INFORMATION THEORETIC AND PROBABILISTIC MEASURES FOR POWER ANALYSIS OF DIGITAL CIRCUITS

by

Diana Marculescu

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

(ELECTRICAL ENGINEERING)

August, 1998

To my family

Acknowledgments

I would like to thank my advisor, Professor Massoud Pedram, for his continuous support, guidance and encouragement throughout the process of working towards my Ph.D. degree. I am deeply grateful for his support in difficult moments and for being not only my advisor, but also a friend. Special thanks to his wife, Afsane, who always encouraged me during my last years as a doctoral student. Many thanks to Professors Melvin Breuer and Michael Arbib for serving in my thesis committee and for giving me key advice during my last year as a doctoral student. I would also like to thank my colleagues in the Low Power CAD group at USC, especially Chi-Ying Tsui and Chih-Shun Ding for interesting and stimulating discussions.

Also, I cannot omit from my acknowledgment list the people who guided and influenced my career even before it started: my elementary school teacher Veronica Marinescu, who introduced me to the wonderful world of mathematics; from "Politehnica" University of Bucharest, Professors Octavian Stanasila and Paul Flondor of the Department of Mathematics who were my mathematics mentors during my college years; Professor Cristian Giumale of the Department of Computer Science who taught me how to be a good computer scientist; and my Master's thesis advisor, Professor Irina Athanasiu of the Department of Computer Science, who encouraged me to pursue an academic career.

Last, but not least, I would like to thank to the ones without whom this would have never been possible, my family: especially to my husband Radu and my son Andrei, for helping me and being a constant and loving support in every difficult moment, and to my parents Gina and Adrian, and my sister Daniela for encouraging me and believing in me. With all my heart, this thesis is dedicated to all of them.

Contents

List of Figures

List of Tables

Abstract

Driven by increased levels of device integration and complexity, together with higher device speed, power dissipation has become a crucial design concern, limiting the number of devices that can be put on a chip and dramatically affecting the packaging and cooling costs of ICs.

The research presented in this thesis focuses on power analysis of digital circuits which is an indispensable component of any power optimization and synthesis environment. The mathematical foundations of this work are derived from the probability and information theories and provide a comprehensive solution for the problem of power estimation at different levels of abstraction.

Based on the concept of stochastic sequential machine, we present an effective technique for compacting a long sequence of input vectors into a much shorter input sequence so as to reduce the circuit or gate level simulation time by orders of magnitude while maintaining the accuracy of power estimates. In practice, compaction ratios of 1-3 orders of magnitude can be obtained without much loss in accuracy (6% on average).

At a higher level of design abstraction, we describe a technique to estimate the power consumption in digital circuits based on two abstract measures: entropy and informational energy. A major advantage of this technique is that it is not simulation-based and thus is extremely fast, yet it produces accurate power estimates (i.e., 10% error on average).

Finally, we investigate the effect of finite-order statistics of the input stream on the finite-state machine behavior. Formally, we prove that assuming temporal independence or even using first-order temporal models for the input stream is insufficient and may induce significant inaccuracies. Experimental results show that, if the order of the source is

underestimated, the steady-state probabilities can be off by more than 100% from the correct ones, thus affecting the accuracy of power estimates.

The results presented in this thesis represent a significant step towards a better understanding of the behavior of digital circuits from a probabilistic point of view. They are applicable to not only power estimation, but also other areas where probabilistic modeling is appropriate.

Chapter 1 Introduction

1.1 Basic issues in power estimation

In the past the major concerns of the VLSI designers were area, speed, cost, and reliability. In recent years, however, this has changed and, increasingly, power is being given comparable weight to area and speed. This is mainly due to the remarkable success of personal computing devices and wireless communications systems which demand high-speed computation and complex functionality with low power consumption. For this type of applications, the main concern is the average power consumption and, consequently, the need for prolonging the battery life. In addition, power is becoming a more important issue for high-end products (microprocessors) which require costly packaging and cooling devices. Finally, there is the issue of reliability. High power systems tend to run hot and high temperature tends to exacerbate failure mechanisms in silicon. Thus, there is a whole range of applications that will benefit from techniques and mechanisms of reducing power consumption. Because “one cannot optimize what he cannot analyze,” power analysis tools which give accurate power estimates are extremely important.

Power dissipation in CMOS circuits comes from three sources: leakage currents which include the reverse-biased junction and subthreshold currents, short-circuit currents which flow due to the DC path between the supply rails during output transitions, and capacitive switching currents which are responsible for charging and discharging of capacitive loads during logic transitions. In well-designed circuits with relatively high threshold voltages, the first two sources are very small compared to the last one. Therefore, to estimate the total power consumption of a module (in a gate level implementation), we may only

account for the capacitive switching currents, yet achieve sufficient levels of accuracy [62]:

$$P_{avg} = \frac{f_{clk}}{2} \cdot V_{DD}^2 \cdot \sum_n (C_n \cdot sw_n) \quad (1.1)$$

where f_{clk} is the clock frequency, V_{DD} is the supply voltage, C_n and sw_n are the capacitive load and the average switching activity of gate n, respectively (the summation is performed over all gates in the circuit). As we can see, in this formulation the average switching activity per node is a key factor, and therefore its correct calculation is essential for accurate power estimation. Note that for the same implementation of a module, different input sequences may give rise to different switching activities at the circuit inputs and at the outputs of internal gates and consequently, completely different power values.

Circuit and gate level power estimation techniques roughly fall in two categories: *simulative* and *nonsimulative*. General simulation techniques can provide sufficient accuracy, but the computational cost is too high. On the other hand, probabilistic techniques are faster, but their accuracy is limited. Despite the efforts directed in both directions, the performance gap between simulative and probabilistic approaches has remained basically the same during the last few years [51].

A major concern 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 to 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

[14][53] and therefore capture all internal dependencies, to efficient techniques which partially account for dependencies in an incremental manner [41][59][22][7]. Recently, some authors [43][35] 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 seriously impaired. We will refer to this as *'the input problem'* which is important not only in power estimation, but also in power minimization.

At higher levels, designers are becoming more and more interested in register-transfer level (RTL) modules (adders, multipliers, registers, multiplexers) and strategies to put them together to build complex digital systems. Power minimization in digital systems is not an exception to this trend. Having as soon as possible in the design cycle an estimate of power consumption can save significant redesign efforts or even completely change the entire design architecture. At this level of abstraction, fidelity is more important than accuracy, that is, relative evaluation of different designs in terms of their power dissipation is often sufficient.

Higher levels of abstraction have been considered in power estimation of digital circuits, but here many problems are still pending a satisfactory solution. Most of the high level prediction tools combine deterministic analysis with profiling and simulation in order to address data dependencies. Important statistics include the number of instructions of a given type, the number of bus, register and memory accesses and the number of I/O operations executed within a given period [5][38]. Analytic modeling efforts have been described in [26] where a parametric power model was developed for macromodules. However, the trade-off between flexibility and accuracy is still a

challenging task as major difficulties persist due to the lack of precise information and the conceptual complexity which characterizes these levels.

In conclusion, a number of issues appear to be important for power estimation and low-power synthesis. First, the *statistics* and the *length of the input sequences* which are applied to a circuit are important. Probabilistic techniques are very efficient, but to be accurate, they need to take into account complex input dependencies which in turn reduce their efficiency. On the other hand, simulation-based techniques provide higher accuracy at the expense of higher computational cost. Thus, it is desirable to have a power estimation technique which combines the advantages of both worlds. To solve this problem, we propose a simulative power estimation technique based on *sequence compaction*, that is, to generate a much shorter sequence exhibiting the same behavior as the original one. Potentially, a good solution to the problem of generating a short and meaningful set of stimuli (to be used later for simulation) will reduce the computational gap between simulative and nonsimulative techniques for power estimation.

Second, it is desirable to have as early as possible in the design cycle an estimate of the power consumption. This will enable designers to have a better view of the design process and to control more efficiently the power optimization step. For this purpose, we propose a global measure which characterizes the average switching activity per module. This measure, combined with information about the total capacitance of the module, will provide the designer with an early estimate of the total power consumption.

Finally, for the special case of controllers, new and more accurate techniques are needed to analyze their probabilistic behavior. This type of analysis is useful not only in

controller power estimation at higher levels of abstraction, but can also be used for verification or synthesis purposes. The main problem and the most difficult one is to correctly estimatie the steady-state transition probabilities for a finite-state machine. Previous work in this area considered the simplifying assumption of temporal independence on the input stream, but a comprehensive and correct probabilistic analysis of the finite-state machine has to take into account complex temporal and spatial dependencies that may appear on the primary inputs.

1.2 Thesis contribution

Based on the previous remarks, this thesis addresses the following issues:

- *Constrained sequence generation using stochastic sequential machines*

Power estimation techniques must be fast and accurate to be applicable in practice. Not surprisingly, these two requirements interfere with one another and at same point they become contradictory. General simulation techniques can provide sufficient accuracy, but the price we have to pay 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 require simulation results for large circuits. The importance of correlations not only inside the target circuit (due to reconvergent fanout regions), but also at the circuit inputs has been pointed out in [35][36]. From this perspective, this part of our work shifts the focus from ‘the circuit problem’ (induced by reconvergent fanout) to ‘the input problem’ (or input correlations) and improves the state-of-the-art by proposing an original solution for *constrained sequence generation* (that is, satisfying a prescribed set of statistics). In

our research, we concentrated on the following areas of application for constrained sequence generation:

–*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 order*) among them. As illustrated in Figure 3.1, this is important especially in low-power design. This issue will be discussed in detail in Section 3.2.

–*Sequence compaction:* This is basically the ability to construct a *representative sequence (*short enough to be efficiently simulated) which is equivalent with the original sequence as far as the total power consumption is concerned. The shorter sequence is used to reproduce the operating context in which the circuit is supposed 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 Section 3.3 and Section 3.4.

–*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 [15][16]. Such a probability transformer, when placed in front of the circuit under consideration, enables the use of a probabilistic power estimation approach that assumes temporal independence of the input stimuli since 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. The product machine (*input SSM, target circuit*) (see Figure 3.2) can thus be analyzed using probabilistic power estimation techniques such as [60].

The mathematical foundation of the approach relies on the *stochastic sequential machines* (SSMs) theory. This research reveals a general procedure for SSM synthesis and describes a new framework for sequence characterization to match designer's needs for sequence generation or compaction. Results show that, in addition to providing an elegant and theoretical framework for sequence characterization, this approach results in up to 1000X compaction ratios while not exceeding an average of 6% error in power estimates. Moreover, the generality of these results makes them applicable in other areas where the environment (that is, the input statistics) are important. The bulk of this work appeared in [30] and was later extended in [31].

- *Information-theoretic measures for high-level power estimation*

Based on two concepts from information theory (namely, *entropy* and *informational energy)*, this part of our work presents fast, yet accurate, power estimation at the algorithmic behavioral levels. Entropy characterizes the uncertainty of a sequence of applied vectors and thus is intuitively related to switching activity. Indeed, it is shown that, under the temporal independence assumption, the average switching activity of any bit is upper bounded by half of its entropy. Knowing the statistics of the input stream and having some information about the structure (or functionality) of the circuit, the input and output entropies per bit are calculated using a closed form expression that gives the output entropy per bit as a function of the input entropy per bit, a structure-dependent information scaling factor, and the distribution of gates as a function of logic depth in the circuit (or using a compositional technique which has a linear complexity in terms of the circuit size). Next, the average entropy per circuit line is calculated and used as an estimate of the average switching activity per signal line. This is then used to estimate the power dissipation of the module. A major advantage of this technique is that it is not simulation based (as are most high-level power estimation tools) and is thus very fast, yet it produces accurate power estimates. More specifically, for a large set of benchmark circuits, including random logic and data-path circuits (adders, multipliers) this technique provides power estimates in a matter of seconds with an error of only 10% on average when compared to circuit level simulation. In general, using structural information can provide more accurate results. On the other hand, evaluations based on functional information need less information about the circuit and therefore may be more appealing in practice as

they provide an estimate of power consumption earlier in the design cycle. The main part of this research first appeared in [28] and was further extended in [29].

- *Steady-state probability estimation in finite-state machines considering high-order temporal effects*

This part of our work investigates, from a probabilistic point of view, the effect of finite-order statistics of the input stream on the finite-state machine (FSM) behavior. As the main theoretical contribution, we extend the previous work done in [17][19] on steady-state probability calculation in FSMs. More precisely, our approach accounts for complex spatiotemporal correlations which are present at the primary inputs when the target machine models hardware which receives data from real applications. In particular, it is shown that if the input to an FSM can be modeled as a finite order Markov chain, then this is also true jointly for the inputs and state lines of the FSM [32]. Moreover, underestimating the order of the underlying input Markov chain may produce false conclusions regarding the set of reachable states of the FSM and their corresponding probability distribution. To solve the problem analytically, one needs to appropriately model the input and then construct the correct set of Chapman-Kolmogorov equations. This is done via *constrained reachability analysis* which accounts for the specific way in which the input source excites the FSM. To find the correct set of reachable states, one has to know the constraints characterizing the input sequence in terms of valid input vectors or valid vector sequences. In the case of finite-order Markov chains, these constraints can be efficiently represented using Ordered Binary Decision Diagrams (OBDDs) [4] which offer a canonical and compact set representation. Doing a constrained reachability analysis

using OBDD set representations [4], the correct set of reachable states and valid state transitions are found. The set of equations is then solved using advanced numerical techniques based on *stochastic complementation* and *aggregation/disaggregation*. The stochastic complementation method has the advantage of providing as a by-product the exact transition matrix for the states of the analyzed FSM, whereas the aggregation/ disaggregation technique is a very efficient tool for solving the problem for a large class of Markovian sources known as *nearly completely decomposable* systems. These contributions provide a deep insight in theoretical aspects of FSM probabilistic analysis along with a practical solution for finding the correct steady-state probability distribution. These results are useful in synthesis, verification and low-power design of sequential circuits under high-order correlations on the primary inputs. This work (with application to power estimation) was presented in [33].

1.3 Thesis outline

The remainder of the thesis is organized into five main chapters. Chapter 2 presents some basic concepts from probability and information theory that will be used throughout the thesis. Chapter 3 deals with the problem of circuit and gate level power estimation using constrained sequence generation. Chapter 4 presents a high-level power estimation technique based on information theoretic measures. Chapter 5 deals with the probabilistic analysis of FSMs at higher levels of abstraction. Finally, Chapter 6 summarizes the main contribution and outlines possible directions for future investigations.

Chapter 2 Preliminaries

2.1 Basic probability theory concepts

In this section we present the basic definitions and notation. Far from being exhaustive, we restrict our attention to only those concepts that are required by our working hypotheses. For a complete documentation, we refer the reader to references [23][47].

2.1.1 Finite-order MCs

A *stochastic process* is defined as a family of random variables $\{x(t), t \in T\}$ defined on a given probability space and indexed by the parameter t representing time, where t varies over the index set T. The stochastic process is said to be *stationary* when it is invariant under an arbitrary shift of the time origin. In this case, the values assumed by the random variable *x(t)* are called *states*, and the set of all possible states forms the *state space* of the process.

A *Markov process* $\{x(t), t \in T\}$ is a stochastic process whose past has no influence on the future if its present is specified. This is so called “Markov property” and defines a fundamental subclass of stochastic processes. We shall assume that the transitions out of state *x(t)* are independent of time and, in this case, the Markov process is said to be *time-homogeneous*.

If the state space of a Markov process is *discrete*, the Markov process is referred to as a Markov chain (MC*)*. In what follows, we consider only MCs with finite state space. If we assume that the index set T is also discrete, then we have a *discrete-parameter MC*. We may assume without loss of generality that T = {0, 1, 2,...} and denote the MC as $\{x_n\}_{n \geq 0}$.

Definition 2.1 (lag-one MC) A discrete stochastic process $\{x_n\}_{n \geq 0}$ is said to be a lag-one MC if at any time step $n \geq 1$ and for all states $\{\alpha_1, \alpha_2, ..., \alpha_q\}$:

$$p(x_n = \alpha_{i_n} | x_{n-1} = \alpha_{i_{n-1}}, x_{n-2} = \alpha_{n-2}, \ldots, x_0 = \alpha_{i_0}) = p(x_n = \alpha_{i_n} | x_{n-1} = \alpha_{i_{n-1}}) \quad (2.1)$$

The conditional probabilities $p(x_n = \alpha_{i_n} | x_{n-1} = \alpha_{i_{n-1}})$ are called *single-step transition probabilities* and represent the conditional probabilities of making a transition from state $\alpha_{i_{n-1}}$ to state α_{i_n} at time step *n*. In homogeneous MCs these probabilities are independent of *n* and consequently written as $p_{ij} = p(x_n = j | x_{n-1} = i)$ for all $n = 1, 2, ...$ The matrix *Q*, formed by placing p_{ij} in row *i* and column *j*, for all *i* and *j*, is called the *transition probability matrix*. We note that *Q* is a *stochastic matrix* because its elements satisfy the following two properties: $0 \leq p_{ij} \leq 1$ and $\sum_j p_{ij} = 1$.

An equivalent description of the MC can be given in terms of its *state transition graph* (STG). Each node in the STG represents a state in the MC, and an edge labelled p_{ij} (from node *i* to node *j)* implies that the one-step transition probability from state *i* to state *j* is p_{ij}.

Changes of states over $n > 1$ time steps are ruled by probability rules expressed in terms of p_{ij}. Let us denote by p_{ij}^n the probability of transition from state *i* to state *j* in exactly *n* steps, namely: $p_{ij}^n = p(x_{m+n} = j | x_m = i)$, for any integer *m*. It is easily seen that probabilities p_{ij}^n (which are called *n-step transition probabilities*) represent the entries of the Q^n matrix (called *n-step transition matrix)*, $n \geq 1$. The Q^n matrix itself is still a stochastic matrix and satisfies the identity $Q^{m+n} = Q^m \cdot Q^n$, $m, n \geq 0$ (Q^0 is by definition

the unit matrix *I*) or just the system of equations $p_{ij}^{m+n} = \sum_k p_{ik}^m \cdot p_{kj}^n$, known as the *Chapman-Kolmogorov equations*. In other words, to go from *i* to *j* in *(m+n)* steps, it is necessary to go from *i* to an intermediate state *k* in *m* steps and then from *k* to *j* in the remaining *n* steps. By summing over all possible intermediate states *k*, we consider all possible distinct paths leading from *i* to *j* in *(m+n)* steps.

Definition 2.2 (lag-*k* MC) A discrete stochastic process $\{x_n\}_{n \geq 0}$ is said to be a lag-*k* MC if at any time step $n \geq k$:

$$p(x_n = \alpha_{i_n} \mid x_{n-1} = \alpha_{i_{n-1}}, x_{n-2} = \alpha_{i_{n-2}}, \ldots, x_0 = \alpha_{i_0}) \\ = p(x_n = \alpha_{i_n} \mid x_{n-1} = \alpha_{i_{n-1}}, x_{n-2} = \alpha_{i_{n-2}}, \ldots, x_{n-k} = \alpha_{i_{n-k}}) \quad (2.2)$$

It should be noted that any lag-*k* MC can be reduced to a lag-one MC based on the following result.

Proposition 2.1 [47] If $\{u_n\}_{n \geq 0}$ is a lag-*k* MC then $\{v_n\}_{n \geq 0}$, where $v_n = (u_n, u_{n+1}, \ldots, u_{n+k-1})$, is a multivariate first-order MC. ■

As a consequence, the study of lag-*k* MCs is practically reduced to study the properties satisfied by lag-one MCs. For clarity, we will refer subsequently only to lag-one MCs but, by virtue of Proposition 2.1, all results translate to lag-*k* MCs.

2.1.2 Classification of states, decomposability and probability distributions

To better analyze the long-run behavior of a MC we need to distinguish between states that are guaranteed to be visited infinitely often and states to which the system may never return.

Definition 2.3 (recurrent/transient state) A state in a MC is called *recurrent* if the probability of returning to it after $n \geq 1$ steps is greater than zero. Otherwise, the state is called *transient*.

This definition does not imply that the transient state cannot be visited many times, only that the probability of returning to the state in an infinite number of steps is zero. In our subsequent discussion, we will consider that all states are recurrent since all transient states vanish (that is, their steady-state probabilities become zero) after a finite number of steps.

From Definition 2.3, if i is a recurrent state, we have that $p_{ii}^{n} > 0$, for some $n \geq 1$. If the greatest common divisor over all such integers n is $d > 1$, then the state i is also called *periodic* of period d. Otherwise (i.e., if $d = 1$), the state i is called *aperiodic*. A state that is recurrent and aperiodic is said to be *ergodic*. If all the states of a MC are ergodic, then the MC itself is said to be ergodic.

Definition 2.4 (nondecomposable/decomposable MC) A MC is said to be *nondecomposable* (or *irreducible*) if every state can be reached from every other state in a finite number of steps. Otherwise, the chain is called *decomposable* (*reducible*) and its transition matrix can be written as a block-diagonal matrix having the form:

$$Q = \begin{bmatrix} Q_1 & 0 & \dots & 0 \\ 0 & Q_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & Q_p \end{bmatrix} \tag{2.3}$$

where Q_1, Q_2,..., Q_p are square matrices and the p sets of states contain only recurrent states that do not communicate among them.

The above definition, may be used in an approximate sense too; we say that the MC is *nearly completely decomposable* if its transition matrix Q can be partitioned in the form

$$Q = \begin{bmatrix} Q_{11} & Q_{12} & \cdots & Q_{1p} \\ Q_{21} & Q_{22} & \cdots & Q_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ Q_{p1} & Q_{p2} & \cdots & Q_{pp} \end{bmatrix} \tag{2.4}$$

where the non-zero elements in the off-diagonal blocks are small[1] compared to those in the diagonal blocks. As we shall see later, the concept of near decomposability plays a significant part in our framework.

We now turn our attention to distribution defined on the states of a MC. We shall denote by π_i^n the probability that the MC is in state i at step n, that is $\pi_i^n = p(x_n = i)$. In vector notation we have that $\pi^n = (\pi_1^n, \pi_2^n, ..., \pi_i^n, ...)$, where π is a row vector. The probability that the MC is in state i at step n is given by $\pi^n = \pi(0) \cdot Q^n$, where $\pi(0)$ denotes the initial state distribution of the chain. For nondecomposable and aperiodic MCs it may be shown that the *limiting distribution* $\pi = \lim_{n \to \infty} \pi^n$ always exists and it is independent of the initial probability distribution. In addition, the following important results hold.

1. The precise meaning of ‘small’ will be defined later in Section 5.4.

Proposition 2.2 [23] For a nondecomposable MC, the equation $\pi \cdot Q = \pi$ has a unique solution that represents the *stationary distribution* (or stationary probability vector) of the MC. ■

The unique solution of the equation in Proposition 2.2 can actually be determined by solving the system of equations $\pi_j = \sum_i \pi_i \cdot p_{ij}$ with $\sum_j \pi_j = 1$. The vector π represents the *left eigenvector* corresponding to the unit eigenvalue, that is, it satisfies the standard equation $\pi \cdot Q = \lambda_1 \cdot \pi$ for $\lambda_1 = 1$.

