k **= 4**	k **= 6**
C1355	0.02	0.01	0.10	0.73	2.22	2.02	0.25	0.22
C1908	0.01	0.01	0.08	0.59	1.63	1.51	0.19	0.16
C3540	0.03	0.02	0.13	0.84	2.22	2.02	0.25	0.22
C432	0.01	0.01	0.09	0.57	1.81	1.73	0.20	0.18
C499	0.02	0.01	0.10	0.73	2.22	2.02	0.25	0.22
C6288	0.01	0.01	0.07	0.54	1.54	1.45	0.17	0.16
C880	0.04	0.02	0.17	1.13	3.92	3.55	0.43	0.37
s1196	0.01	0.01	0.03	0.06	0.55	0.51	0.06	0.06
s344	0.01	0.01	0.13	0.02	0.13	0.11	0.02	0.01
s641	0.01	0.01	0.08	0.60	1.75	1.67	0.19	0.17
s838	0.01	0.01	0.08	0.59	1.70	1.55	0.18	0.17
s9234	0.01	0.01	0.09	0.57	1.81	1.73	0.20	0.18
s298	-		0.01		0.04		0.01	
s386	-		0.11		0.07		0.01	

.

TABLE 3. Total power for sequences of type 2

Circ.	**Exact Power**	***r* = 5**	***r* = 10**
C1355	3783.17	3863.27	3918.51
C1908	6352.03	6683.00	6592.43
C3540	14471.32	12603.73	13034.91
C432	1809.95	1706.08	1860.58
C499	4390.45	4470.10	4467.74
C6288	104117.45	95628.77	92198.86
C880	3787.93	3526.17	3716.96
	Avg. Error (%)	6.11	5.06

TABLE 4. Total power for sequences of type 3

Circ.	**Exact Power**	***r* = 10**	***r* = 50**	***r* = 200**
C1355	1878.10	1882.05	1895.54	1900.87
C1908	977.69	958.75	989.99	958.95
C3540	674.15	591.81	592.93	581.40
C432	1033.16	1041.26	1049.38	986.68
C499	2323.73	2316.62	2304.67	2289.72
C6288	2073.94	2048.41	2024.81	2014.79
C880	1355.13	1344.20	1329.08	1380.29
	Avg. Error (%)	2.50	2.99	3.94

TABLE 5. Node-by-node sw_act analysis for sequences of type 3

Circ.	MAX	MEAN	RMS	STD
C1355	0.0286	0.0032	0.0070	0.0063
C1908	0.0206	0.0008	0.0032	0.0031
C3540	0.0448	0.0025	0.0082	0.0078
C432	0.0300	0.0036	0.0068	0.0057
C499	0.0286	0.0031	0.0070	0.0063
C6288	0.0538	0.0004	0.0018	0.0018
C880	0.0208	0.0013	0.0042	0.0040