2.2 Basic information theory concepts

2.2.1 Entropy

Let $A_1, A_2, ..., A_n$ be a complete set of events which may occur with the probabilities $p_1, p_2, ..., p_n$ that is:

$$p_k \geq 0, (\forall k), \quad k = 1, 2, \ldots, n \qquad \sum_{k=1}^{n} p_k = 1 \tag{2.5}$$

We introduce the following definition:

Definition 2.5 (Self-Information) The *self-information* of the event A_k, written $I(A_k)$, is defined as:

$$I(A_k) = -\log(p_k) \tag{2.6}$$

where log stands for $\log_2$. The function $I(A_k)$ vs. $p_k = p(A_k)$ is plotted in Figure 2.1.

Self-information can be interpreted as the amount of information provided by occurrence of the event A_k. According to this interpretation, the less probable an event is,

the more information we receive when it occurs. A certain event (one that occurs with probability 1) provides no information at all, whereas an unlikely event provides a large amount of information. The measurement unit for I is the *bit* (*bi*nary digi*t)*; when $p_k = 1/2$ ($k = 1, 2$), $-\log(p_k) = 1$ bit, so that 1 bit of information occurs on the choice of one from two equally likely events.

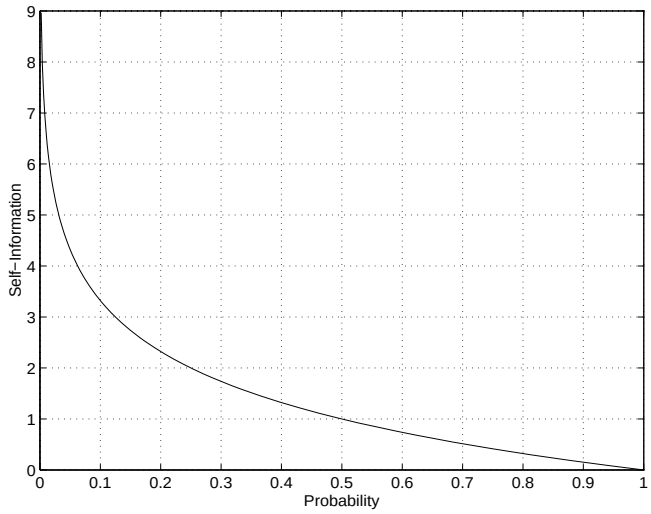


Figure 2.1 Self-information $I(A_k)$ vs. probability p_k.

Let us consider an experiment where the outcome is unknown in the beginning; such an experiment exposes a probability finite field $\mathcal{A}_n$, completely characterized by the probability distribution $p_1,p_2,...,p_n$. To quantify the content of information revealed by the outcome of such an experiment, Shannon introduced the concept of entropy [47].

Definition 2.6 (Entropy) Entropy of a finite field $\mathcal{A}_n$ (denoted by $H(\mathcal{A}_n)$) is given by:

$$H(\mathcal{A}_n) = H(p_1, p_2, \ldots, p_n) = -\sum_{k=1}^{n} p_k \log p_k \tag{2.7}$$

In other words, entropy is the weighted average of self-information over all events of the field. We plot in Figure 2.2 the entropy of a Boolean variable as a function of its signal probability, that is $H(\mathcal{A}_2) = -p \cdot \log p - (1-p) \cdot \log(1-p)$.

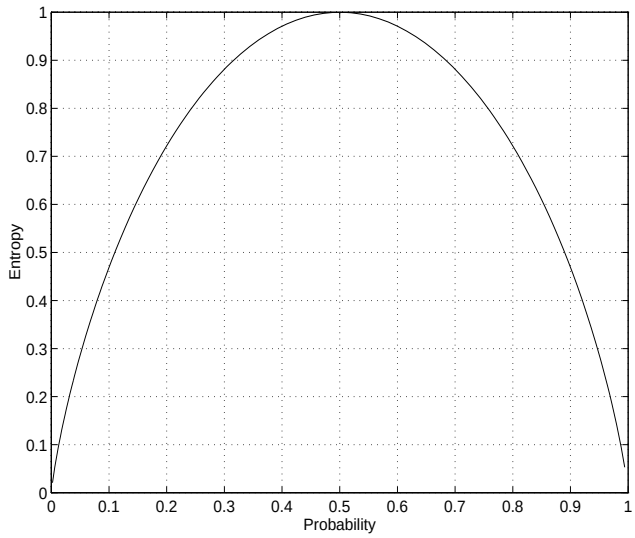


Figure 2.2 Entropy $H(A_2)$ vs. probability

We pointed out earlier that self-information of an event increases as its uncertainty grows. Thus, entropy may be also regarded as a *measure of uncertainty* in the sense that the larger the entropy, the less certain the outcome of a random experiment in $\mathcal{A}_n$. Entropy is also a measure of the *information* contained in each event A_i (given by $I(A_i) = -\log(p_i)$).

Note: Information and uncertainty have the same quantitative measure (i.e. entropy), but different meanings. As information increases, the uncertainty decreases and vice versa. Indeed, uncertainty is equal to the amount of information which is needed to make the outcome of an experiment known.

First, we should note that $\log(p_k) \leq 0$ since $0 < p_k \leq 1$ and so $H(\mathcal{A}_n) \geq 0$. Thus, the entropy can never be negative. Second, let $p_1 = 1$, $p_2 = \ldots = p_n = 0$. By convention, $p_k \log(p_k) = 0$ when $p_k = 0$ and hence, in this case, $H(\mathcal{A}_n) = 0$. Conversely, $H(\mathcal{A}_n) = 0$ implies that $p_k \log(p_k) = 0$ for all k, so that p_k is either 0 or 1. But only one p_k can be unity since their sum must be 1. Hence, entropy is zero if and only if there is complete certainty.

Definition 2.7 (Conditional Entropy) Conditional entropy of some finite field $\mathcal{A}_n$ with probabilities $\{p_i\}_{1 \leq i \leq n}$ with respect to $\mathcal{B}_m$ (with probabilities $\{q_i\}_{1 \leq i \leq m}$) is defined as:

$$H(\mathcal{A}_n | \mathcal{B}_m) = -\sum_{j=1}^{n} \sum_{k=1}^{m} p_{jk} \cdot \log(p_{jk} / q_k) \tag{2.8}$$

where p_{jk} is the joint probability of events A_j and B_k. In other words, conditional entropy refers to the uncertainty left about $\mathcal{A}_n$ when $\mathcal{B}_m$ is known.

Definition 2.8 (Joint Entropy) Given two finite fields $\mathcal{A}_n$ and $\mathcal{B}_m$, their joint entropy is defined as:

$$H(\mathcal{A}_n \times \mathcal{B}_m) = -\sum_{j=1}^{n} \sum_{k=1}^{m} p_{jk} \cdot \log(p_{jk}) \tag{2.9}$$

Based on these two concepts, one can find the information *shared* by two complete sets of events:

$$I(\mathcal{A}_n; \mathcal{B}_m) = H(\mathcal{A}_n) + H(\mathcal{B}_m) - H(\mathcal{A}_n \times \mathcal{B}_m)$$

which is called *mutual information* (or *transinformation*). Moreover, by using the above definitions, one can show that:

$$I(\mathcal{A}_n;\mathcal{B}_m) = H(\mathcal{A}_n) - H(\mathcal{A}_n|\mathcal{B}_m) \tag{2.10}$$

The Venn diagram for these relations is shown in Figure 2.3:

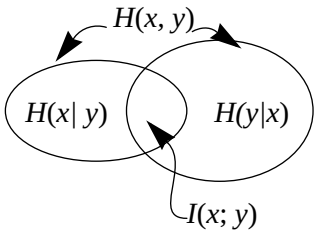


Figure 2.3 A set-theoretic representation of different entropies

Entropy satisfies the following basic properties [47]:

Proposition 2.3 Entropy is maximized when all events are equiprobable, that is:

$$H(p_1, p_2, \ldots, p_n) \le H\left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right) = \log(n) \tag{2.11}$$

Proposition 2.4 If $\mathcal{A}_n$ and $\mathcal{B}_m$ are any two independent finite fields, $\mathcal{A}_n \times \mathcal{B}_m$ a new finite field with *nm* events, then

$$H(\mathcal{A}_n \times \mathcal{B}_m) = H(\mathcal{A}_n) + H(\mathcal{B}_m) \tag{2.12}$$

Shannon's entropy is equally applicable to partitioned sets of events. More precisely, given a partitioning $\prod = \{A_1,A_2,...,A_n\}$ on the set of events, the entropy of this partitioning is:

$$H(\Pi) = -\sum_{i=1}^{n} p(A_i)\log p(A_i) \tag{2.13}$$

where $p(A_i)$ is the probability of class A_i in partition $\prod$.

2.2.2 Informational energy

Assuming that we have a complete set of events $\mathcal{A}_n$ (as in equation (2.5)), we may regard the probability p_k as the *information* associated with the individual event A_k. Hence, as in the case of self-information, we may define an average measure for the set $\mathcal{A}_n$:

Definition 2.9 (Informational Energy) The global information of the finite field $\mathcal{A}_n$ (denoted by $E(\mathcal{A}_n)$) may be expressed by its *informational energy* (also called the *Gini Function*[1]) as:

$$E(\mathcal{A}_n) = \sum_{j=1}^{n} p_j^2 \tag{2.14}$$

We plot in Figure 2.4 the informational energy of a Boolean variable as a function of its signal probability, that is $E(\mathcal{A}_2) = p^2 + (1-p)^2$.

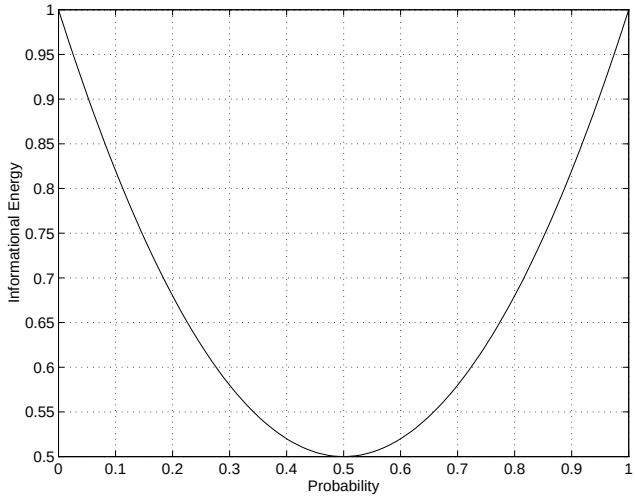


Figure 2.4 Informational energy $E(\mathcal{A}_2)$ vs. probability

1. This was first used by Corrado Gini in a study from "Atti del R. Ist. Veneta di Scienza," *Lettere ed Arti*, 1917, 1918, v. LXXVII

Due to its simplicity, the informational energy was used mainly in statistics (not necessarily in conjunction with Shannon's entropy) as a characteristic of a distribution (discrete as is the case here, or continuous in general). However, without a precise theory, its use was rare until its usefulness was proved [45].

The value of the informational energy is always upper-bounded by 1. This is because $\sum_{j=1}^{n} p_j^2 \leq \left(\sum_{j=1}^{n} p_j \right)^2 = 1$ with equality iff one of the events has probability 1 and the rest of them have the probability 0. Thus, $E(\mathcal{A}_n) = 1$ iff the experiment provides the same determinate and unique result. We give in the following propositions a few basic properties satisfied by the informational energy [46].

Proposition 2.5 The informational energy becomes $1/n$ when all events are equally likely and 1 when one of the events in $\mathcal{A}_n$ is certain:

$$E(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}) = \frac{1}{n} \leq E(\mathcal{A}_n) \tag{2.15}$$

Proposition 2.6 If the uniformity (or the uncertainty) of the system increases, then its informational energy decreases.

Proposition 2.7 If $\mathcal{A}_n$ and $\mathcal{B}_m$ are two independent finite fields, then $E(\mathcal{A}_n \times \mathcal{B}_n) = E(\mathcal{A}_n) \cdot E(\mathcal{B}_n)$.

2.3 Summary

This chapter has presented the main theoretical concepts and notation that will be used throughout the thesis. More precisely, the mathematical foundation of the approaches presented in Chapter 3 and Chapter 5 relies on probabilistic modeling, while the

information theory concepts are useful for the better understanding of the approach presented in Chapter 4.

Chapter 3 Circuit and Gate-Level Power Analysis

3.1 Introduction

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 same point they become contradictory. General simulation techniques can provide sufficient accuracy, but the price we have to pay 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. It has been pointed out in [35][36] the importance of correlations not only inside the target circuit (due to reconvergent fanout regions), but also at the circuit inputs. From this perspective, this part of our work shifts the focus from 'the circuit problem' (induced by reconvergent fanout) to 'the input problem' (or input correlations)

To illustrate these issues, let us consider a simple example: suppose that a 4-bit ripple-carry adder is fed successively by two input sequences S_1 and S_2, as it is shown in Figure 3.1(a). To estimate the total power consumption of the circuit (in a gate level implementation), we have to sum over all the gates in the circuit the average power dissipation due to the capacitive switching currents (as in equation (1.1)). Thus, the average switching activity per node (gate) is a key parameter that needs to be determined correctly.

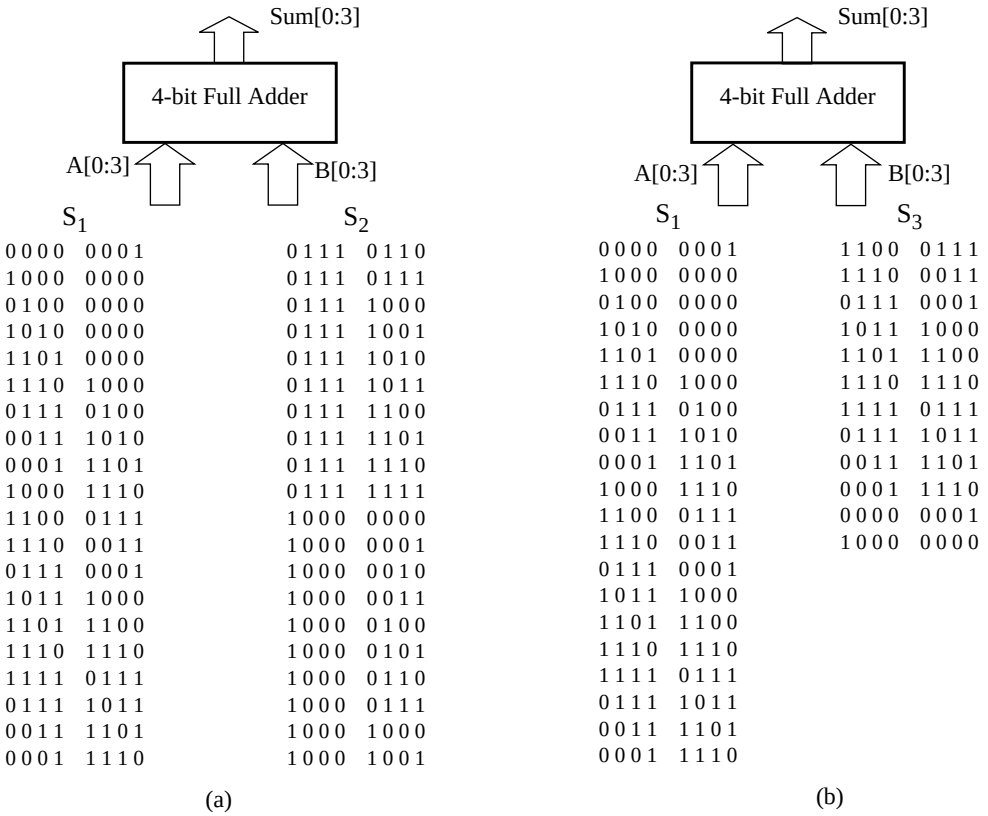


Figure 3.1 An example

The two sequences in Figure 3.1(a), 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 value of total power consumption. On the other hand, sequences S_1 and S_3 shown in Figure 3.1(b) have different lengths (S_3 is 40% shorter than S_1) but lead to very similar statistics on the bit lines. This leads to P_{S3} = 446 μW, a value which is only 2% difference 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 inputs to be applied to the circuit under consideration.

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 [2], are of little or no help here. The reason is the set of input statistics which must be preserved or reproduced during sequence generation for use by power simulators, is quite complex. One such attempt is [40] 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 Figure 3.1 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 used as an incrementor in the address calculation unit of a memory chip, then primary inputs A and B will likely receive sequences like S_2; on the other hand, if this adder was to be part of a DSP system used for noise analysis, then its inputs 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 an accurate

power value 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.

Over the years, many important problems in sequential circuit synthesis and optimization have been approached using concepts from automata theory. Finite automata 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, exhibit stochastic behavior because of random malfunctioning of components [61].

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 chapter, 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 Figure 3.2(b).

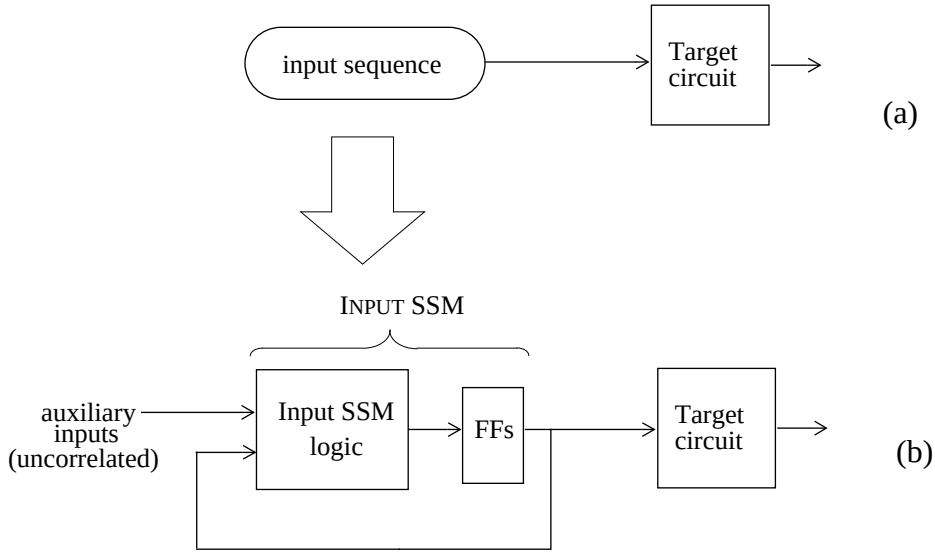


Figure 3.2 Constrained sequence generation

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.

To conclude, both simulation-based approaches and probabilistic techniques for power estimation may benefit from this research. The issues brought into attention in this chapter 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.

This chapter is organized as follows: Section 3.2 introduces some basic definitions from SSM theory and gives the main decomposition theorems used in SSM synthesis.

Section 3.3 discusses the constrained sequence generation problem while Section 3.4 gives a practical procedure for sequence compaction. Section 3.5 is devoted to practical considerations and experimental results. Finally, we conclude by summarizing our main contribution.

3.2 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 [50].

3.2.1 Stochastic machines: basic definitions

Definition 3.1 (Mealy-type SSM) A Mealy-type SSM is a quadruple $\mathcal{M} = (S, X, Y, \{A(x, y)\})$ 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

$$\sum_{y \in Y} \sum_{j=1}^{|S|} a_{ij}(y|x) = 1 \qquad \text{where} \qquad A(y|x) = [a_{ij}(y|x)] \tag{3.1}$$

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 3.2 Let $\mathcal{M}$ 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.

Example 3.1 Let $\mathcal{M} = (S, X, Y, \{A(y|x)\})$ with $X = \{0, 1\}$, $Y = \{a, b\}$, $S = \{s_1, s_2\}$ and

$$A(a|0) = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \qquad A(b|0) = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{2} & 0 \end{bmatrix} \qquad A(a|1) = \begin{bmatrix} 0 & \frac{1}{2} \\ 0 & 0 \end{bmatrix} \qquad A(b|1) = \begin{bmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$ and let

$\pi = \left(\frac{1}{4} \ \frac{3}{4}\right)$ be an initial distribution for $\mathcal{M}$. First, we can see that $\mathcal{M}$ is correctly defined, that is, equation (3.1) is verified for every possible initial state. Inspecting for instance the first row in matrices $A(a|0)$ and $A(b|0)$, we observe that machine $\mathcal{M}$, initially in state s_1 and fed with symbol 0, will remain in state s_1 and produce a symbol a with probability 1/2, will remain in state s_1 and produce a symbol b with probability 1/4, or will go to state s_2 and generate a symbol b with probability 1/4. Also, from Definition 3.2 we have that:

$$A(ab|00) = A(a|0)\cdot A(b|0) = \begin{bmatrix} \frac{1}{8} & \frac{1}{8} \\ \frac{1}{4} & 0 \end{bmatrix}.$$

This means for instance, that the probability of the machine outputting the word ab, having been in state s_1 and fed the word 00 on the input, is $\frac{1}{8} + \frac{1}{8} = \frac{1}{4}$ (irrespective of the final state of the machine).

Definition 3.3 (Moore-type SSM) A Moore-type SSM is a quintuple $\mathcal{M} = (S, X, Y, \{A(x)\}, \Lambda)$ where S, X, and Y are as in Definition 3.1, $\{A(x)\}$ 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 3.4 Let $\mathcal{M}$ be an SSM. Let $A(u) = [a_{ij}(u)] = A(x_1) \cdot A(x_2) \cdot \ldots \cdot A(x_k)$; 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.

Example 3.2 Let $\mathcal{M} = (S, X, Y, \{A(x)\}, \Lambda)$ with $S = \{0, 1\} = X = Y$, $\Lambda(0) = 1$, $\Lambda(1) = 0$, and the following transition matrices: $A(0) = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{3}{4} \end{bmatrix}$ $A(1) = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$. We observe that machine $\mathcal{M}$, initially in state 0 and fed with symbol 0, will remain in state 0 and produce a symbol 1 with probability 1/2 or will go to state 1 and generate a symbol 0 with probability 1/2; on the other hand, if fed with symbol 1, it will remain in state 0 with probability 2/3 or go to state 1 with probability 1/3, generating a symbol 1 and 0,

respectively. From Definition 3.4, we have that $A(00) = A(0) \cdot A(0) = \begin{bmatrix} \frac{3}{8} & \frac{5}{8} \\ \frac{5}{16} & \frac{11}{16} \end{bmatrix}$. This means for instance, that starting in state 0 and fed the sequence 00 on the input, the probability that the machine goes to state 1 in two steps (with output 0) is 5/8.

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. 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.

3.2.2 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 3.1 [11] Any $m \times n$ stochastic matrix A can be expressed in the form $A = \sum p_i \cdot U_i$ where $p_i > 0$, $\Sigma p_i = 1$, 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 $m(n\text{-}1) + 1$.

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 shall see later, it gives the basis to efficiently apply Theorem 3.1 on large matrices which may arise in practice.

Theorem 3.2 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 $A = \sum_{i=1}^{t} p_i \cdot U_i$ according to the above result. That means that either A has been decomposed using exactly t matrices U_i, or that considering only the first t matrices in this decomposition has been satisfactory for the 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 3.1 (see Section 3.4).

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 Figure 3.3 a simplified block diagram of the network which synthesizes such a machine.

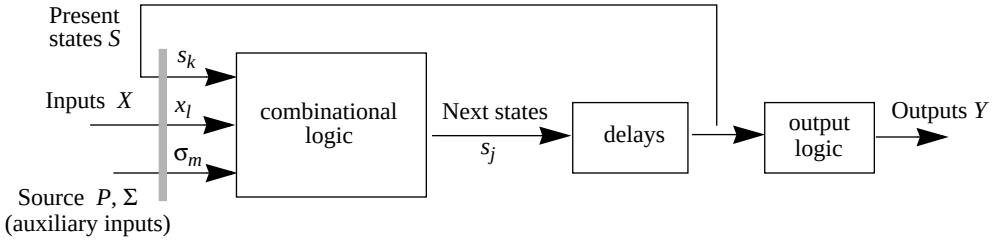


Figure 3.3 The general structure of a SSM

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 3.1 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 3.3 Let's synthesize now the stochastic machine $\mathcal{M}$ defined in Example 3.2.

Putting together $A(0)$ and $A(1)$ we have the whole transition matrix A as:

$A = \begin{bmatrix} A(0) \\ A(1) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{3}{4} \\ \frac{2}{3} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$. Applying Theorem 3.1, 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.

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

Using standard Karnaugh approach, we obtain the circuit which synthesizes the given SSM, as shown in Figure 3.4.

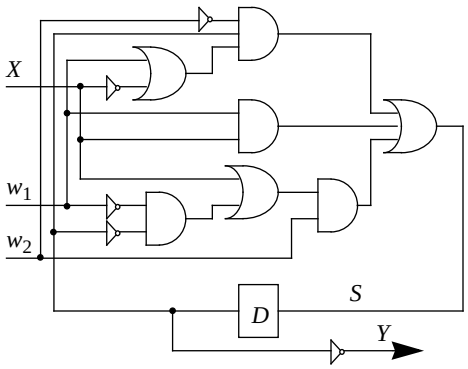


Figure 3.4 An example of SSM

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 3.1, involves essentially the synthesis of a combinational circuit with feedback, and construction of information sources with prescribed probability distributions.

3.3 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 bounds that demonstrate the possibility

of using different input sequences while still having the same total power consumption in the target circuit.

3.3.1 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 introduction the reader is referred to [50]. For practical purposes, we restrict our attention only to reduced stochastic machines of Moore-type.

Definition 3.5 Two reduced stochastic machines $\mathcal{M}$ and $\mathcal{M}$* (as in Definition 3.3) 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 $\mathcal{M}$ there corresponds a state s_j of $\mathcal{M}^*$, 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 $\mathcal{M}$ and $\mathcal{M}^*$ as follows:

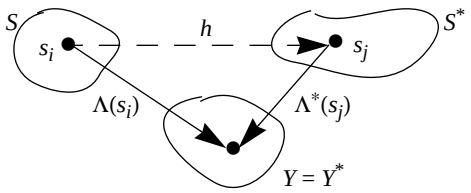


Figure 3.5 Output equivalence

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 3.6 The output sequence Y generated by machine $\mathcal{M}$ is ε-*equivalent* with the output sequence Y^* produced by $\mathcal{M}^*$ if $\|A - A^*\| < \varepsilon$, where the norm is defined as $\|A\| = max|a_{ij}|$. (We note the particular case ε = 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 $\mathcal{M}$, we need to know how close is the sequence generated by $\mathcal{M}^*$ 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 3.1 with limited precision.

3.3.2 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 [47] 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 $A^T \cdot p = \lambda \cdot p$.)

Theorem 3.3 [63] Every stochastic matrix has 1 as a simple[1] eigenvalue and all other eigenvalues have absolute values less than one. ■

This is a consequence of the Perron-Frobenius theorem which states that for every matrix with nonnegative entries, there exists a simple, positive eigenvalue greater than the absolute value of any other eigenvalue. Since the proof of this theorem is very intricate, we refer the reader to reference [63]. However, Theorem 3.3 is very important for us because it makes possible to analyze perturbations on matrix A. To this effect, let's assume that the

1. This means that the multiplicity of the root $\lambda = 1$ in the equation $A^T \cdot p = \lambda \cdot p$ is one.

newly generated sequence is characterized by the matrix $A^* = [a_{ij}^*]$ where $a_{ij}^* = a_{ij} + \varepsilon_{ij}$ (ε_{ij} represents the error induced by perturbations) and $|\varepsilon_{ij}| < 1$. We can write $A^* = A + \varepsilon \cdot B$ where $\varepsilon = max|\varepsilon_{ij}|$ and $b_{ij} = \frac{\varepsilon_{ij}}{\varepsilon}$. Because A^* characterizes a sequence of vectors, it is also a stochastic matrix and therefore, as stated in Theorem 3.3, 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 3.4 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 $k_1\varepsilon + k_2\varepsilon^2 + ...$ ■

This theorem follows from the theory of algebraic functions developed in [63]. Since $\|a_{ij} - a_{ij}^*\| = 0(\varepsilon)$, it is easy to see that:

Corollary 3.1 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 $\|p_{i \to j} - p_{i \to j}^*\| = 0(\varepsilon)$.

Proof: Follows immediately from Theorem 3.4. ■

We have thus proved that we can asymptotically reproduce an initial sequence by preserving its matrix *A*. From a practical point of view, let's 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 3.2 If P and P^* are the values of the total power consumption for two sequences satisfying the conditions in Corollary 3.1, then we have that $|P^* - P| = 0(\varepsilon)$.

Proof: We have $P_{avg} = \frac{f_{clk}}{2} \cdot V_{DD}^2 \cdot \sum_k (C_k \cdot sw_k)$ where C_k is the output capacitance of gate k and sw_k is the average switching activity at the output of gate k. Since sw_k can be written as: $sw_k = \sum_{i,j} p_{i \to j} \cdot n_{ij}^k$, where $p_{i \to j}$ is the transition probability of going from input vector v_i to input vector v_j and n_{ij}^k is the number of transitions at the output of gate k when vector v_i is followed by vector v_j at the input of the circuit, we can write the total power consumption as $P = \frac{V_{dd}^2}{2 \cdot T_{cycle}} \cdot \sum_{i,j,k} p_{i \to j} \cdot C_k \cdot n_{ij}^k$. Assuming that the input sequence is approximated by another input sequence such that the new set of transition probabilities satisfies $\left\| p_{i \to j} - p_{i \to j}^* \right\| = 0(\varepsilon)$, then the error made in the value of total power consumption is given by $|P^* - P| \le \frac{V_{dd}^2}{2 \cdot T_{cycle}} \cdot \sum_{i,j,k} \left| p_{i \to j}^* - p_{i \to j} \right| \cdot C_k \cdot n_{ij}^k = 0(\varepsilon)$. ■

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

We should point out that the above proof is valid only for combinational circuits. For sequential circuits the same result applies if we replace the transition probability on the primary inputs with the transition probability on the primary inputs *and* state lines (jointly). As we shall see in Chapter 5, if the primary inputs of a FSM can be modeled as a lag-k Markov chain [47], then the lag-k Markov chain on the primary inputs and state lines

is also preserved. Since in this chapter we assume that the primary inputs of the target circuit are modeled as lag-one Markov chains, Corollary 3.2 is also valid for sequential circuits.

Example 3.4 Let's consider the circuit in Figure 3.6 fed by the upper sequence which contains only vectors '01' and '11' with characteristics given by the matrix $A = \begin{bmatrix} 0.5 & 0.5 \\ 0.25 & 0.75 \end{bmatrix}$ (i.e. for example, the conditional probability of going from '01' to '11' is 0.5 and from '11' to '01' is 0.25).

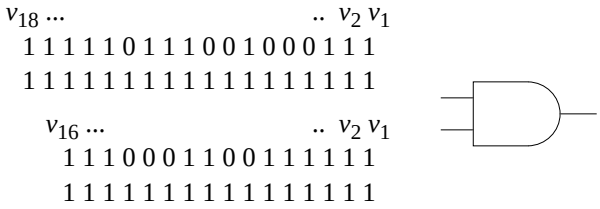


Figure 3.6 A simple example of compaction

For this matrix, by solving the Chapman-Kolmogorov equations, we get the vector of state probabilities $p = \begin{bmatrix} 0.33 \\ 0.67 \end{bmatrix}$ (that is, the state probability is 0.33 for '01' and 0.67 for '11'). Accordingly, the transition probabilities for the initial sequence are: $p_{1\to 1} = a_{11} \cdot p_1 = 0.33 \cdot 0.5 = 0.167$ and similarly $p_{1\to 3} = 0.33 \cdot 0.5 = 0.167$, $p_{3\to 1} = 0.67 \cdot 0.25 = 0.167$, $p_{3\to 3} = 0.67 \cdot 0.75 = 0.5$.

Let's investigate the effect of a perturbation of matrix A such that the new matrix A^* has the form $A^* = \begin{bmatrix} 0.5-\varepsilon_1 & 0.5+\varepsilon_1 \\ 0.25+\varepsilon_2 & 0.75-\varepsilon_2 \end{bmatrix} = A + \varepsilon \cdot B$, where $\varepsilon = max(\varepsilon_1, \varepsilon_2)$ and B has all

its elements less than 1 in absolute value. In this case, the state probability vector becomes $p^* = \begin{bmatrix} \frac{0.33 + 1.33 \cdot \varepsilon_2}{1 + 1.33 \cdot (\varepsilon_1 + \varepsilon_2)} \\ \frac{0.67 + 1.33 \cdot \varepsilon_1}{1 + 1.33 \cdot (\varepsilon_1 + \varepsilon_2)} \end{bmatrix}$. The condition $\|p - p^*\| = 0(\varepsilon)$ in Theorem 3.4 is satisfied for any values of ε_1 and ε_2 if the Taylor expansion around point zero of $\frac{1}{1 + 1.33 \cdot (\varepsilon_1 + \varepsilon_2)}$ converges, that is $|1.33 \cdot (\varepsilon_1 + \varepsilon_2)| < 1$ or $\varepsilon = max(\varepsilon_1, \varepsilon_2) < \frac{1}{2 \cdot 1.33} = 0.375$.

For example, if the circuit is fed in turn by the lower sequence in Figure 3.6, then the corresponding stochastic matrix and state probability vector become $A^* = \begin{bmatrix} 0.6 & 0.4 \\ 0.18 & 0.82 \end{bmatrix}$ and $p^* = \begin{bmatrix} 0.31 \\ 0.69 \end{bmatrix}$ (thus, $\varepsilon_1 = 0.1$ and $\varepsilon_2 = 0.07$). For this set of values, the transition probabilities are: $p^*_{1 \to 1} = 0.1875$, $p^*_{1 \to 3} = 0.125$, $p^*_{3 \to 1} = 0.125$, and $p^*_{3 \to 3} = 0.5625$.

Accordingly, the total power consumption of the circuit is initially $P = \frac{C \cdot V_{dd}^2}{T_{cycle}} \cdot (p_{1 \to 3} + p_{3 \to 1}) = \frac{C \cdot V_{dd}^2}{T_{cycle}} \cdot 0.33$ and becomes $P^* = \frac{C \cdot V_{dd}^2}{T_{cycle}} \cdot 0.25$. Thus, we get $|P - P^*| = \frac{C \cdot V_{dd}^2}{T_{cycle}} \cdot 0.08$.

3.4 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 3.2 for the synthesis of the stochastic machine $\mathcal{M}$. 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.

3.4.1 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 3.5 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:

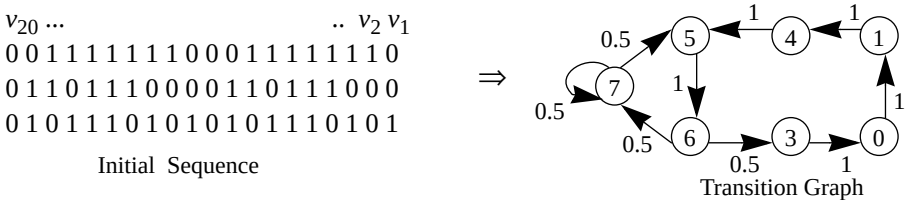


Figure 3.7 A sequence and its transition graph

In the right side of Figure 3.7, 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 the initial 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 $\mathcal{M}$ be the SSM associated with this sequence; as we can see, $S = \{0, 1, 3, 4, 5, 6, 7\}$ and then $|S| = 7$. Right now, we are trying to synthesize a new machine $\mathcal{M}^*$, output-equivalent with $\mathcal{M}$, and eventually to generate an equivalent (and compacted) sequence with the initial one, using $\mathcal{M}^*$. To make our job easier, let’s assume also that $Y = S$ and $Y^* = S^*$ ($\mathcal{M}$ and $\mathcal{M}^*$ 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 3.5, 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 3.1:

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

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

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:

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

A possible implementation for $\mathcal{M}^*$ (with D Flip-Flops) is given in Figure 3.8. 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).

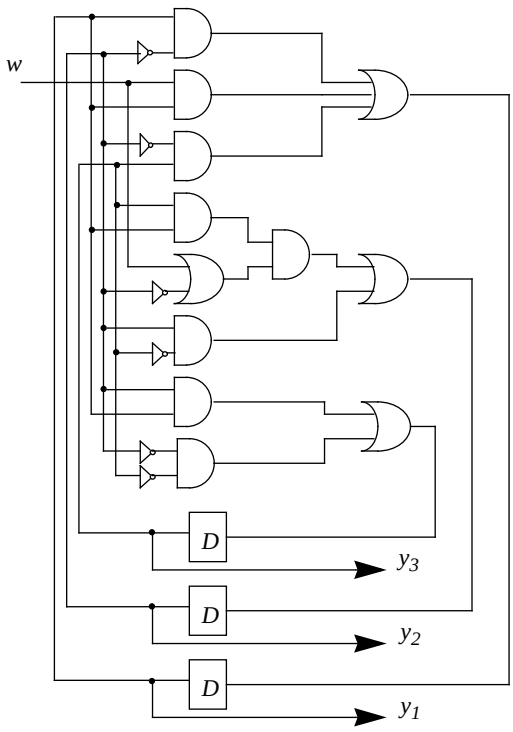


Figure 3.8 The SSM generating the initial sequence

To generate a sequence, the SSM $\mathcal{M}$* 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:

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

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 $\mathcal{M}^*$, the initial stochastic characteristics are preserved exactly; indeed from Definition 3.6 with ε = 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 Figure 3.8, we would have obtained other output sequences, but all of them still satisfying the inequality from Definition 3.6.

Remark: Decomposing the initial matrix *A* (via Theorem 3.1), 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 $\mathcal{M}$;

- 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 significantly simplified.

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

```
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 3.1 and Theorem 3.2 */
   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;
```

3.4.2 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 or not a set of bits are in the same block 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 $\sum_{1 \le p < q \le b} \sum_{\substack{x \in G_p \\ y \in G_q}} \alpha(x, y) \cdot cost(x, y)$ 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 (α = 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 [24][12].

Example 3.6 Let us consider the input sequence in Figure 3.7, and let *x, y, z* be the 3 bits for representing $v_1, v_2,...,v_{20}$ that feed the simple circuit in Figure 3.9(a); for this sequence, we get the following values of individual transition probabilities:

Transition Prob.	*x*	*y*	*z*
0->0	0.2	0.3	0
0->1	0.1	0.2	0.4
1->0	0.1	0.2	0.4
1->1	0.6	0.3	0.2

For the pair (*x*, *y*), we have the pairwise transition probabilities: $p(x_{0->0}\, y_{0->0}) = 0.1$, $p(x_{0->1}\, y_{0->0}) = 0.1$, and so on. 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* (Figure 3.9(c)) assigning to each bit a vertex and weighting the edges with the cost function (α coefficients are α(*x*,*y*) = α(*y*,*z*) = 1/2, α(*x*,*z*) = 1/4).

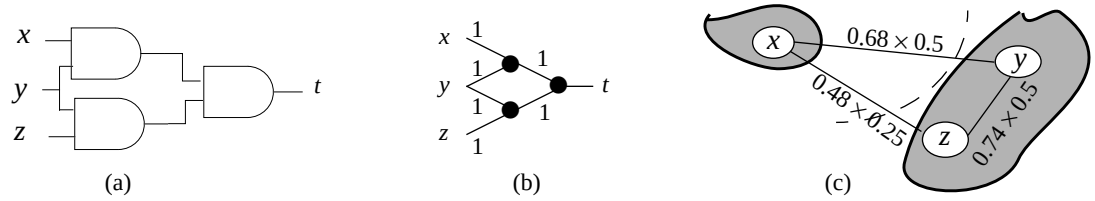


Figure 3.9 An example of a bit-dependency graph

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.

3.5 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.

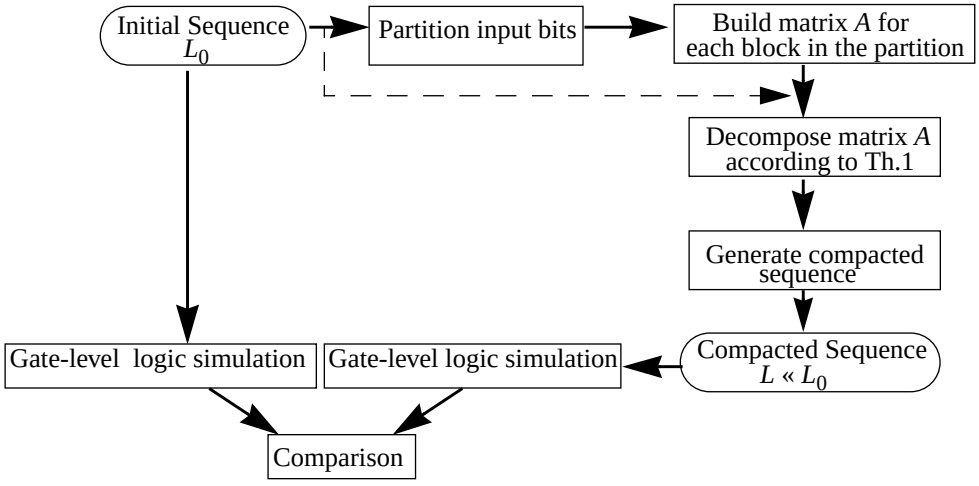


Figure 3.10 The experimental setup

Our strategy is depicted in Figure 3.10 and follows the steps of the algorithm in Section 3.4. 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* [56], that is the process of transforming an input sequence into a smaller one, such that the new body of data represents a *good approximation* to 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.

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 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 managed as a whole, the set of input bits is partitioned into b subsets (blocks) using the Kernighan-Lin heuristic as described in Section 3.4. To each block, we then associate a stochastic machine (SSM_1, SSM_2,...,SSM_b in Figure 3.11(b)). 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.

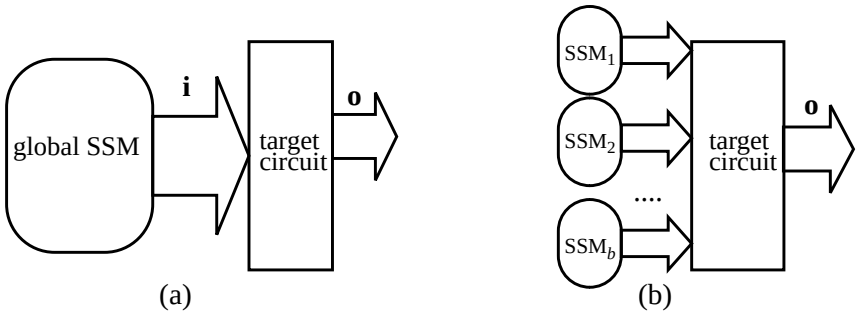


Figure 3.11 Two possible strategies

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 3.4 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 3.1. 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 Figure 3.11(a)) 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 3.1 and Table 3.2 we provide our results for type 1 sequences of length L_0=100,000 compacted with different compaction ratios (namely r = 50, 100 and 1000) using the strategy in Figure 3.11(b); 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 Figure 3.11(b)), 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 3.1. Total power (μW@20MHz) for type 1 sequences (zero delay)

			Power Compaction ratio 50		Power Compaction ratio 100		Power Compaction ratio 1000	
Circ.	#Inp.	Exact Power	$k = 4$	$k = 6$	$k = 4$	$k = 6$	$k = 4$	$k = 6$
C1355	41	3354.60	3363.00	3352.20	3373.20	3349.80	3438.00	3297.60
C1908	33	4919.40	4870.80	4954.80	4836.60	4930.20	4724.40	4922.40
C3540	50	11810.40	11757.60	11713.20	11707.20	11591.40	11662.20	11200.20
C432	36	2473.80	2463.60	2441.40	2475.60	2460.60	2426.40	2476.80
C499	41	4305.00	4315.20	4296.00	4325.40	4292.40	4405.20	4186.80
C6288	32	42499.80	42879.00	42421.20	42935.40	42434.40	43686.00	41299.20
C880	60	4875.60	4935.60	4879.20	4975.80	4861.80	4974.00	4794.60
s1196	14	6488.40	6323.40	6350.40	6394.80	6374.40	6666.60	6006.00
s344	9	1730.40	1611.60	1616.40	1617.60	1635.60	1701.60	1791.00
s641	35	2471.40	2371.80	2392.20	2386.20	2380.80	2343.00	2320.20
s838	34	1479.00	1453.80	1460.40	1470.60	1428.60	1473.00	1369.20
s9234	36	20364.60	20030.40	19670.40	20021.40	19607.40	19596.00	20405.40
	Average Error (%)		1.77	1.66	1.70	1.79	2.55	3.24
s298	3	873.60	873.00		871.80		856.80	
s386	7	1771.20	1768.80		1770.00		1774.20	
	Average Error (%)		0.10		0.14		1.05	

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, both for zero or real-delay models. Thus, for C880 in Table 3.1, instead of simulating 100,000 vectors with an exact power of 4875.60 μW, one can use only 1000 vectors with an estimate of 4861.80 μW ($k = 6$) or just 100 vectors with a power consumption estimated as 4974.00 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.

Table 3.2. Total power (μW@20MHz) for type 1 sequences (real delay)

			Power Compaction ratio 50		Power Compaction ratio 100		Power Compaction ratio 1000	
Circ.	#Inp.	Exact Power	$k = 4$	$k = 6$	$k = 4$	$k = 6$	$k = 4$	$k = 6$
C1355	41	4218.00	4251.00	4232.40	4276.80	4236.00	4336.80	4134.60
C1908	33	6990.00	6970.80	7019.40	6894.00	6966.60	6597.60	6921.00
C3540	50	19603.20	19654.20	19393.80	19497.00	19102.20	19626.60	18646.20
C432	36	3070.80	3054.00	3018.00	3065.40	3024.60	3024.00	3052.20
C499	41	5374.20	5390.40	5374.80	5431.20	5397.00	5530.80	5235.60
C6288	32	347886.0	351669.60	348599.40	354933.60	349987.20	359386.80	336701.40
C880	60	5990.40	6018.60	6021.60	6084.00	5976.60	6117.60	5871.00
s1196	14	7698.60	7492.20	7512.60	7594.20	7452.60	7914.60	6989.40
s344	9	1814.40	1841.40	1849.20	1851.60	1865.40	1975.80	2062.80
s641	35	2908.80	2779.80	2798.40	2805.00	2806.20	2674.80	2713.20
s838	34	1551.00	1538.40	1540.20	1554.60	1503.60	1551.60	1438.20
s9234	36	21693.60	21081.60	21081.60	21679.20	21042.60	21064.20	21875.40
	Average Error (%)		1.13	1.33	1.05	1.81	3.50	3.32
s298	3	975.00	974.40		972.60		954.60	
s386	7	1996.80	2022.60		2023.20		2030.40	
	Average Error (%)		0.68		0.78		1.89	

On the efficiency side, in Table 3.3 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 3.2) if this is satisfactory from a practical point of view.

Table 3.3. CPU time (sec.)

	Partition		Decomposition		Generation r = 50		Generation r = 100		Generation r = 1000	
Circ.	k = 4	k = 6	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	4.42	4.05	2.22	2.02	0.25	0.22
C1908	0.01	0.01	0.08	0.59	3.24	3.00	1.63	1.51	0.19	0.16
C3540	0.03	0.02	0.13	0.84	4.42	4.05	2.22	2.02	0.25	0.22
C432	0.01	0.01	0.09	0.57	3.61	3.32	1.81	1.73	0.20	0.18
C499	0.02	0.01	0.10	0.73	4.42	4.05	2.22	2.02	0.25	0.22
C6288	0.01	0.01	0.07	0.54	3.05	2.88	1.54	1.45	0.17	0.16
C880	0.04	0.02	0.17	1.13	7.85	7.10	3.92	3.55	0.43	0.37
s1196	0.01	0.01	0.03	0.06	1.10	1.01	0.55	0.51	0.06	0.06
s344	0.01	0.01	0.13	0.02	0.27	0.23	0.13	0.11	0.02	0.01
s641	0.01	0.01	0.08	0.60	3.57	3.22	1.75	1.67	0.19	0.17
s838	0.01	0.01	0.08	0.59	3.36	3.10	1.70	1.55	0.18	0.17
s9234	0.01	0.01	0.09	0.57	3.61	3.32	1.81	1.73	0.20	0.18
s298	-		0.01		0.08		0.04		0.01	
s386	-		0.11		0.14		0.07		0.01	

In Table 3.4 and Table 3.5, 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 Figure 3.11(a) 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 3.5.

Table 3.4. Total power (μW@20MHz) for type 2 sequences

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
	Average. Error (%)	6.11	5.06

Table 3.5. Total power (μW@20MHz) for type 3 sequences

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

As reported in Table 3.4 and Table 3.5, the results are still good, the average relative error being around 5% on average. 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 [21]. 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 3.6 we present the results obtained for sequences of type 3 and compaction ratio $r = 200$.

Table 3.6. Node-by-node switching activity analysis for type 3 sequences

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

To derive the values in Table 3.6, 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.

3.6 Summary

In this chapter, 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 on 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 chapter 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.

Chapter 4 High-Level Power Analysis

4.1 Introduction

Having as soon as possible in the design cycle an estimate of power consumption, can save significant redesign efforts or even completely change the entire design architecture. At high levels of abstraction, consistency is more important than accuracy, that is, relative evaluation of different designs for power dissipation is often sufficient. Higher levels of abstraction have been considered in power estimation of digital circuits, but here many problems are still pending a satisfactory solution. The problem of power estimation at the RT-level is different from that at the logic level: whereas at gate level it is desirable to determine the switching activity at each node (gate) in the circuit (Figure 4.1(a)), for RT-level designs an average estimate per module is satisfactory (Figure 4.1(b)). In other words, some accuracy may be sacrificed in order to obtain an acceptable power estimate early in the design cycle and at a significantly lower computational cost.

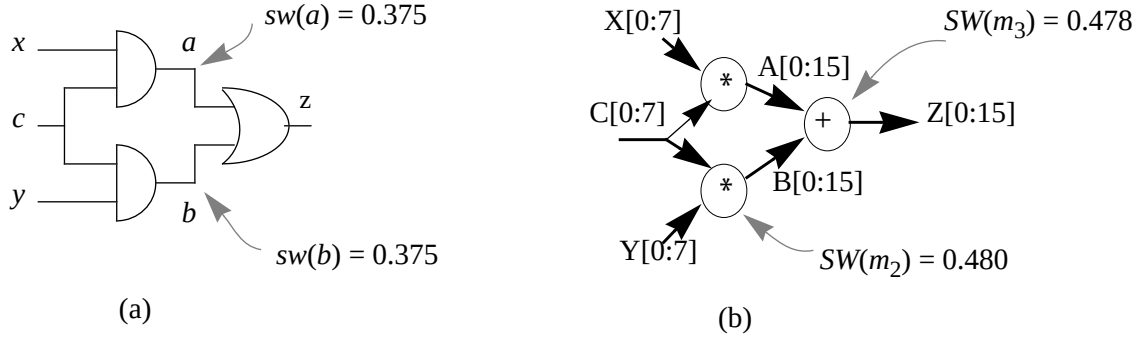


Figure 4.1 Power estimation issues at the logic and RT-levels

In the data-flow graph considered in Figure 4.1(b), the total power consumption may be estimated as: $P_{total} = P_{wires} + P_{modules}$. Usually, the interconnect power consumption is either estimated separately or included in the power consumption of the modules, therefore we can write: $P_{total} \approx \sum_{m_j \in M} P_{m_j} \propto \sum_{m_j \in M} (C_{m_j} \cdot SW_{m_j})$ where the summation is

performed over the set of modules *M used in the data-flow graph*, and C_{m_j}, SW_{m_j} stand for the capacitance loading and the average switching activity of module m_j, respectively. Basically, what we propose is to characterize the average switching activity of a module (SW_{m_j}) through the average switching activity for a *typical signal line* in that module (sw_{avg}). More formally, for a generic module m_j having n internal lines (each characterized by its capacitance and switching activity values c_i and sw_i, respectively), we have:

$$P_{m_j} \propto \sum_{i=1}^{n} (c_i \cdot sw_i) \approx sw_{avg} \cdot \sum_{i=1}^{n} c_i = sw_{avg} \cdot C_{m_j} \tag{4.1}$$

We assume that module capacitances C_{m_j} are either estimated or taken from a library, therefore we concentrate on estimating the average switching activity per module. This is a quite different strategy compared to the previous work. The only other proposed method for power estimation at RT-level is simulative in nature, requires pre-characterization of the modules and may be summarized as follows: first, RT-level simulation is performed to obtain the average switching activity at the inputs of the modules and then, this switching activity is used to "modulate" a switched capacitance value (product of the switching activity and the physical capacitance) which is pre-computed and stored for each module in the library to obtain the power dissipation estimate [27]. Compared to this methodology, the distinctive feature of the present approach, is that *it does not require simulation*; its predictions are based only on the

characteristics of the input sequence and some knowledge about the function and/or structure of the circuit (see Section 4.4 for details).

In this chapter, we address the problem of power estimation at the RT-level from an information theoretical point of view [3]. Traditionally, entropy has been considered a useful measure for solving problems of area estimation [20][8], timing analysis [6] and testing [1][57]. We propose two new measures for estimating the power consumption of each module based on *entropy* and *informational energy*. Our entropy/informational energy-based measures simply provide an approximation for the functional activity in the circuit without having to necessarily simulate the circuit. With some further simplifications, simple closed form expressions are derived and their value in practical applications is explored.

Note: We point out that, although this research targets RT-level and behavioral design, it also presents as a by-product, a technique applicable to logic level designs. This is a first step in building a unified framework for power analysis from gate level to behavioral level.

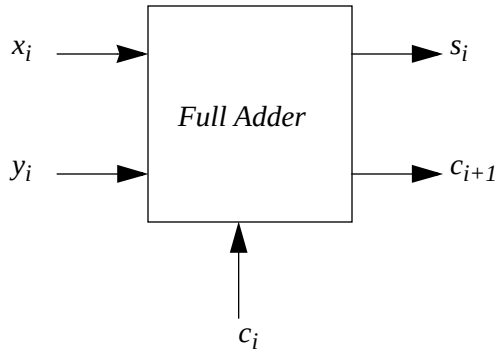
4.2 Theoretical framework

4.2.1 An entropy based approach

The applications of entropy fall in two categories. The first category consists of problems related to the determination of unknown distributions based on the principle of maximum entropy; that is, determining the distribution of some partition of events subject to given constraints (statistical mechanics). In the second category (coding

theory), we are given the source entropy and wish to construct various random variables (code lengths) so as to minimize their expected values. In this research, we focus on the relationship between the information (entropy) contained in a binary stream and the resulting power consumption when this binary stream is applied to the inputs of a combinational logic block.

Example 4.1 The truth table for a randomly excited 1-bit full adder is given below:



c_i	x_i	y_i	s_i	c_{i+1}
0	0	1	1	0
1	0	0	1	0
1	1	0	0	1
1	1	1	1	1
0	1	1	0	1
1	0	1	0	1
0	1	0	1	0
0	0	0	0	0

Class A_1

Figure 4.2 1-bit full adder

where x_i, y_i are the inputs, c_i is carry-in, s_i is the sum bit and c_{i+1} is carry-out. The output space is partitioned in four classes as $\prod = \{A_1,A_2,A_3,A_4\} = \{10, 01, 11, 00\}$, where $p(A_1) = p(A_2) = 3/8$, $p(A_3) = p(A_4) = 1/8$; applying equation (2.13) we obtain the entropy of the partition Π $H(\prod) = 1.8113$. We observe that within a class there is no activity on the outputs; this means that output transitions may occur only when one has to cross class boundaries in different time steps. If the output sequence is a purely random one, then exactly H bits are needed to represent the output sequence; therefore the average number of transitions per word (or average switching activity per word) will be $H/2$. In any other nonrandom arrangement, for a minimum length encoding scheme, the average number of

transitions per word will be $\leq H/2$, so in practice, $H/2$ can serve as a conservative upper bound on the number of transitions per word. In our example, we find an average switching value approximately equal to 0.905 which matches fairly well the exact value 1 deduced from the above table. Such a measure was suggested initially by Hellerman to quantify the *computational work* of simple processes [20].

More formally, if a signal x is modeled as a lag-one Markov chain with conditional probabilities p_{00}, p_{01}, p_{10}, p_{11} and signal probabilities p_0 and p_1 as in Figure 4.3, then we can characterize it through the conditional entropy between 2 successive steps in time as:

$h\,(x^+|x^-) = -p_1\,(p_{10}\log p_{10} + p_{11}\log p_{11}) - p_0\,(p_{00}\log p_{00} + p_{01}\log p_{01})$.

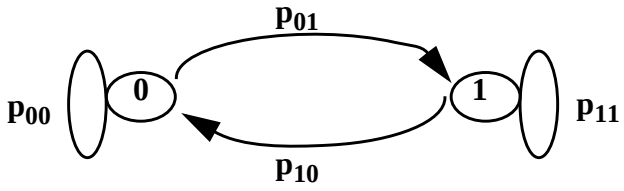


Figure 4.3 A two-state Markov-chain modeling a signal line x

The signal probabilities can be expressed in terms of conditional probabilities as: $p_1 = \frac{p_{01}}{p_{01} + p_{10}}$ and $p_0 = \frac{p_{10}}{p_{01} + p_{10}}$, respectively [35]. Using the well-known identity $-\ln(1-a) = \sum_{k=1}^{\infty} \frac{a^k}{k}$ for $0 < a < 1$, we obtain:

$$h\,(x^+|x^-) = \frac{1}{2 \cdot \ln(2)} \cdot \frac{2 \cdot p_{01} \cdot p_{10}}{p_{01} + p_{10}} \cdot \sum_{k=1}^{\infty} \left[\frac{(p_{00}^{k-1} + p_{01}^{k-1}) \cdot p_{00} + (p_{10}^{k-1} + p_{11}^{k-1}) \cdot p_{11}}{k} \right].$$

We note that $\frac{2 \cdot p_{01} \cdot p_{10}}{p_{01} + p_{10}}$ is exactly the switching activity of line x (denoted by $sw\,(x)$) and that $a^{k-1} + b^{k-1}$ (for $a + b = 1$) is minimum when $a = b = 0.5$. Thus, $h\,(x^+|x^-) \geq$

$\frac{1}{2 \cdot \ln(2)} \cdot sw(x) \cdot \sum_{k=1}^{\infty} \frac{(p_{00} + p_{11}) \cdot \frac{1}{2^{k-2}}}{k}$ or, using again the above identity: $h(x^+|x^-) \geq 2 \cdot sw(x) \cdot (p_{00} + p_{11})$. So, what we got is an upper bound for the switching activity of signal x in the most general case when it is modeled as a lag-one Markov chain: $sw(x) \leq h(x^+|x^-) / [2 \cdot (p_{00} + p_{11})]$. Unfortunately, there is no relationship between p_{00} and p_{11} and their sum can take any value from the interval [0; 1]. To obtain an upper bound useful in practice, we assume *temporal independence*, that is: $p_1 = p_{01} = p_{11}$, $p_0 = p_{10} = p_{00}$ and $h(x^+|x^-) = h(x)$. In this simpler case, $p_{00} + p_{11} = 1$ and hence the relationship between $sw(x)$ and $h(x)$ is exactly the one based on intuition: $sw(x) \leq h(x) / 2$.

To evaluate H, one can use basic results from information theory concerning transmission of information through a module. More precisely, for a module with input X and output Y, we have $I(X;Y) = H(X) - H(X|Y)$ and by symmetry $I(X;Y) = H(Y) - H(Y|X)$ due to the commutativity property of mutual information. When input X is known, no uncertainty is left about the output Y and thus $H(Y|X)$ is zero. Therefore, the information transmitted through a module can be expressed as $H(Y) = H(X) - H(X|Y)$ which represents the amount of information provided about X by Y. For instance, in Figure 4.2 $I(X; Y) = H(Y) = 3 - 1.1887 = 1.8113$; thus, informally at least, the observation of the output of the module provides 1.8113 bits of information about the input, on average. However, in real examples, this type of characterization is essentially very expensive as long as the input/output relation is not a one-to-one mapping. This usually ends up in intricate analytical calculations; for instance, the exact

calculation of the output entropy of an *n*-bit adder, would require the knowledge of joint input/output probabilities and a double summation with 2^{2n} terms (as in equation (2.8)).

For interconnected modules, the estimation of output entropy for the entire design is even more complicated. Assume the following interconnection structure among *n*-bit adders (Figure 4.4). To find out the information transmitted through the whole design, we are faced with very expensive computations: the number of needed joint probabilities (and the number of terms in the summation equation (2.8)) becomes $\Theta(2^{4n})$. Therefore, this approach is quite impractical.

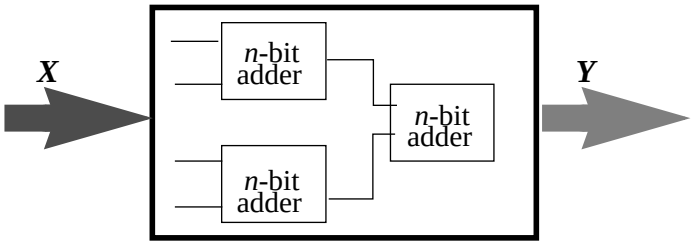


Figure 4.4 Complexity increase for interconnected modules

As a consequence, in order to analyze large designs, we target a *compositional approach* where the basic modules are already characterized in terms of transinformation and what is left to find, is only a *propagation mechanism* among them (all details are given in Section 4.4).

As we have seen, an appropriate measure for the average switching activity of each net in the circuit is its entropy value. Basically, what we need is a mapping $\xi \rightarrow \xi'$ from the actual set ξ of the nets in the target circuit (each having a possibly distinct switching activity value) to a virtual set ξ', which contains the same collection of wires, but this

time each net has the same value of switching activity. More formally, $\xi \rightarrow \xi'$ is a mapping such that the following conditions are satisfied:

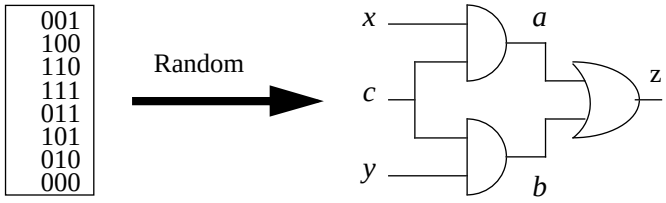
$|\xi| = |\xi'|$

$sw(x_i) = sw(x_j) \quad x_i, x_j \in \xi'$ (4.2)

Bearing in mind this, one can express the total number of transitions per step as:

$$SW(\xi) = SW(\xi') \leq n \cdot \frac{h(\xi')}{2} \quad (4.3)$$

where n stands for the presumed cardinality of ξ' and $h(\xi')$ represents the average entropy per bit of any net in ξ'. To clarify these ideas, let us consider the simple circuit in Figure 4.5.



x	c	y	a	b	z
0	0	1	0	0	0
1	0	0	0	0	0
1	1	0	1	0	1
1	1	1	1	1	1
0	1	1	0	1	1
1	0	1	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0

Figure 4.5 An example to illustrate the mapping $\xi \rightarrow \xi'$

In this example, we feed the circuit with a 3-bit random sequence and tabulate in the right side, the logic values obtained in the entire circuit by logic simulation. We have that $\xi = \{x, c, y, a, b, z\}$ with the switching profile (in number of transitions) {4, 4, 4, 2, 2, 2}, respectively. Doing a quick calculation, we get $SW(\xi) = 2.25$ transitions per step. On the other hand, $\xi' = \{x, c, y, a, b, z\}$ with the average entropy $h(\xi') = (3 * 1 + 2 * 0.811 + 0.954) / 6 = 0.9295$ characterizing each signal in the set; using equation we get an expected value $SW(\xi') = 2.73$ which is greater than $SW(\xi)$, but sufficiently close to it.

Unfortunately, in large circuits, it is expensive to compute $h(\xi')$ as the complete set of events characterizing a circuit is exponential in the number of nodes. To avoid the brute force approach, that is exhaustive logic simulation, we make some simplifying assumptions which will be detailed in Section 4.3.

4.2.2 An informational energy based approach

Based on the concepts in Section 2.2, one can find a relationship between the switching activity and the informational energy. Let $e(x)$ denote the informational energy of a single bit x. Considering x modeled as a lag-one Markov chain with conditional probabilities p_{00}, p_{01}, p_{10}, p_{11} and signal probabilities p_0 and p_1 as in Figure 4.3, we can characterize it through the conditional informational energy between two successive steps in time by: $e(x^+|x^-) = p_1 (p_{10}^2 + p_{11}^2) - p_0 (p_{00}^2 + p_{01}^2)$. Since the switching activity of line x is $\frac{2 \cdot p_{01} \cdot p_{10}}{p_{01} + p_{10}}$ and $p_1 = \frac{p_{01}}{p_{01} + p_{10}}$, $p_0 = \frac{p_{10}}{p_{01} + p_{10}}$ one can write the following relationship between conditional informational energy and switching activity: $e(x^+|x^-) = 1 - sw(x) \cdot (p_{00} + p_{11})$. Again, in the most general case, p_{00} and p_{11} can take any values and thus even if we have an exact relation between energy and switching activity ($sw(x) = (1 - e(x^+|x^-)) / (p_{00} + p_{11})$) we cannot bound the sum from the denominator. However, under temporal independence assumption [35], we have an *exact* relation since $p_{00} + p_{11} = 1$ and thus:

$$sw(x) = 2p(x)(1 - p(x)) = 1 - e(x) \tag{4.4}$$

For instance, returning to the 1-bit full-adder example in Figure 4.2, we find E_{input} = 0.125 and E_{output} = 0.875. Thus, on average, the output exposes an informational energy of 0.437 and, based on equation (4.4), a switching activity of 0.563 (compared to the exact value of 0.5). Thus, informational energy (along with entropy) seems to be a reliable candidate for estimation of energy consumption.

Once again we consider the virtual mapping $\xi \rightarrow \xi'$, where each net in ξ' is characterized by the same amount of average informational energy e (ξ'). Based on equation (4.2), the expected switching activity per step in the whole circuit SW (ξ'), can be expressed as:

$$SW(\xi) = SW(\xi') = n \cdot (1 - e(\xi')) \tag{4.5}$$

where the cardinality of ξ' is assumed to be n.

Considering the simple case in Figure 4.5, we get e (ξ') = (3 * 0.5 + 2 * 0.625 + 0.531) / 6 = 0.546 and therefore, SW (ξ') = 2.71 which matches well the actual value (2.25).

However, in real circuits, direct computation of e (ξ') is very costly; to develop a practical approach, we need further simplifications as will be shown subsequently.

4.2.3 Quantitative evaluations

In order to derive a consistent model for energy consumption at RT-level, we first have to abstract somehow the information present at gate level. Thus, a simplified model is taken as a starting point.

Let us consider some combinational block realized on n levels as a leaf-DAG[1] of 2-input NAND gates (a similar analysis can be carried out for 2-input NOR gates and the final result is the same). We assume that inverters may appear only at primary inputs/ outputs of the circuit; we do not include these inverters in the level assignment step. One can express the signal probability of any net at level j+1 as a function of the signal probability at level j by:

$$p_{j+1} = 1 - p_j^2 \qquad \forall j = 0, \ldots, n-1 \tag{4.6}$$

Similarly, the signal probability of any net at level j+2 is given by:

$$p_{j+2} = 1 - (1 - p_j^2)^2 \qquad \forall j = 0, \ldots, n-2 \tag{4.7}$$

The average entropy per net at level j is given by:

$$h_j = -p_j \cdot \log p_j - (1 - p_j) \cdot \log(1 - p_j) \tag{4.8}$$

Using the corresponding average entropy per net at level j+2, the parametrized relationship between h_j and h_{j+2} can be approximated by $h_{j+2} \approx \frac{h_j}{2}$ when j is sufficiently large (values greater than 6). Hence we get expressions for entropy per bit at even/odd levels of the circuit: $h_{2j} \approx \frac{h_0}{2^j}$ and $h_{2j+1} \approx \frac{h_1}{2^j}$, where h_0, h_1 are entropies per bit at the primary inputs and first level, respectively. To get a closed form expression, we may further assume that h_1 may be estimated in terms of h_0 as $h_1 \approx \frac{h_0}{\sqrt{2}}$ (in fact, the exact entropy decrease for a pseudorandom excited NAND gate is 0.811, but for uniformity, we

1. In a leaf-DAG, only the leaf nodes have multiple fanouts.

chose this way). Thus, for a 2-input NAND gate leaf-DAG, the entropy per bit at level *j* may be approximated as:

$$h_j \approx \frac{h_0}{2^{j/2}} \tag{4.9}$$

This may be further generalized for the case of *f*-input NAND gate leaf-DAGs, observing that increasing the fanin from 2 to *f*, produces an decrease in the number of levels by log *(f)*. Hence, for a fanin of *f*, equation (4.9) becomes:

$$h_j \approx \frac{h_0}{2^{(j \cdot \log f)/2}} = \frac{h_0}{f^{\,j/2}} \tag{4.10}$$

We call $\frac{1}{\sqrt{f}}$ *information scaling factor;* it characterizes each logic component (gate, module or circuit). We will see how this relation is affected by the circuit structure and functionality in general. In any case, this provides a starting point for estimating the total entropy at each level in the circuit. In general, the total entropy over all levels *N* in the circuit would thus be:

$$H_{total} = \sum_{j=0}^{N} H_j = \sum_{j=0}^{N} n_j \cdot h_j \tag{4.11}$$

where H_j is the total entropy at level *j* and n_j is the number of nodes on level *j*.

All these considerations can be easily extended for the case of informational energy. Considering the same assumptions as in previous section and using equation (4.6), the informational energy per net at level *j* may be expressed as:

$$e_j = p_j^2 + (1 - p_j)^2 \tag{4.12}$$

Applying equation (4.6) for level j+2 and substituting in equation (4.12), we get the following parameterized dependency between the informational energies at levels j + 2 and j:

$$e_{j+2} = (1-(1-p_j^2)^2)^2 + (1-p_j^2)^4 \qquad e_j = p_j^2 + (1-p_j)^2 \tag{4.13}$$

Using a similar approach as in the case of entropy, we get the following expression for the average informational energy per bit at level j in a circuit with fanin f:

$$e_j = 1 - \frac{1-e_0}{f^{\,j/2}} \tag{4.14}$$

From here, an estimate can be drawn for the total energy at level j, and thus for the total energy over all the levels of the circuit:

$$E_{total} = \sum_{j=0}^{N} E_j = \sum_{j=0}^{N} n_j \cdot e_j \tag{4.15}$$

where E_j is the total energy at level j and again n_j is the number of nodes on level j.

4.3 Information modeling

4.3.1 Theoretical results

As we have seen, an estimate of the average switching activity for a module can be obtained from the total entropy (informational energy) over all levels of the circuit. An exact technique would be too expensive to use in practice; on the other hand, since we are dealing with RT-level designs, the internal structure may be unknown. So, in order to make things manageable, we will use the following simplifying assumptions:

A_1. *Uniform Network Structure:* Nodes are uniformly distributed over the levels of the circuit.

In other words, we assume the same number of nodes on each level of the circuit. Also, all the gates on each level are assumed to get their inputs from the previous level. This will significantly simplify our task in obtaining closed-form formulae for average switching activity per module (see Section 4.4 for the effect of other common network structures when assumption A_1 is relaxed).

As we have seen, for particular structures (that is leaf-DAGs containing 2-input NAND/NOR gates), there is a simple relationship between the values of entropy (informational energy) on adjacent levels in the circuit. Unfortunately, in practice these structures are too restrictive to be considered alone: random logic circuits exhibit a large fanout, not only at the primary inputs, but also at internal nodes. This is one reason to reconsider the above relations and make them applicable in practice. In addition, logic circuits contain a mixture of gates: while NAND (AND), NOR (OR) are *entropy decreasing*, XORs and inverters are *entropy preserving* gates. More precisely, the output entropy of XORs is 1 when they are randomly excited and therefore their information scaling factor is 1. Their behavior is still described by equation (4.10) for $f = 1$ (similar considerations apply to informational energy). In general, any generic "gate" having almost equal-sized ON and OFF sets, exposes the same *almost entropy preserving* characteristic.

Example 4.2

a b c	*f g*
0 0 0	0 0
0 0 1	1 0
0 1 0	1 1
0 1 1	0 0
1 0 0	1 1
1 0 1	0 0
1 1 0	0 0
1 1 1	1 0

As we can easily see, the boolean function *f* has equal ON- and OFF-sets and thus, implements an information preserving gate. If exhaustively excited, the output entropy of *f* is:

H_f = - 0.5 log(0.5) - 0.5 log(0.5) = 1 and thus the information scaling factor is 1.

On the other hand, *g* has unbalanced ON- and OFF-sets (only 2 out of 8 minterms are in the ON-set). The corresponding output entropy is:

H_g = - 0.25 log(0.25) - 0.75 log(0.75) = 0.8113 which corresponds to its information scaling factor.

In short, we may say that *both structural* and *functional* aspects are important. At RT-level both of them have to be abstracted and used in an implicit manner to compensate the lack of explicit information which characterize high-level representations. To overcome this difficulty, we will use another simplifying assumption:

A_2. *Uniform Information Variation:* The entropy and informational energy per bit at level *j* are estimated in terms of f_{eff} as:

$$h_j = \frac{h_0}{f_{eff}^{j/2}} \quad \text{and} \quad e_j = 1 - \frac{1 - e_0}{f_{eff}^{j/2}}, \quad j = 0, 1, ..., N. \tag{4.16}$$

Differently stated, we assume that each “generic gate” from a given circuit is characterized by an *effective information scaling factor* whose value depends on both structure and functionality of gates. In fact, the above formula is similar with equation (4.10) and equation (4.14) derived for a particular circuit (leaf-DAG).

Under assumptions A_1 and A_2, we may state the following:

Proposition 4.1 The average entropy (informational energy) per bit in an *N*-level circuit, may be estimated as:

$$h_{avg} = h_{in} \cdot \frac{1 - \left(\frac{h_{out}}{h_{in}}\right)^{\frac{N+1}{N}}}{(N+1) \cdot \left(1 - \left(\frac{h_{out}}{h_{in}}\right)^{\frac{1}{N}}\right)}$$

$$e_{avg} = 1 - (1 - e_{in}) \cdot \frac{1 - \left(\frac{1 - e_{out}}{1 - e_{in}}\right)^{\frac{N+1}{N}}}{(N+1) \cdot \left(1 - \left(\frac{1 - e_{out}}{1 - e_{in}}\right)^{\frac{1}{N}}\right)} \qquad (4.17)$$

where h_{in} (e_{in}), h_{out} (e_{out}) are the average input and output entropies (energies) per bit.■

Proposition 4.1 gives us an estimate of the average entropy/informational energy in a circuit with *N* levels. The factor f_{eff} is "hidden" in the relationship between *N*, h_{in} (e_{in}) and h_{out} (e_{out}) since the outputs are considered to be on level *N*:

$$h_{out} = h_N = \frac{h_0}{f_{eff}^{N/2}} = \frac{h_{in}}{f_{eff}^{N/2}} \qquad e_{out} = e_N = 1 - \frac{1 - e_0}{f_{eff}^{N/2}} = 1 - \frac{1 - e_{in}}{f_{eff}^{N/2}} \qquad (4.18)$$

The greater the number of levels is, the smaller the value for f_{eff} will be, which somehow suggests that the loss of information per bit from one level to another decreases with the number of levels. However, the usefulness of these formulas is limited, since in general at RT-level we know very little about the internal structure of the circuit. To

compensate this lack of information, we add a new assumption valid for circuits with large logical depth:

A_3. *Asymptotic Network Depth:* The number of levels N is large enough to be considered infinity.

Using this, we get the following:

Corollary 4.1 For sufficiently large N, the average entropy and informational energy per bit in the circuit are given by:

$$h_{avg} = \frac{h_{in} - h_{out}}{\ln \frac{h_{in}}{h_{out}}} \qquad e_{avg} = 1 - \frac{e_{in} - e_{out}}{\ln \frac{1 - e_{out}}{1 - e_{in}}} \tag{4.19}$$

What we have obtained so far are simple formulae for estimating the average entropy (informational energy) per bit, and from these, the average switching activity over all the nets in the module. The main difficulty in practice is to estimate the actual output entropy h_{out} (or informational energy e_{out}), since the information usually available at this level of abstraction is not detailed.

4.3.2 The influence of structure and functionality

All logic gates belonging to a given module can be characterized by an effective factor f_{eff} which captures information about the circuit structure and functionality. How can we model a general circuit for entropy/energy based evaluations? One can consider equation (4.17) and equation (4.19), where the information scaling factor reflects not only the structure, but also the fraction of information preserving gates.

Example 4.3 Let us consider for instance, circuit C17 given below:

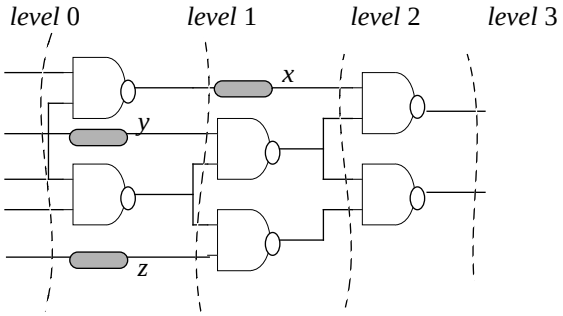


Figure 4.6 An example of levelization - Circuit C17

To levelize it properly (every wire that connects the output of a gate at level i to the input of a gate at level $i+2$ must go through some buffer gate at level $i+1$), we added three "dummy" components x, y, z. Logically, x, y, z function as buffers but informationally, they are entropy preserving elements. Considering the nodes uniformly distributed throughout the circuit (according to assumption A_1), the average number of nets per level is (6+5+4+2)/4 = 4.25. Applying random vectors at the circuit inputs, the exact value of the entropy per bit at the output is obtained as h_{out} = 0.44. The effective scaling factor can be calculated as a weighted sum over all the gates in the circuit; thus the corresponding f_{eff} is: $\left(\frac{9}{3 \cdot 1 + 6 \cdot \frac{1}{\sqrt{2}}} \right)^2 = 1.55$ (there are 3 entropy preserving and 6 entropy decreasing gates). From equation (4.19) we get an estimate for the output bit entropy (j = 3) as h_{out} = 0.51 which is reasonably close to the exact value. Based on the input and output entropy, we may get an estimate for the average entropy per bit and thus for the switching activity. The average switching activity for a generic net in the circuit is $sw_{avg(sim)}$ = 0.437 (from simulation) and based on equation (4.17), we get $sw_{avg(est)}$ =

0.382 which is very good compared to simulation. A similar analysis can be performed for the informational energy.

4.4 Practical considerations

4.4.1 Using structural information

These considerations are equally applicable to data-path operators with known internal structure as well as to control circuits represented either at gate or RT-level.

If some structural information is available (such as the number of internal nodes, the number of logic levels), the average entropy (informational energy) may be evaluated using the actual values of f_{eff}, N and the distribution of nodes on each level. In all cases, the output entropy (informational energy) is the same, computed as in equation (4.16). The average entropy (or informational energy) for the whole module depends on the actual distribution of nodes in the circuit. In practice, some common distributions are:

a) Uniform distribution (this case was treated in detail in Section 4.3).

b) Linear distribution (e.g. circuit C17 in Figure 4.6):

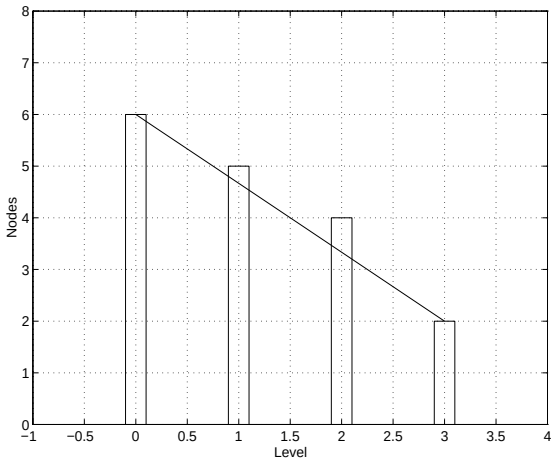


Figure 4.7 An example of linear distribution of nodes

Using a similar approach as the one in Section 4.3, we found the following result (valid for sufficiently large *N*) for a generic *n* input, *m* output module:

$$h_{avg} = \frac{2 \cdot n \cdot h_{in}}{(n+m) \cdot \ln\left(\frac{h_{in}}{h_{out}}\right)} \cdot \left(1 - \frac{m}{n} \cdot \frac{h_{out}}{h_{in}} - \frac{\left(1 - \frac{m}{n}\right) \cdot \left(1 - \frac{h_{out}}{h_{in}}\right)}{\ln\left(\frac{h_{in}}{h_{out}}\right)} \right) \tag{4.20}$$

c) Exponential distribution (e.g. a balanced tree circuit with 8 inputs):

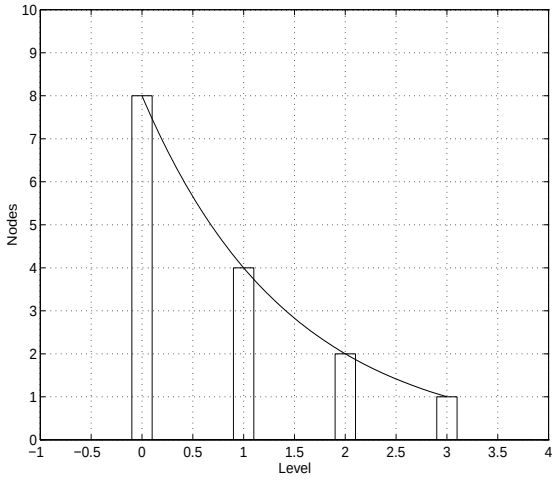


Figure 4.8 An example of exponential distribution of nodes

In this case, we have:

$$h_{avg} = \frac{h_{in} \cdot \ln\left(\frac{n}{m}\right)}{1 - \frac{m}{n}} \cdot \frac{1 - \frac{m \cdot h_{out}}{n \cdot h_{in}}}{\ln\left(\frac{n \cdot h_{in}}{m \cdot h_{out}}\right)} \tag{4.21}$$

Note: The main advantage of equation (4.20) and equation (4.21) is that they allow an estimate of average entropy (and therefore for average switching activity) of modules or gate-level circuits, *without* resorting to logic simulation or probabilistic techniques.

Similar derivations apply for informational energy. We can see that when $n = m$, we get the same results as in Section 4.3 (see equation (4.19)).

4.4.2 Using functional information

Common data-path operators allow a quick estimation of h_{out} based on the "compositional technique" introduced in [57]. There, the Information Transmission Coefficient (*ITC*) is defined as the fraction of information that is transmitted through a function; it may be computed by taking the ratio of the entropy on the outputs of a function and the entropy on the inputs of that function. For convenience, we call *ITC*s "Entropy Transmission Coefficients" (*HTC*s) either for input or component values. In Table 4.1 we give the *HTC* values for some common 8-bit data-path operators as they appear in [57].

Table 4.1. *HTC* Values for common 8-bit data-path operators

Operator	HTC	Operator	HTC
Addition	0.500	Negation	1.000
Subtraction	0.500	And, Or	0.406
Multiplication	0.461	<, >	0.063
Divide by 2	0.875	Multiplexer	0.471

Using the following relationship between the *HTCs* on the output signals and the *HTC* values on the input signals for a particular component, we may estimate the *HTC* values throughout the circuit as:

$$HTC_{out} = HTC_{comp} \cdot \sum_{i=1}^{n} \frac{w_i}{W} \cdot HTC_i \tag{4.22}$$

where HTC_{comp} is the *HTC* for the component of interest, HTC_i is the *HTC* value for input *i*, *n* is the number of input signal paths for the component, w_i is the data-path width for input *i*, and $W = \sum_{i=1}^{n} w_i$.

In particular, the output entropy h_{out} may be estimated relying solely on the RT-level description of the circuit. The main advantage of such an approach, is that *it needs only a high-level view of the design* in order to derive useful information.

The *HTC* value on the inputs is in fact the entropy normalized to the data-path width. To make the compositional technique useful for our purpose, we need to consider both the input and output entropies normalized to the data-path width. For an *n*-input data-path operator, we have:

$$h_{out} = HTC_{comp} \cdot (n \cdot h_{in}) \tag{4.23}$$

where h_{in} is the average input bit entropy.

Example 4.4 Let's apply the compositional technique to the following circuit composed by two 8-bit adders (A_1 and A_2), and one multiplier (M_1).

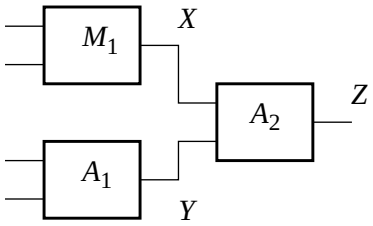


Figure 4.9 The compositional technique

Since M_1 and A_1 are fed by the primary inputs, we have that $HTC_X = HTC_{M1} = 0.461$ and $HTC_Y = HTC_{A1} = 0.500$. For all the primary inputs of the modules M_1 and A_1 we have h_{in}

= 1. According to equation (4.23) we have that $h_X = 0.461 \cdot 2 \cdot 1 = 0.92$ and $h_Y = 0.500 \cdot 2 \cdot 1$. Assuming that all paths have the same width, then the weighted average of the inputs to A_2 is $\frac{h_X + h_Y}{2}$, that is $\frac{1 + 0.92}{2} = 0.96$ and therefore $h_Z = 0.500 \cdot 2 \cdot 0.96 = 0.96$.

However, in general, the *HTCs* associated with a component are not constant and depend on the input bit entropy (or input *HTC*s) as we shall see later in this section.

A similar technique can be introduced to compute the output informational energy as follows.

Definition 4.1 The fraction of informational energy transmitted through a function called "Energy Transmission Coefficient" (*ETC*) is defined as the ratio of the output and input informational energy.

In Table 4.2 we give the values of the *ETC* coefficients for the same data-path operators considered in Table 4.1.

Table 4.2. *ETC* Values for common data-path operators

Operator	ETC	Operator	ETC
Addition	0.500	Negation	1.000
Subtraction	0.500	And, Or	0.625
Multiplication	0.516	<, >	0.063
Divide by 2	1.125	Multiplexer	0.471

Hence, a compositional technique can also be developed here. Similar to equation (4.22), we may evaluate ETC_{out} as a function of *ETC* for the component of interest and the *ETC* values for all inputs:

$$ETC_{out} = ETC_{comp} \cdot \sum_{i=1}^{n} \frac{w_i}{W} \cdot ETC_i \qquad (4.24)$$

Thus, for modules with n inputs, the output informational energy is related to the input energy by the following relation:

$$e_{out} = ETC_{comp} \cdot (n \cdot e_{in}) \tag{4.25}$$

where e_{in} is the input informational energy per bit.

This type of considerations can be used for any type of data-path operator which has the *scalability property* that is, all sizes of the operator behave similarly up to a multiplicative factor (which is the data-path width). Common arithmetic operators exhibit the scalability property, but unfortunately, there are many other circuits (e.g. control circuits) which cannot be treated in this manner. In those cases, equation (4.19) has to be used in conjunction with some information about the circuit structure in order to get reliable estimates for average switching activity.

4.4.3 HTC and ETC variations with the input statistics

As presented in [57], Thearling and Abraham's compositional techniques is only an approximation because it does not consider any dependency which may arise in practical examples. In reality, every module may be embedded in a larger design and therefore its inputs are no longer independent due to the structural dependencies (namely, the reconvergent fan-out). As a consequence, the values given in Table 4.1 or Table 4.2 (which correspond to the case of pseudorandom inputs) cannot be used without error as we proceed from circuit inputs to circuit outputs; in order to be accurate we need a more detailed analysis as will be described in the following.

Without loss of generality, we restrict ourselves to the case of 8- and 16-bit adders and multipliers and for each of them, we consider two scenarios:

- Each module is fed by biased input generators, that is input entropy (informational energy) per bit varies between 0 and 1 (respectively 0.5 and 1); each such module is separately analyzed.

- Modules are included in a large design with reconvergent fanout branches, so that inputs of the modules cannot be considered independent. The structure of such an example design is the following:

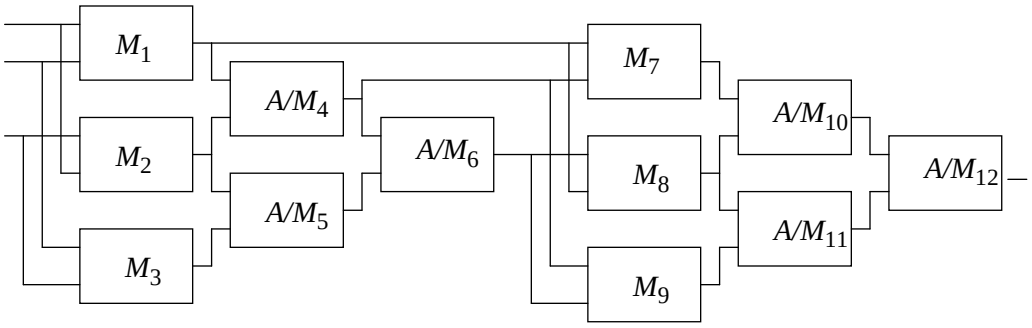


Figure 4.10 The configuration used to analyze the variation of *HTC*/*ETC* values

Initially, for the analysis of multipliers, all modules were considered multipliers; in the analysis of adders, modules 4-6, 10-12 were adders (modules 1-3, 7-9 were still kept multipliers to increase the level of correlation in the design).

In both scenarios, the average input and output entropies (energies) per bit were monitored and using them, the *HTC* and *ETC* values were extracted. Values obtained in both scenarios for each module are plotted on the same graph, as shown in Figure 4.11 through Figure 4.14:

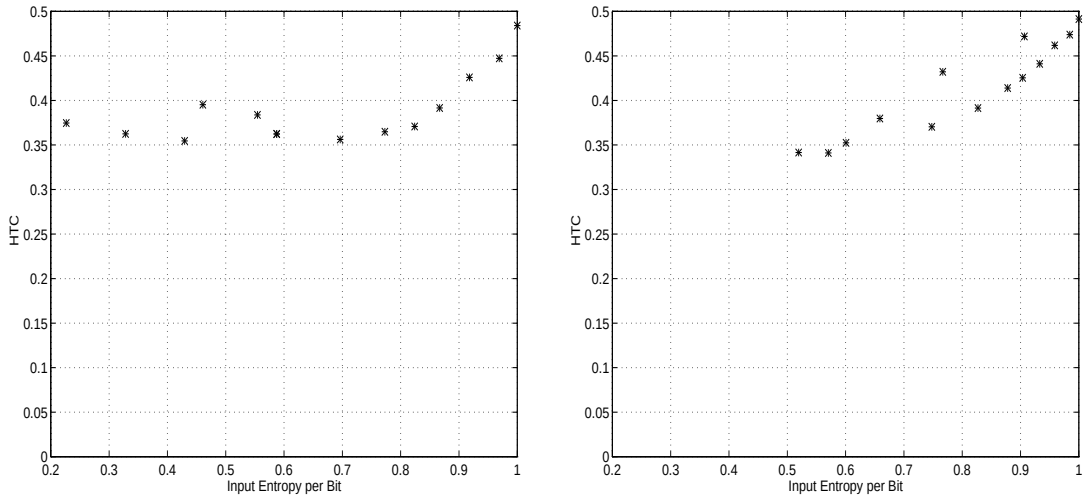


Figure 4.11 *HTC* values for 8- and 16-bit Multipliers

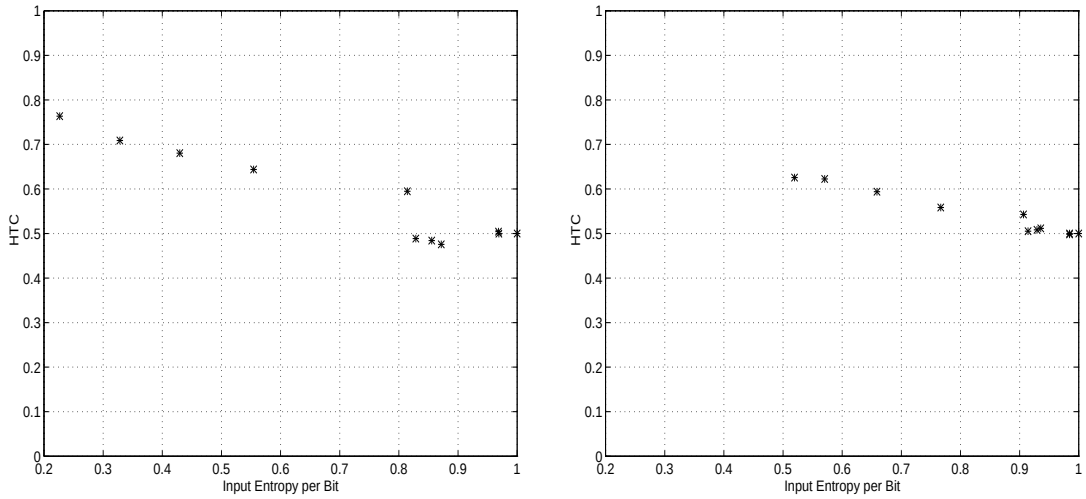


Figure 4.12 *HTC* values for 8- and 16-bit Adders

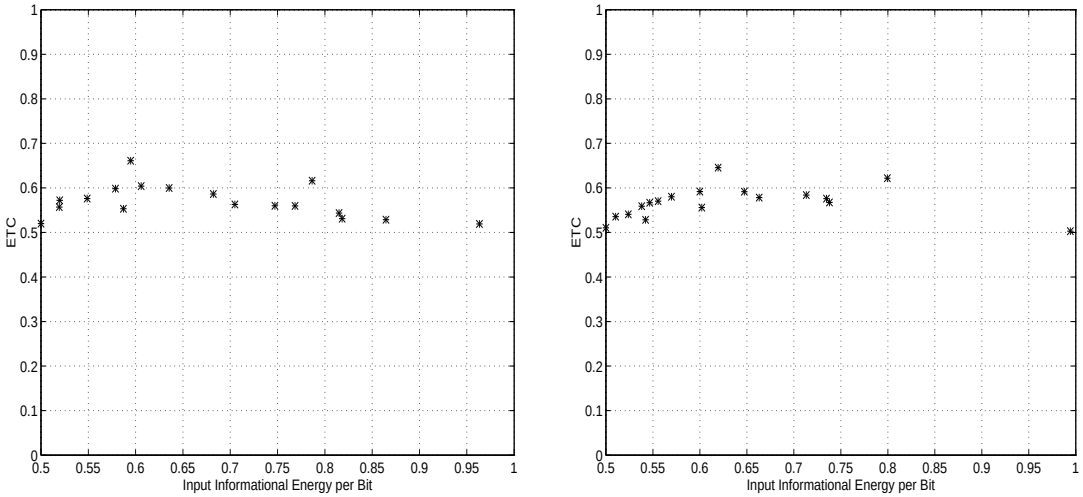


Figure 4.13 *ETC* values for 8- and 16-bit Multipliers

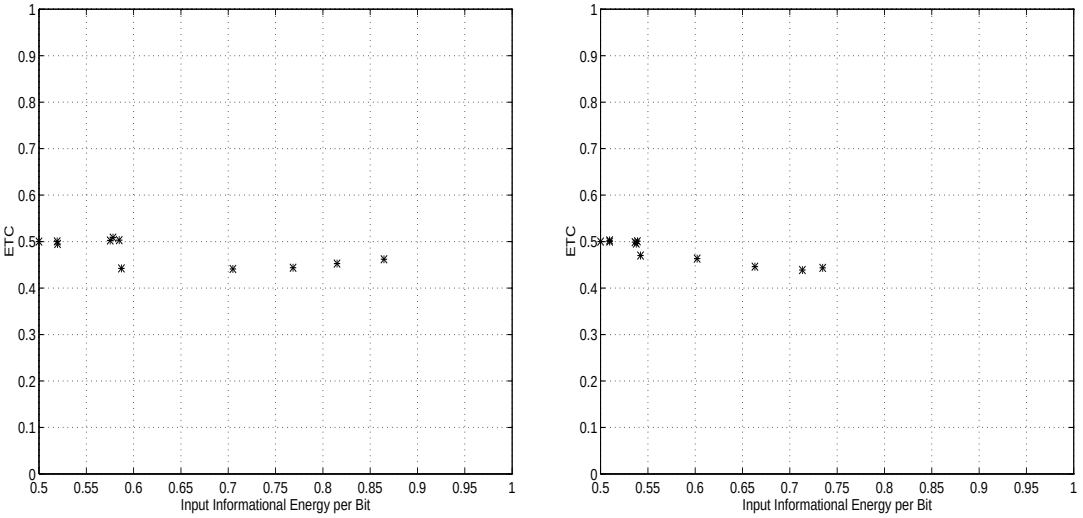


Figure 4.14 *ETC* values for 8- and 16-bit Adders

As we can see, the behavior of the operators is almost independent of the data-path width, so the same values for *HTC* (*ETC*) can be safely used in both cases. We point out that *ETC* values are less dependent on the input statistics than *HTC*s and presumably, the

technique based on informational energy will be less prone to error when the inputs are far from the pseudorandom case. The dependence of *HTC*s and *ETC*s on the input statistics (that is, input entropy and/or informational energy) can be described empirically by the following simple relations:

$$HTC^{add} \approx HTC_0^{add} \cdot (2 - h_{in})$$

$$HTC^{mul} \approx HTC_0^{mul} \cdot 2^{h_{in}-1} \tag{4.26}$$

$$ETC^{add} \approx ETC_0^{add}$$

$$ETC^{mul} \approx ETC_0^{mul}$$

where the 0-subscripted values correspond to the pseudorandom case (reported in Table 4.1 and Table 4.2). These equations can be easily used to adjust the *HTC/ETC* coefficients in order to analyze large designs more accurately. Differently stated, using equation (4.26) we do not loose information from level to level because we account for structural dependencies.

If the data-path is within a loop, then we may resort to a *loop-unrolling* technique in order to evaluate the output entropy (informational energy). For this purpose, equation (4.26) can be used to compute the actual values of *HTC*/*ETC* coefficients across successive iterations. For instance, assume we have to compute the dot-product of two random vectors *X* and *Y* using the following data-flow graph:

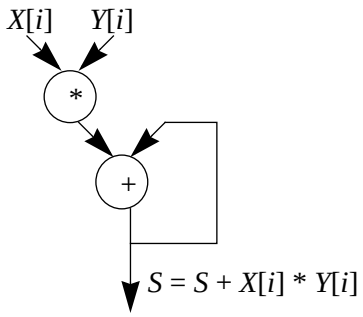


Figure 4.15 Dot-product computation

Monitoring the output entropy (informational energy) values, we got the following behavior as a function of the number of steps performed in the unrolling process (Figure 4.16).

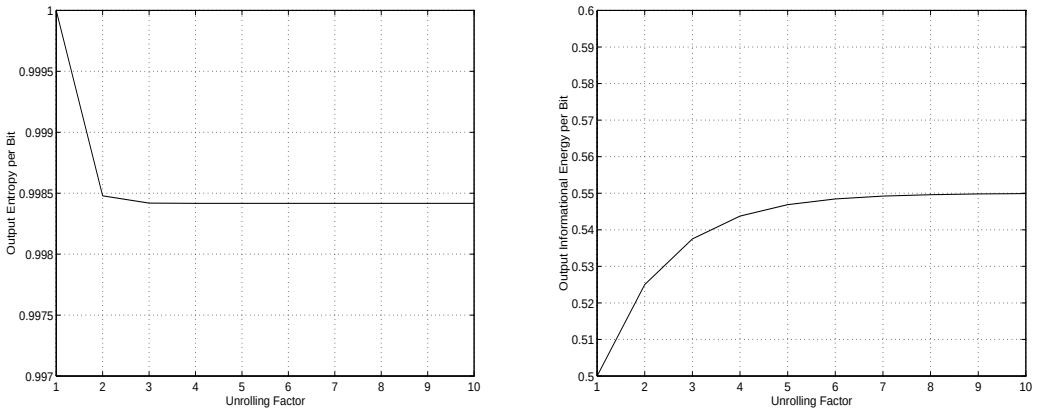


Figure 4.16 h_{out} and e_{out} behavior during loop-unrolling

This is a typical behavior in practice, that is, the two information measures converge rapidly to the steady-state values and therefore we may exploit this property to use it in conjunction with the compositional technique described before.

In any case, this framework is also open to simulation (the *zero-knowledge scenario*); this may provide accurate values for output entropy (informational energy) values, but

with a much higher computational cost. In practice, we thus have the following options to analyze a logic- or RT-level design:

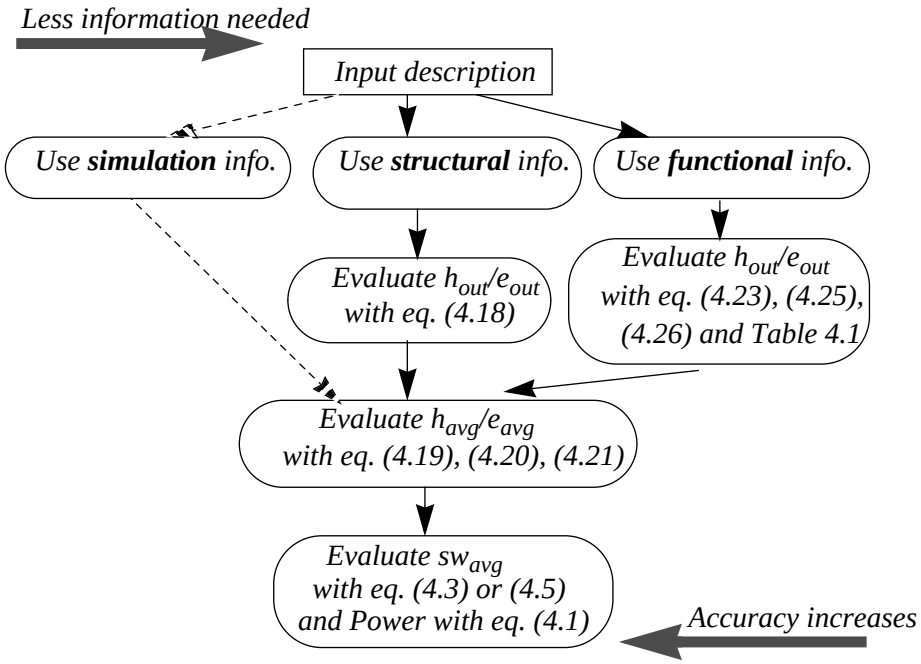


Figure 4.17 Flowchart of the power estimation procedure

In general, using structural information can provide more accurate results either based on entropy or informational energy measures. On the other hand, evaluations based on functional information require less information about the circuit and therefore may be more appealing in practice as they provide an estimate of power consumption earlier in the design cycle.

To illustrate this aspect, we provide here our results obtained for some common data-path operators (4-bit adders and multipliers) with different structures: ripple-carry and carry-lookahead adders, and array and Wallace-tree multipliers. The exact values for the average switching activity per module are shown in Table 4.3; they were obtained by exhaustive logic simulation using the SIS logic simulator.

Table 4.3. Exact values for average switching activity

Ripple-Carry Adder	Carry-Lookahead Adder	Array Multiplier	Wallace-tree Multiplier
0.4187	0.3591	0.2846	0.2860

Results of Table 4.3 show that for modules with the same functionality, the structure plays an important role; specifically, the average switching activity may be different even when the input statistics are the same. An important observation is that looking only at the input/output relationship (even through exhaustive simulation), one is unable to distinguish between two different implementations of the same functional module, but having different average switching activities.

To asses the accuracy of the different approaches in Figure 4.17, we considered first the two 4-bit adders (ripple-carry and carry-lookahead) having the following node distributions:

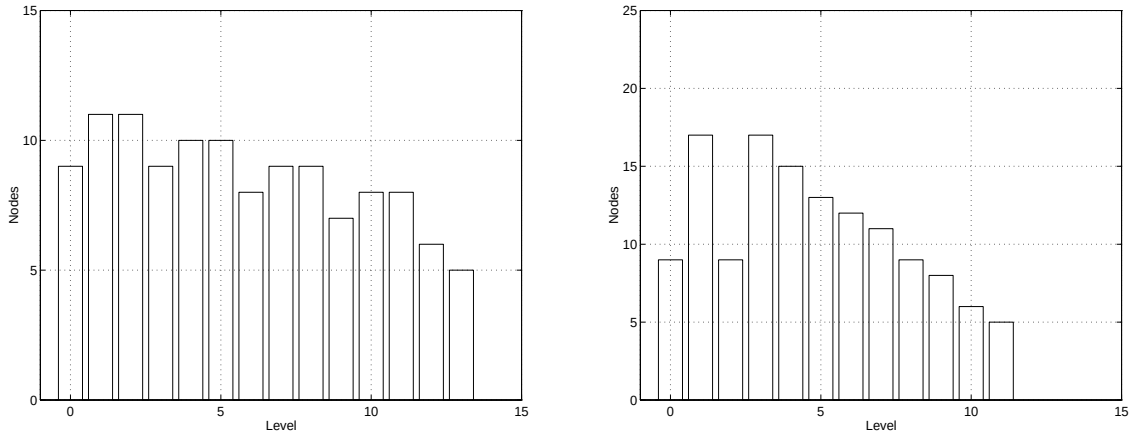


Figure 4.18 4-bit ripple-carry (left) and carry-lookahead adder (right)

We estimated first the average switching activity per module using the compositional technique presented previously under the uniform distribution assumption (the default option). After that, assuming that more structural information is available (i.e. node

distribution was taken to be linear), we re-calculated the switching activity using equation (4.18). As shown in Table 4.4, the quality of results is improved in the latter case. (We report the absolute error defined as $abs\_err = |sw_{avg(sim)} - sw_{avg(est)}|$ where $sw_{avg(sim)}$ was taken from Table 4.3).

Table 4.4. Analysis of structurally different adders

	abs_err for Ripple-carry Adder		*abs_err* for Carry-lookahead Adder	
Measure	Functional info	Structural info	Functional info	Structural info
h	0.0494	0.0351	0.1073	0.0588
e	0.0493	0.0350	0.1072	0.0587

The best results are obtained when structural information is available. This is expected as structural information provides more detailed description of the circuit in question.

For the sake of completeness, we also analyzed two structurally different 4-bit multipliers (array and Wallace tree). We give in Figure 4.19 the node distributions for these circuits and in Table 4.5 the results of a complete analysis based on functional or structural information. The distribution of nodes per level was considered exponential for the Wallace-tree multiplier; the actual distribution in the case of array multiplier was approximated by a piecewise linear fit. Again, the best results were obtained when using structural information to calculate the output entropy.

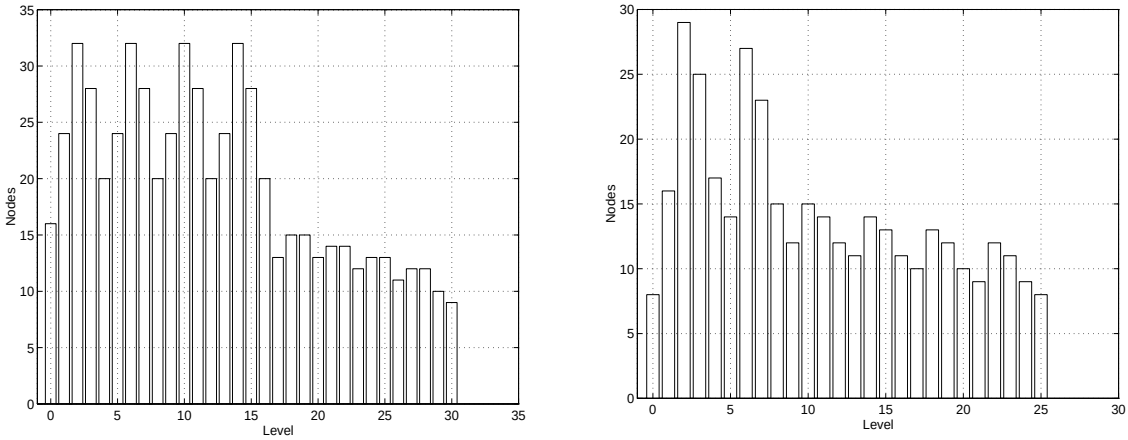


Figure 4.19 4-bit array (left) and Wallace-tree multipliers (right)

Table 4.5. Analysis of structurally different multipliers

	abs_err for Array Multiplier		*abs_err* for Wallace-Tree Multiplier	
Measure	Functional info	Structural info	Functional info	Structural info
h	0.0517	0.0429	0.0873	0.0102
e	0.0444	0.0427	0.0871	0.0101

To conclude this section, the structural approach is more appropriate to be used when a gate-level description is available (and therefore detailed information can be extracted) whereas the functional approach (using the compositional technique) suits better for RT/ behavioral level descriptions.

4.5 Experimental results

The main advantage of the proposed power consumption model is that it *does not need any simulation*. In the following we report the accuracy of this model.

Two experiments were performed: one involving individual modules (ISCAS'85 benchmarks and common data-path components) and the other involving a collection of data-path modules specified by a data flow graph.

4.5.1 Validation of the structural approach (gate-level descriptions)

The experimental setup consisted of a pseudorandom input generator feeding the modules under consideration. The values of the entropy and informational energy for the circuit inputs were extracted from the input sequence, while the corresponding values at the circuit outputs and the average values of entropy or informational energy were estimated as in Section 4.3 (using structural information). These average values were then used to estimate the average switching activity per node; the latter, when weighted by an average module capacitance, is a good indicator of power/energy consumption.

We report in Table 4.6 our results on ISCAS'85 benchmark circuits and common data-path operators. $Power_{sim}$ and sw_{sim} are the exact values of power and average switching activity obtained through logic simulation under SIS. C_{module} stands for the module capacitance as in equation (4.1); it is taken as the product of number of nets and the average load seen by each gate in the circuit. We also report for comparison under the *Power* column the value of power obtained using the approximation in equation (4.1). The error introduced by this approximation is on average 5.55%[1]. In the next four columns we report our results for average switching activity and power calculated as in equation (4.1) for both entropy- and informational energy-based approaches.

1. The percentage error was calculated as $\left|\frac{value_{sim} - value_{est}}{value_{sim}}\right| \cdot 100$.

Table 4.6. Total power and average switching activity

	Simulated values with SIS				Estimated values as in Section 4.3			
Circuit	$Power_{sim}$	C_{module}	sw_{sim}	*Power*	sw_{avg} from *h*	$Power_{est}$ from *h*	sw_{avg} from *e*	$Power_{est}$ from *e*
add8	1019.39	9.46	0.4199	993.06	0.4410	1042.63	0.4410	1042.51
add16	2082.70	18.91	0.4282	2024.31	0.4546	2149.40	0.4546	2148.58
add32	3980.91	37.82	0.4073	3851.02	0.4923	4654.89	0.4923	4654.74
mul4	2785.53	25.04	0.4170	2610.42	0.4594	2875.60	0.4471	2799.03
mul8	10949.51	100.16	0.4081	10218.82	0.4545	11381.09	0.4410	11043.80
mul16	44076.52	400.64	0.4094	41005.50	0.4515	45222.57	0.4373	43795.33
mul32	166235.89	1602.56	0.3904	156409.86	0.4498	180219.10	0.4351	174333.74
C1355	5163.54	53.59	0.3779	5062.92	0.4261	5707.99	0.4058	5436.64
C1908	5957.64	66.09	0.3417	5645.74	0.4280	7071.99	0.4090	6756.85
C3540	13966.75	175.03	0.2809	12291.48	0.3586	15691.77	0.3281	14354.56
C432	2985.67	29.22	0.3862	2821.19	0.4468	3263.71	0.4331	3163.47
C499	6085.72	61.69	0.3794	5851.30	0.4301	6633.53	0.4120	6354.57
C6288	41860.77	430.24	0.3582	38527.99	0.4354	46832.16	0.4173	44880.26
C880	5200.12	54.78	0.3540	4848.03	0.4465	6115.32	0.4318	5913.98

As we can easily see, the *average percentage error* (over all the circuits) is 15.81% (12.03%) for entropy (informational energy)-based evaluations of average switching activity, whereas for total power estimation is 9.27% (5.85%) for entropy (informational energy)-based approaches.

4.5.2 Validation of the functional approach (data-flow graph descriptions)

In Figure 4.20 we consider a complete data-path represented by the data-flow graph of the differential equation solver given in [49]; all the primary inputs were considered as having 8 bits and the output entropy of the entire module was estimated with the compositional technique based on *HTC*s or *ETC*s. The *HTC*/*ETC* values for the multipliers and adders were taken as in equation (4.26). Using the entropy based approach, the average switching activity was estimated as 0.1805, whereas when using

the informational energy, the average switching activity was 0.1683. Comparing these results against the exact value of 0.1734 obtained by behavioral simulation, the overall error in estimating the average switching activities using entropy and informational energy is 0.0071 and 0.0051, respectively.

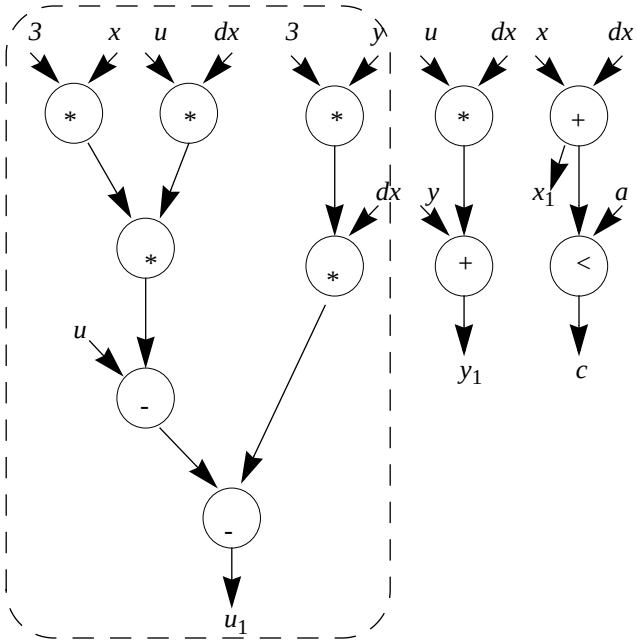


Figure 4.20 A data-path example

Our technique can also be applied to analyze different implementations of the same design in order to trade-off power for area, speed or testability. Suppose we select from the data-path in Figure 4.20 only the part which computes u_1. In Figure 4.21 we give two possible implementations of the selected part of the data-path. One is the same as above and the other is obtained using common algebraic techniques such as factorization and common subexpression elimination. All the primary inputs were assumed to be random (except inputs '3' and '*dx*' which are constants). Each adder or multiplier is labelled with its average switching activity value *SW*. In the first case, applying the compositional

technique based on entropy, we obtain an average switching activity of 0.186, whilst using informational energy this value is 0.197. For the second implementation, the corresponding values are 0.242 from entropy and 0.282 from informational energy which show an average increase in switching activity of 30%. However, considering the module capacitance of adders and multipliers (as given in Table 4.6), we actually get a total switched capacitance of 331.15 for the first design and 268.88 for the second one (using entropy-based estimations). This means a decrease of 19%, and thus, the second design seems to be a better choice as far as power consumption is concerned.

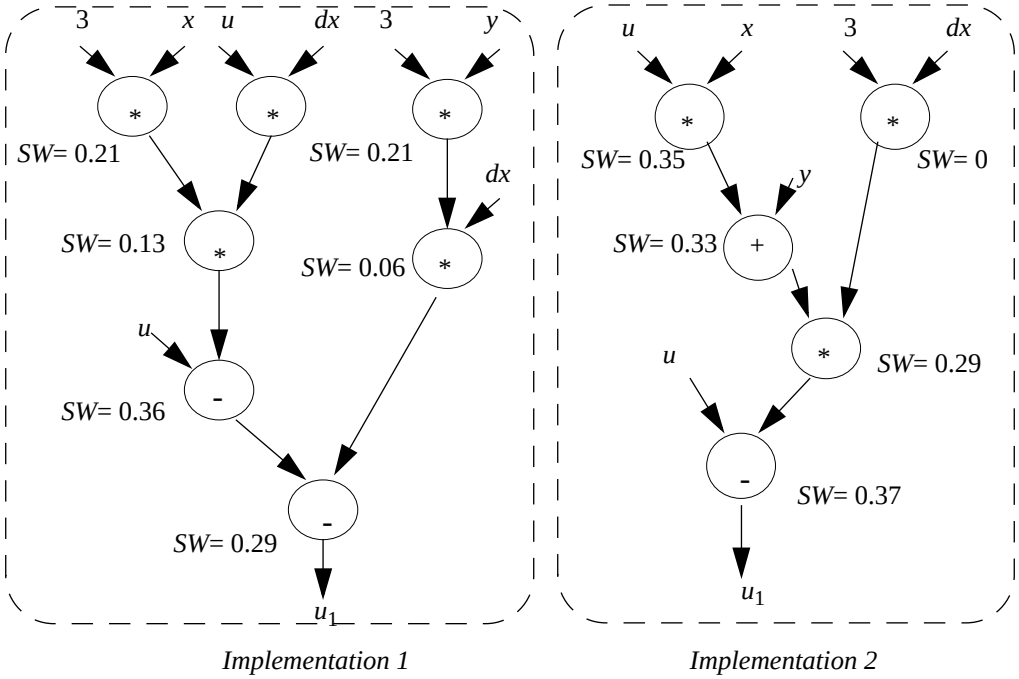


Figure 4.21 Comparison between two possible implementations

4.6 Summary

In this chapter, we have proposed two new measures to estimate the power consumption in logic- or RT-level digital circuits. Their foundation relies on statistical modeling of the circuit behavior and seems to successfully overcome the lack of information which characterizes higher levels of abstraction. The accuracy of the model is investigated using common benchmarks for pseudorandom input sequences. As shown, the average switching activity may be predicted without simulation using either entropy or informational energy averages; the error in prediction is generally small enough to be satisfactory in practice.

Chapter 5 Probabilistic Analysis of FSMs

5.1 Introduction

In the last decade, probabilistic approaches have received a lot of attention as a viable alternative to deterministic techniques for analyzing complex digital systems. Logic synthesis [48], verification [17], testing [52] and more recently, low-power design [18] have benefited from using probabilistic techniques. In particular, the behavior of FSMs has been investigated using concepts from the Markov chain (MC) theory.

Studying the behavior of the MC provides us with different variables of interest for the original FSM. In this direction, [19][60] are excellent references where steady-state and transition probabilities (as variables of interest) are estimated for large FSMs. Both techniques are analytical in nature but, in order to manage complexity, make some simplifying assumptions, temporal independence of the primary inputs being the most notable one. This latter assumption limits the applicability of the results. As we will show in this chapter, temporal correlations longer than one time step can significantly affect the overall behavior of the FSM and therefore result in very different values for the actual transition probabilities compared to those predicted in [19][60]. More interestingly, it will be shown that if one ignores the effect of finite-order statistics at the primary inputs of the FSM, it is possible to wrongly predict non-zero steady-state probabilities for some transient states (which normally occur in the beginning of operation, but disappear once the machine reaches its steady-state regime). This kind of erroneous prediction for the state occupation probabilities is especially harmful in timing verification, when one tries to find the probability of occurrence of an event related to a state or transition in the circuit. Knowledge of correct state occupancy probabilities is also important in state

assignment, re-encoding for low-power and power estimation via probabilistic or statistical approaches.

Addressing these issues, the present chapter extends the previous work reported in [19] to explicitly incorporate complex spatiotemporal correlations in steady-state and transition probabilities calculation for FSMs. The analysis itself relies on time-homogeneous discrete-parameter MCs which are used in two different ways:

- first, an input modeling MC is used to model the binary input stream that typifies the application data (called also *trace*) feeding the target FSM[1];
- second, a composite MC is used to model the serial connection of the input-modeling MC and the MC associated with the FSM itself.

In fact, using the joint transition probabilities of the primary inputs and internal states in the target machine, these two models can be merged. More precisely, if the sequence feeding the target circuit has temporal correlations of order *k*, then a lag-*k* MC model of the sequence will suffice to model correctly the joint transition probabilities of the primary inputs and internal states in the target circuit. As a consequence, we can use this accurate model to solve the set Chapman-Kolmogorov equations and, once the solution is determined, one can derive the state occupancy probabilities in a straightforward manner.

Studying the composite MC requires reachability analysis on the target machine. At this point, our work differs substantially from what other researchers have considered in the past, in the sense that our reachability analysis is constrained by the actual input

1. We point out that although the Markov model is derived for a particular input trace, it is completely general and represents in a compact form the whole class of input sequences having the same characteristics.

sequence and accounts for the very specific way in which the input source excites the target FSM.

Last but not least, MC analysis involves sophisticated numerical techniques; to date, Gauss-Jacobi and power method have been extensively used in steady-state probability calculation [19]. We present instead two different algorithms based on *stochastic complementation* and *iterative aggregation/disaggregation*, which provides, aside from a deep insight into theoretical aspects of MC analysis, an efficient solution for a large class of MCs, i.e. *nearly completely decomposable* (NCD) systems. The research presented in this chapter thus improves the state-of-the-art in two ways:

- first, based on *high-order Markov models*, it shows an effective analytic way to account for spatial and temporal correlations of the input sequence on steady-state probability calculation in standard FSMs;
- second, using the concept of *constrained reachability analysis*, it shows how the correct set of Chapman-Kolmogorov equations can be constructed;
- third, based on *stochastic complementation* and *iterative aggregation/disaggregation* techniques, it presents *exact* and *approximate* techniques for finding the state occupancy probabilities in the target machine.

This chapter is organized as follows. In Section 5.2 we formulate the problem we want to solve and present the basic Markov model. Section 5.3 focuses on constrained reachability analysis issue. In Section 5.4 we present exact and approximate methods for steady-state probabilities calculation, and we point out some issues regarding complexity

and convergence of algorithms. Finally, we present some experimental results for common sequential benchmarks, and we conclude by summarizing our main contribution.

5.2 FSM steady-state analysis: problem formulation

In this section, we introduce formally the problem we have to solve and we present two Markov models that we use to solve it: one model is the MC associated to the state lines of the FSM and another one is the MC that models the input sequence that feeds the target FSM. Within this framework, we also analyze the effects of input statistics on FSM behavior.

5.2.1 FSM characterization

As used by Hachtel et al. in [17], the probabilistic behavior of an FSM can be studied by regarding its state transition graph (STG) as a MC. More precisely, attaching to each out-going edge of each state in the target FSM a transition probability that corresponds to that particular transition, one actually obtains a MC as defined in Chapter 2. Furthermore, studying the behavior of the underlying MC provides us different variables of interest in the original FSM. For instance, we may be interested in defining the MC associated to state lines of the FSM (denoted by $\{s_n\}_{n \geq 0}$) because, by virtue of Proposition 2.2 in Chapter 2, we may find the state occupation probabilities for the machine. To this end, the equation to solve is:

$$\pi \cdot Q_S = \pi \quad \text{with} \quad \sum_{all\ i} \pi_i = 1 \tag{5.1}$$

where π and $Q_S = (p_{ij})_{1 \le i, j \le m}$ denote the stationary distribution and the transition matrix of the chain, respectively.

To set up the above Q_S matrix, the authors in [17] consider that all input combinations are equiprobable during the normal operation of the machine and therefore, the one-step transition probability matrix can be obtained from the transition relation in a straightforward manner. However, in practice, the situation can be quite different: different input sequences may exercise the machine in different ways and thus produce substantially different STG structures. Due to the feedback lines, the behavior of the state lines themselves is strongly dependent on the characteristic of the input sequences present at primary inputs and therefore, to set up the Q_S matrix which actually accounts for the influence of correlations at the primary inputs on the state lines of the FSM is a key (and nontrivial!) task. To construct the exact Q_S matrix that corresponds to the actual input trace, we also associate a finite-order MC to the primary input stream; we assume that this MC is described by the matrix $Q_X = (q_{ij})_{1 \le i, j \le l}$. To see what difference can make the actual input sequence in FSM analysis, let us consider the following example.

Example 5.1 Let S_1 and S_2 be two 2-bit sequences, of length 48, as shown in Figure 5.1(a). These two sequences, have exactly the same set of first-order temporal statistics that is, they cannot be distinguished as far as wordwise one-step transition probabilities are concerned. In fact, in Figure 5.1(b) we provide the wordwise transition graph for these two sequences. Each node in this graph is associated to a distinct pattern that occurs in S_1 and S_2 (the topmost bit is the most significant one, e.g. in S_1, v_1 = '1', v_2

= ‘2’, v_3 = ‘3’,..., v_{48} = ‘1’). Each edge represents a valid transition between any two valid patterns and has a nonzero probability associated with it. For instance, the pattern ‘3’ in S_1 and S_2 is always followed by ‘1’ (thus the edge between nodes ‘3’ and ‘1’ has the probability 1) whereas it is equally likely to have either ‘0’, ‘2’ or ‘1’ after pattern ‘1’.

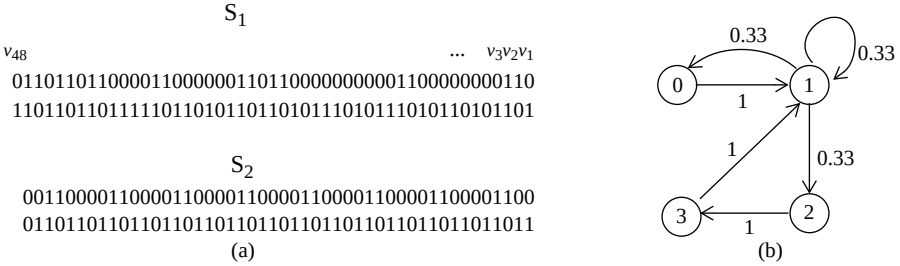


Figure 5.1 Two sequences with the same first-order characteristics

Starting with different initial states and using a random number generator we may, of course, generate other sequences equivalent with S_1 and S_2 as far as the one-step transition probabilities are concerned. We can then see the graph in Figure 5.1(b) as a compact, canonical, characterization of sequences S_1 and S_2. Suppose that we want to compute the occurrence probability of string v = ‘01 10’ that is, the probability that transition $1 \rightarrow 2$ is taking place in S_1. To this effect, we just use $p(v) = p(v_1 v_2) = p(v_1) \cdot p(v_2 | v_1)$ which gives us the value of 1/6. If we are interested in finding the two-step transition probability $0 \rightarrow 1 \rightarrow 1$ in S_2, then we use the formula $p(v) = p(v_1 v_2 v_3) = p(v_1) \cdot p(v_2 | v_1) \cdot p(v_3 | v_2 v_1)$, and then we get the value of 1/6.

If we consider now that the sequence S_1 is applied to the benchmark *dk17*, then using the STG-based calculations when all input combinations are equally likely to occur, one can find the matrix shown in Figure 5.2(b). On the other hand, simulating the actual

sequence S_1 and analyzing the state occupancy probabilities of the circuit, one can construct the matrix $Q_S{}^{trace}$ shown in Figure 5.2(c).

S_1

v_{48} ... $v_3v_2v_1$

011011011000011000000110110000000000011000000000110

110110110111110110101101101011010111010110101101101

(a)

$$Q_S^{equi} = \begin{bmatrix} 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0.25 & 0 & 0.25 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0 & 0.25 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0.75 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0.25 & 0 & 0 & 0.25 & 0 \\ 0.5 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0.25 \\ 0.5 & 0 & 0 & 0.25 & 0.25 & 0 & 0 & 0 \end{bmatrix}$$

(b)

$$Q_S^{trace} = \begin{bmatrix} 0 & 0.47 & 0 & 0 & 0.53 & 0 & 0 \\ 0 & 0.25 & 0.5 & 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0.39 & 0.61 & 0 & 0 & 0 \\ 0.46 & 0 & 0.54 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.77 & 0 & 0 & 0.23 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0.33 & 0 & 0.33 & 0.33 & 0 & 0 & 0 \end{bmatrix}$$

(c)

Figure 5.2 Building the Q_S matrix in two different hypotheses

Obviously, solving equation (5.1) for these two matrices will result in two very different solutions: the matrix in Figure 5.2(b) gives the solution of equation (5.1) as [0.22 0.15 0.01 0.25 0.17 0.15 0.04 0.01], while in the second case we get the solution [0.12 0.08 0.42 0.26 0.09 0.02 0.02]. This shows that accounting for correlations at the primary inputs is important for calculating steady-state probability distribution of the FSM. In fact, this type of dependence is mentioned by the authors in [19] but they account for it only from a signal probability perspective. More precisely, they allow certain signals in their analysis to be fixed at 0 or 1 or have a signal probability different that 0.5 and for this new STG (obtained from the original one by deleting some edges), they perform steady-state probability analysis using the power method for the underlying matrix Q_S. Obviously, this solution has only limited applicability because it is very common in practice to have much

more complicated temporal correlations among at the primary input lines. In the following, we investigate the effect of input statistics on FSM behavior.

5.2.2 The effect of input sequence on FSM behavior

Referring to the general FSM in Figure 5.3, we proposed up to this point two interacting Markov models: one for the primary inputs $\{x_n\}_{n \geq 0}$ (which characterizes the trace) and another one, dependent on the first one, for the state lines $\{s_n\}_{n \geq 0}$ (which characterizes the machine itself). In fact, these two models can be conceptually merged via the joint probabilities $p(x_n s_n)$ and $p(x_n s_n x_{n-1} s_{n-1})$; as we can see in Figure 5.3, they completely characterize the input that feeds the next state and the output logic of the target circuit.

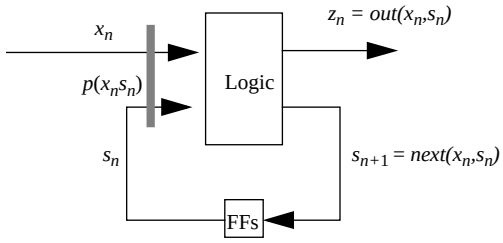


Figure 5.3 Modeling the target FSM

Under the general assumption of ergodicity we can prove the following result:

Theorem 5.1 If the sequence feeding a target sequential circuit has order k, then a lag-k MC which correctly models the input sequence, also correctly models the k-step conditional probabilities of the primary inputs and internal states in the target circuit, that is $p(x_n s_n | x_{n-1} s_{n-1} x_{n-2} s_{n-2} \ldots x_{n-k} s_{n-k}) = p(x_n | x_{n-1} x_{n-2} \ldots x_{n-k})$.

Proof: Let $p(x_n s_n x_{n-1} s_{n-1} \ldots x_{n-k} s_{n-k})$ be the joint transition probability for inputs and states over k consecutive time steps. Then we have:

$$p(x_n s_n \ldots x_{n-k} s_{n-k}) = \begin{cases} p(x_n x_{n-1} \ldots x_{n-k} s_{n-k}) & \text{if } next(x_i, s_i) = s_{i+1} \quad i = n-k, \ldots, n-1 \\ 0 & \text{otherwise} \end{cases} \quad (5.2)$$

For the first alternative we have:

$$p(x_n s_n \ldots x_{n-k} s_{n-k}) = p(x_n x_{n-1} \ldots x_{n-k} s_{n-k}) = p(x_n s_{n-k} | x_{n-1} \ldots x_{n-k}) \cdot p(x_{n-1} \ldots x_{n-k}) \quad (5.3)$$

Since x_n is a lag-k MC, x_n depends only on x_{n-1}, x_{n-2},..., x_{n-k}. On the other hand, s_{n-k} is a function of x_{n-k-1} and s_{n-k-1}. Thus, when x_{n-1}, x_{n-2},..., x_{n-k} are fixed, x_n and s_{n-k} are independent, that is:

$$p(x_n s_{n-k} | x_{n-1} x_{n-2} \ldots x_{n-k}) = p(x_n | x_{n-1} x_{n-2} \ldots x_{n-k}) \cdot p(s_{n-k} | x_{n-1} x_{n-2} \ldots x_{n-k})$$

or equivalently, using equation (5.3), we obtain:

$$p(x_n s_n \ldots x_{n-k} s_{n-k}) = p(x_n | x_{n-1} x_{n-2} \ldots x_{n-k}) \cdot p(x_{n-1} x_{n-2} \ldots x_{n-k} s_{n-k}) .$$

Dividing both sides by $p(x_{n-1}x_{n-2}...x_{n-k}s_{n-k})$ and using the second part of equation (5.3) we obtain exactly $p(x_n s_n|x_{n-1}s_{n-1}x_{n-2}s_{n-2}...x_{n-k}s_{n-k}) = p(x_n|x_{n-1}x_{n-2}...x_{n-k})$. For the second alternative, if $x_n s_n x_{n-1} s_{n-1} x_{n-2} s_{n-2} ... x_{n-k} s_{n-k}$ is not a valid sequence, then $p(x_n s_n|x_{n-1}s_{n-1}x_{n-2}s_{n-2}...x_{n-k}s_{n-k}) = 0$ and this concludes our proof. ■

We note therefore that preserving order-k statistics implies also that order-k statistics will be captured for inputs and states. In general, modeling an order-k source by a lower order source may introduce large inaccuracies as shown in the next example.

Example 5.2 We consider once again the sequences S_1 and S_2 that feed the benchmark *dk17* and illustrate that indeed, if the input sequence has order two, then modeling it as a lag-one MC will *not* preserve the first-order joint transition probabilities (primary inputs and internal states) in the target circuit. We simulated the benchmark *dk17 (*starting with

the same initial state *‘19’)* for both sequences and we present in Figure 5.4 (Figure 5.4(a) is for S_1 and Figure 5.4(b) is for S_2) the wordwise transition graphs obtained for the signal lines ($x_n s_n$).

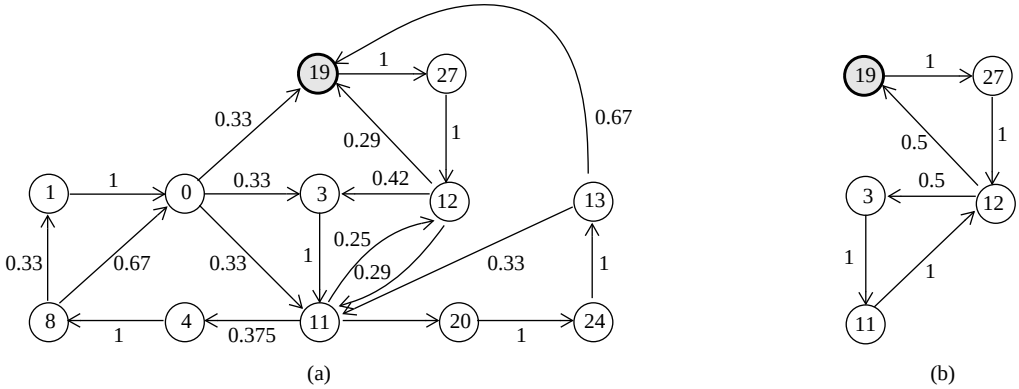


Figure 5.4 The MCs for the input and state lines

The benchmark *dk17* has 2 primary inputs and 3 FFs therefore in Figure 5.4, any node is decimally encoded using 5 bits. For instance, the initial state *‘19’* corresponds to the binary code ‘10011’ that is, ‘10’ for primary inputs and ‘011’ for state lines. After that, applying ‘11’ on the primary inputs, the present state becomes ‘011’, therefore we enter the node ‘11011’ = 27 in Figure 5.4. As we can see, because S_1 can be modeled as a first order Markov source, while S_2 must be modeled as a second order Markov source, the corresponding transition graphs are quite different. From a practical point of view, this means that if one underestimates a high-order source (for instance, assuming that second- or higher-order temporal correlations are not important), then one may end up not preserving even the first-order transition probabilities in the target circuit.

To conclude this section, we note that these findings not only clarify the relationship between the primary inputs and state lines of the FSM, but also provide a theoretical

justification for considering high-order temporal models in steady-state probability analysis of FSMs.

5.3 Sequence-driven reachability analysis

In this section we will first present a theoretical framework for sequence-driven reachability analysis, followed by a practical solution to this problem. We point out that sequence-driven reachability analysis differs from classical reachability analysis in that it accounts for constraints on the inputs, that is, the possible set of input vectors applicable to the circuit and their sequencing.

In Chapter 3 it has been shown that to any first-order MC, one can associate a *Stochastic Sequential Machine* (SSM) that generates symbols according to the conditional probabilities of the initial MC. Specifically, a synthesis procedure for the SSM modeling the input sequence has been proposed by the authors. Based on this synthesis procedure, the target FSM and its input can be viewed as in Figure 5.5(a).

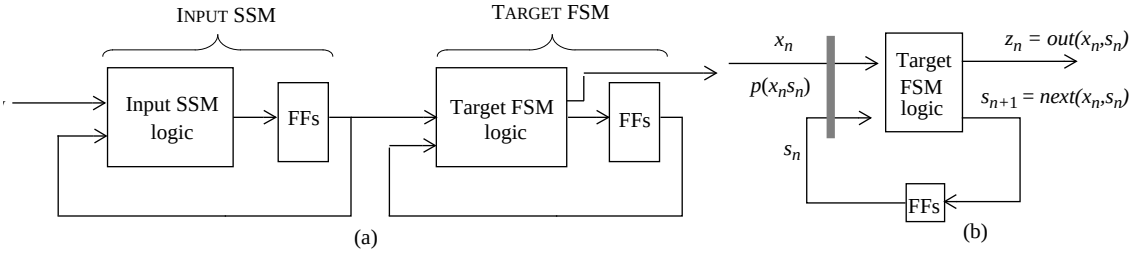


Figure 5.5 A model for FSM analysis using input sequence modeling

The primary inputs of the SSM in Figure 5.5(a) (called *auxiliary inputs*) are generated according to a probability distribution such that the states of the SSM have exactly the behavior of the inputs of the target FSM. Thus, the analysis can be done on the *product machine* (*input SSM, target FSM*) which has temporally uncorrelated inputs with a

prescribed probability distribution. What we are interested in, is the probability distribution on the state lines of the target FSM.

Referring to the general FSM in Figure 5.5(b), we proposed up to this point two interacting Markov models: one for the primary inputs $\{x_n\}_{n \geq 0}$ (which characterizes the trace) and another one, for the state lines $\{s_n\}_{n \geq 0}$ (which characterizes the behavior of the machine itself). In fact, these two models can be conceptually merged via joint transition probabilities $p(x_n s_n)$ and $p(x_n s_n x_{n-1} s_{n-1})$; as we can see in Figure 5.5(b), these probabilities completely characterize the input that feeds the next state and the output logic of the target circuit.

To make the presentation more clear, we introduce a new formulation based on matrices, mostly because this is a convenient notation for our purpose. As we can see in Figure 5.5(b), the first-order MC at the primary inputs x_n can be characterized by the matrix of conditional probabilities $Q_X = (q_{ij})_{1 \leq i,j \leq l}$, where $q_{ij} = p(x_n = v_j | x_{n-1} = v_i)$ and $\{v_1, v_2, ..., v_l\}$ represents the set of possible input patterns. On the other hand, the MC defined jointly for the primary inputs and state lines $\{x_n s_n\}_{n \geq 0}$ can be characterized by the matrix $Q_{XS} = (r_{ip,jq})_{\substack{1 \leq i,j \leq l \\ 1 \leq p,q \leq m}}$, where $r_{ip,jq} = p(x_n s_n = v_j u_q | x_{n-1} s_{n-1} = v_i u_p)$ and $\{u_1, u_2, ..., u_m\}$ represents the set of reachable states. (Q_{XS} is the stochastic matrix of the product machine mentioned above.) From this joint characterization, we can easily derive the state probabilities as: $p(s_n) = \sum_{all\ x_n} p(x_n s_n)$ which is actually our variable of interest. Based on

Theorem 3.1, the following result provides the starting point in finding the correct matrix Q_{XS}.

Proposition 5.1 The matrix Q_{XS} can be written in the form $Q_{XS} = \begin{bmatrix} q_{11}B_1 & q_{12}B_1 & \cdots & q_{1l}B_1 \\ q_{21}B_2 & q_{22}B_2 & \cdots & q_{2l}B_2 \\ \cdots & \cdots & & \cdots \\ q_{l1}B_l & q_{l2}B_l & \cdots & q_{ll}B_l \end{bmatrix}$,

where $\{B_i\}_{1 \le i \le l}$ is a set of $m \times m$ degenerate[1] stochastic matrices defining the next state function for input v_i; specifically, if $B_i = \left(b^i_{pq}\right)_{1 \le p,q \le m}$ then $b^i_{pq} = \begin{cases} 1 & \text{if } next(v_i, u_p) = u_q \\ 0 & \text{otherwise} \end{cases}$. ■

Corollary 5.1 The product machine (*input SSM*, *target FSM*) (as in Figure 5.5(a)) is also a SSM whose auxiliary inputs are excited using the same probability distribution as the one used for the input SSM.

Proof: From Theorem 3.1, $Q_X = \sum_i p_i \cdot U_i$ where U_i's are degenerate stochastic matrices.

We can therefore write $Q_{XS} = \sum_i p_i \cdot V_i$ where V_i's are degenerate matrices having the form

$V_i = \begin{bmatrix} u^i_{11}B_1 & u^i_{12}B_1 & \cdots & u^i_{1l}B_1 \\ u^i_{21}B_2 & u^i_{22}B_2 & \cdots & u^i_{2l}B_2 \\ \cdots & \cdots & & \cdots \\ u^i_{l1}B_l & u^i_{l2}B_l & \cdots & u^i_{ll}B_l \end{bmatrix}$; B_i's are as in Proposition 5.1 and u^i_{jk} are the elements of matrix U_i. ■

1. That is, the elements of the B_i matrices are only 0 or 1.

From this point on, in order to compute the steady-state probability for the state lines of the target machine, we can apply any existing approach that computes the probabilities for the states of the product machine (*input SSM, target FSM*) which has the virtue of having temporally uncorrelated inputs. However, this approach can be very inefficient: the task of synthesizing the exact input SSM may require huge memory and computation time. Instead, we propose to model the input as a *Dynamic Markov Tree* of order 1 (DMT_1) [37].

The DMT_1 model contains information about not only the possible binary vectors that can appear on the inputs of the FSM, but also the sequencing of these vectors. Additionally, the wordwise conditional probabilities for the primary inputs are easily extracted from such a model. The benefits of using DMT_1 for input modeling are threefold:

- first, the structure DMT_1 is constructed "on demand", therefore it offers a very compact representation;
- second, the model provides a set of parameters that completely capture spatiotemporal correlations;
- third, its structure is compatible with that of *Binary Decision Diagrams* (BDDs) [4] which have been successfully used in reachability analysis for FSMs [9][58]. To see how these advantages can be exploited, let us take the following example.

Example 5.3 For the sequence S_1 in Figure 5.1, the DMT_1 is given in Figure 5.6(a). The node labels (a_0, b_0) represent the variables encoding the input bits for the current time

step, whereas (a_1, b_1) correspond to the next time step. Left edges are associated with bits equal to 0, right edges to bits equal to 1. Each edge that enters a node is labeled with a positive count that represents the number of times the substring from the root to that particular node appeared in the original sequence. The upper half of this DMT_1 captures the statistics of order zero for the input sequence, whereas the lower half captures the sequencing between any two consecutive vectors. Based on this information and on the edge counts, we can easily compute the conditional or steady-state probabilities for the inputs. For example, p (00) = 5/48 (5 denotes the number of times 00 appears in the input sequence and 48 is the total length of the sequence) and p (01|01) = p (0101) / p (01) = 9/ 27. Also, for the above DMT_1, the corresponding BDD is depicted in Figure 5.6(b). Every possible combination with non-zero probability of occurrence in DMT_1 is part of the ON-set of the corresponding BDD; everything else, represents the OFF-set.

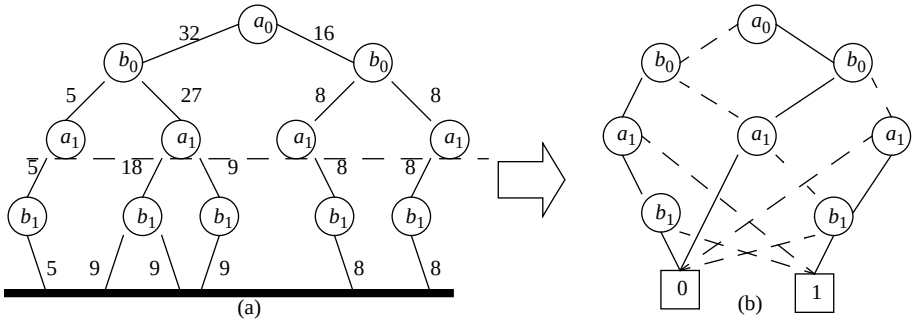


Figure 5.6 The tree DMT_1 and the corresponding BDD

The BDD corresponding to a given DMT_1 actually represents the transition relation δ of the machine that models the *input* to the target FSM (that is "INPUT SSM" in Figure 5.5(a)). Having this representation for the input, our task is then to find all the reachable combinations (*input*, *state*). Let B = {0, 1}, N the number of primary input

variables and δ: $B^N \times B^N \rightarrow B$ defined as $\delta(x^-, x^+) = 1$ if vector x^+ can follow x^- on the input modeled as a lag-one MC (that is, if the corresponding conditional probability is non-zero) and zero otherwise. Also, let M be the number of state variables and *next*: $B^M \times B^N \rightarrow B^M$ be the next state function of the FSM. Then, we may employ the following standard procedure [58] to compute the set C of reachable combinations (*input, state*) for the given FSM and input characterization:

$C_0 = \{(x_0, s_0) \mid x_0$ is any possible initial input, s_0 is any possible initial state$\}$

$C_{i+1} = C_i \cup \{(x, s) \mid \exists (x', s') \in C_i$ s.t. $\delta(x', x) = 1$ and $next(s', x') = s\}$

$C = C_i$ if $C_i = C_{i+1}$.

Example 5.4 Let's assume that $C_0 = \{(01, 011)\}$ for circuit *dk17*. Applying the above algorithm, we get successively $C_1 = C_0 \cup \{(00, 100), (01, 100), (10, 100)\}$, $C_2 = C_1 \cup \{(01, 000), (00, 011), (10, 011), (11, 000)\}$,..., $C_7 = C_6 = C = \{(01, 000), (11, 000), (00, 001), (01, 001), (10, 001), (00, 011), (01, 011), (10, 011), (11, 011), (00, 100), (01, 100), (10, 100), (01, 101), (11, 101), (01, 110), (00, 111), (01, 111), (10, 111)\}$.

The above steps can be performed completely symbolically with the aid of BDDs [9][58]. After having the complete set C of possible (*input, state*) combinations, we can easily build the matrix Q_{XS} based on Proposition 5.1. We should point out that in general, the matrix Q_{XS} 1) may have transient states or 2) may be decomposable. However, these can be dealt with in a similar way to the approach presented in [19] where: 1) transient states are eliminated and 2) the problem is reduced to finding the steady state distribution for each strongly connected component of the underlying MC. In what follows we will

thus refer only to irreducible MCs or matrices, that is those in which each state is reachable from any other state in a finite number of steps. This hypothesis has no theoretical limitation to be extended to the case of reducible MCs or MCs with multiple components.

This section has dealt only with constrained reachability analysis for inputs modeled as lag-one MCs. However, all the results can be easily generalized to lag-*k* MCs using the result in Proposition 2.1.

5.4 Steady-state probability computation

We have constructed by now the correct matrix Q_{XS} (which completely characterizes the FSM behavior) and are therefore ready to solve the basic equation $\pi \cdot Q_{XS} = \pi$ (with $\Sigma \pi_i = 1$). To this end, we first present an exact approach based on stochastic complementation and after that, because in practice Q_{XS} can be a fairly large matrix, we describe an approximate method based on iterative aggregation/disaggregation techniques.

5.4.1 Classical methods

Finding the stationary distribution of MCs with relatively few states is not a difficult problem and standard techniques based on solving systems of linear equations do exist [55]. In order to find the stationary distribution of a MC, one can always employ direct or iterative methods to solve the Chapman-Kolmogorov equations. The direct methods (e.g Gaussian elimination, LU decomposition) always provide the result in a fixed number of steps. On the other hand, iterative methods (e.g. power method, Jacobi, Gauss-Seidel) start with an initial approximation of the solution and iteratively improve on it. The number of

iterations until convergence depends on the actual characteristics of the matrix. However, both types of methods work with the stochastic matrix as a whole. When the matrix size is large, we must resort to decompositional methods that try to solve smaller problems and then aggregate their solutions to find the needed stationary distribution.

5.4.2 Stochastic complementation

For large-scale problems, it is only natural to attempt to somehow uncouple (or decompose) the original MC into smaller chains (which are therefore easier to analyze) and finally, having these partial solutions, to produce the global stationary distribution that corresponds to the original chain. The stochastic complementation approach provides the theoretical basis for uncoupling MCs and furthermore, when applied to matrix Q_{XS}, provides as a by-product the matrix Q_S as we shall see later in one of our results.

Definition 5.1 (stochastic complement) [39] Let Q be a $n \times n$ irreducible stochastic matrix partitioned as

$$Q = \begin{bmatrix} Q_{11} & Q_{12} & \cdots & Q_{1p} \\ Q_{21} & Q_{22} & \cdots & Q_{2p} \\ & \cdots & & \cdots \\ Q_{p1} & Q_{p2} & \cdots & Q_{pp} \end{bmatrix} \tag{5.4}$$

where all diagonal blocks are square. For a given index i, let Q_i denote the principal block submatrix of Q obtained by deleting the i^{th} row and i^{th} column of blocks from Q, and let Q_{i*} and Q_{*i} designate:

$$Q_{i*} = (Q_{i1} Q_{i2} \cdots Q_{i,i-1} Q_{i,i+1} \cdots Q_{ip}) \qquad \text{and} \qquad Q_{*i} = \begin{bmatrix} Q_{1i} \\ \vdots \\ Q_{i-1,i} \\ Q_{i+1,i} \\ \vdots \\ Q_{pi} \end{bmatrix}^T \tag{5.5}$$

That is, Q_{i*} is the i^{th} row of blocks with Q_{ii} removed and Q_{*i} is the i^{th} column of blocks with Q_{ii} removed. The stochastic complement of Q_{ii} is defined to be the matrix

$$S_{ii} = Q_{ii} + Q_{i*} \cdot (I - Q_i)^{-1} \cdot Q_{*i} \tag{5.6}$$

where I is the unit matrix.

It can be shown that every stochastic complement in Q is also an irreducible matrix. In addition, the following theorem has been proved [39]:

Theorem 5.2 Let Q be an $n \times n$ irreducible stochastic matrix as in equation (5.4) whose stationary probability vector π can be written as $\pi = (\xi_1\Phi_1, \xi_2\Phi_3, \ldots, \xi_p\Phi_p)$ with $\Phi_i \cdot e = 1$ for $i = 1, 2, \ldots, p$; e is a column vector defined as: $e = (1,1,\ldots,1)^T$. Then Φ_i is the unique stationary probability vector for the stochastic complement S_{ii} and $\xi = (\xi_1, \xi_2, \ldots, \xi_p)$ is the unique stationary probability vector for the $p \times p$ irreducible stochastic matrix A (called the coupling matrix) whose entries are defined by $a_{ij} = \Phi_i \cdot Q_{ij} \cdot e$. ■

The coupling matrix A corresponds to the MC in which states belonging to the same set of the partition are collapsed in a single state. Thus, ξ describes the steady-state probability of being in such a set of states. Using this important theorem, the following exact procedure can be used to compute the stationary probability vector. (The input for this procedure is the Q matrix given in Definition 5.1).

1. Partition Q into a $p \times p$ block matrix with square diagonal blocks.
2. Form the stochastic complement of each diagonal block:
   $S_{ii} = P_{ii} + P_{i*}(I-P_i)^{-1}P_{*i}$, $i = 1, 2,..., p$.
3. Compute the stationary probability vector of each stochastic complement:
   $\Phi_i \cdot S_{ii} = \Phi_i$; $\Phi_i \cdot e = 1$, $i = 1, 2,..., p$.
4. Form the coupling matrix A whose elements are given by:
   $a_{ij} = \Phi_i \cdot Q_{ij} \cdot e$.
5. Compute the stationary probability vector of A:
   $\xi \cdot A = \xi$; $\xi \cdot e = 1$.
6. Construct the stationary probability vector of Q as:
   $\pi = (\xi_1\Phi_1, \xi_2\Phi_2, \ldots, \xi_p\Phi_p)$.

Figure 5.7 The stochastic complementation algorithm

Example 5.5 Let's consider the following matrix described in [10]:

$$Q = \begin{bmatrix} 0.85 & 0.0 & 0.149 & 0.0009 & 0.0 & 0.00005 & 0.0 & 0.00005 \\ 0.1 & 0.65 & 0.249 & 0.0 & 0.0009 & 0.00005 & 0.0 & 0.00005 \\ 0.1 & 0.8 & 0.0996 & 0.0003 & 0.0 & 0.0 & 0.0001 & 0.0 \\ 0.0 & 0.0004 & 0.0 & 0.7 & 0.2995 & 0.0 & 0.0001 & 0.0 \\ 0.0005 & 0.0 & 0.0004 & 0.399 & 0.6 & 0.0001 & 0.0 & 0.0 \\ 0.0 & 0.00005 & 0.0 & 0.0 & 0.00005 & 0.6 & 0.2499 & 0.15 \\ 0.00003 & 0.0 & 0.00003 & 0.00004 & 0.0 & 0.1 & 0.8 & 0.0999 \\ 0.0 & 0.00005 & 0.0 & 0.0 & 0.00005 & 0.1999 & 0.25 & 0.55 \end{bmatrix}$$

Assuming that the states are partitioned as {(1, 2, 3), (4, 5), (6, 7, 8)}, we compute the stochastic complements as in step 2. For example, $S_{11} = Q_{11} + [Q_{12} \;\; Q_{13}] \cdot \begin{bmatrix} I - Q_{22} & -Q_{23} \\ -Q_{32} & I - Q_{33} \end{bmatrix}^{-1} \cdot \begin{bmatrix} Q_{21} \\ Q_{31} \end{bmatrix} = \begin{bmatrix} 0.8503 & 0.0004 & 0.1493 \\ 0.1003 & 0.6504 & 0.2493 \\ 0.1001 & 0.8002 & 0.0997 \end{bmatrix}$ and its left eigenvector is $\Phi_1 = [0.4012 \;\; 0.4168 \;\; 0.1819]$. Doing the same for all stochastic complements (S_{22} and S_{33}), we can proceed to steps 4 and 5 which compute the coupling matrix and its left eigenvector ξ.

We get $A = \begin{bmatrix} 0.99911 & 0.00079 & 0.00010 \\ 0.00061 & 0.99929 & 0.00010 \\ 0.00006 & 0.00004 & 0.99990 \end{bmatrix}$ and ξ = [0.22333 0.27667 0.50000]. Using ξ and Φ_i's, we can now compute the exact stationary probability distribution as π = [0.0893 0.0928 0.0405 0.1585 0.1189 0.1204 0.2778 0.1018].

We point out that the analysis based on stochastic complementation does not depend in any way on matrix Q being NCD and when implementing the stochastic complement approach, we may choose a partitioning solution that is convenient for us. For instance, based on the functionality of the FSM, we may partition matrix Q_{XS} such that all combinations XS having the same S are clustered in the same block. The reason for doing so is that this way matrix A in the stochastic complement algorithm becomes precisely the matrix Q_S (that is, the transition matrix associated with the state lines in the FSM) and ξ represents therefore the state occupancy probability we are looking for.

Theorem 5.3 If the stochastic complementation algorithm is applied to matrix Q_{XS} and the partitioning is done such that combinations (x_i, s_i), (x_j, s_j) are clustered in the same block if and only if $s_i = s_j$, then A is the matrix associated with the MC for the state lines Q_S and the corresponding probability distribution is given by ξ.

Proof: From the definition of the coupling matrix, A represents the stochastic matrix for the MC when all states (in our case, (x_i, s_i) combinations) belonging to one subset are collapsed into a single state. Since (x_i, s_i) and (x_j, s_j) are in the same subset if and only if $s_i = s_j$, the states of the collapsed MC are exactly $\{s_n\}_{n \geq 0}$, that is, the states of the target FSM. Also, their probability distribution is given by ξ. ■

This important result justifies basically the applicability of stochastic complementation to FSM analysis. We also note that this method has the important feature of being *exact*, but unfortunately, in the worst case is computationally inefficient. More precisely, step 2 is the bottleneck because it involves the inversion of a large matrix, that is

(I - P_i). However, the contribution of the stochastic complementation method lies primarily in the insight it provides into theoretical aspects of NCD systems.

5.4.3 Iterative aggregation/disaggregation

In this section, we present an iterative algorithm based on approximate decomposition that rapidly converges to the exact solution when the MC is NCD. The pioneering work on NCD systems comes from Simon and Ando [54] who studied the dynamic behavior of linear systems. The concept was later extended to performance analysis of computer systems by Courtois [10]. The whole idea behind the dynamic behavior of NCD systems is the existence of two regimes:

- a *short-run dynamics*, when strong interactions within each subsystem are dominant and quickly force each subsystem to a local equilibrium almost independently of what is happening in the other subsystems;
- a *long-run dynamics*, when weak interactions among groups begin to become important and the whole system moves toward a global equilibrium; in this global equilibrium the relative values attained by the states at the end of the short-run dynamics period are maintained.

As a consequence, the state space of the global MC can be partitioned into disjoint sets, with strong interactions among the states of a subset, but with weak interactions among the subsets themselves. This way, each subproblem can be solved separately and the global solution is then constructed from partial solutions.

Iterative aggregation/disaggregation (IAD) methods are particularly useful when the global MC is NCD. More precisely, IAD methods work on partitioned state space as an *aggregation* (or *coupling*) step followed by a *disaggregation* one. The coupling step involves generating a stochastic matrix of block transition probabilities (called *coupling* or *aggregation matrix*) and then determining its stationary probability vector. The disaggregation step computes an approximation to the probability of each state aggregated in the previous step within the same block. The basic iterative algorithm referred in the literature on numerical methods is called KMS (named after its authors Khoury-McAllister-Stewart) [25] and is described in Figure 5.8 (once again, the input of the algorithm is the matrix Q as in Definition 5.1).

1. Let $\pi^{(0)} = (\pi_1^{(0)}, \pi_2^{(0)},...,\pi_p^{(0)})$ be a given initial approximation to the solution π, and set $m = 1$.
2. Compute $\Phi^{(m-1)}$: for $i = 1, 2,..., p$ do
   $\Phi_i^{(m-1)} = \pi_i^{(m-1)} / \left\| \pi_i^{(m-1)} \right\|_1$ (the norm is defined as: $\|x\|_1 = \sum_j |x_j|$).
3. Construct the aggregation matrix $A^{(m-1)}$ whose elements are given by:
   $(A^{(m-1)})_{ij} = \Phi_i^{(m-1)} \cdot Q_{ij} \cdot e$.
4. Solve the eigenvector problem:
   $\xi^{(m-1)} \cdot A^{(m-1)} = \xi^{(m-1)}; \quad \xi^{(m-1)} \cdot e = 1$.
5. (a) Compute the row vector:
   $z^{(m)} = (\xi_1^{(m-1)}\Phi_1^{(m-1)}, \xi_2^{(m-1)}\Phi_2^{(m-1)},\ldots, \xi_p^{(m-1)}\Phi_p^{(m-1)})$.
   (b) Find $\pi^{(m)}$: for $k = 1, 2,..., p$ solve the system of equations:
   $$\pi_k^{(m)} = \pi_k^{(m)} \cdot Q_{kk} + \sum_{j<k} \pi_j^{(m)} \cdot Q_{jk} + \sum_{j>k} z_j^{(m)} \cdot Q_{jk}$$
6. Test for convergence: if the accuracy is sufficient, then stop and take $\pi^{(m)}$ as the solution vector; otherwise, set $m = m + 1$ and go to step 2.

Figure 5.8 The KMS algorithm

In this case, the partitioning criterion is purely numerical. States are aggregated within the same block if they interact strongly enough to favor short-run dynamics. This way, the off-diagonal elements in the global matrix are smaller than those in the blocks on the main diagonal and therefore, the interactions among subsets are minimized. In practice, finding

the partitioning of a NCD stochastic matrix is not a trivial task. One way to do it is to ignore the entries in the matrix that are less than some threshold ε and then find the strongly connected components of the underlying MC. Next, the threshold may be increased and the same analysis is done on the components already found. We should note that the lower the threshold, the higher the rate of convergence, at the expense of larger blocks. On the other hand, if we proceed in partitioning the state space further until it becomes manageable, the convergence rate slows down due to the larger threshold used. More formally, under fairly general conditions (see reference [25] for details), the following result holds for NCD stochastic matrices:

Theorem 5.4 If the matrix Q is partitioned such that $\|Q_{ii}\|_1 = O(1)$ and $\|Q_{ij}\|_1 = O(\varepsilon)$ for $i \neq j$, then the error in the approximate solution using the iterative aggregation/disaggregation algorithm is reduced by a factor of ε at each iteration.■

Example 5.6 Considering again the Courtois matrix in Example 5.5, we can see that taking 0.001 as a threshold value, we find three strongly connected components in the underlying MC: (1, 2, 3), (4, 5) and (6, 7, 8). Considering this partitioning for matrix Q (see the form of matrix Q in Example 5.5), applying the KMS algorithm produces the solution with error less than 10^{-10} in only 3 iterations. More specifically, considering as initial approximation $\pi^{(0)} = \left[\frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\ \frac{1}{8}\right]$, we get $\Phi_1^{(0)} = \left[\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}\right]$, $\Phi_2^{(0)} = \left[\frac{1}{2}\ \frac{1}{2}\right]$ and $\Phi_3^{(0)} = \left[\frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}\right]$. Next, in step 3, based on these values, we can compute the entries of the approximate aggregation matrix.

For example, $(A^{(0)})_{11} = \Phi_1^{(0)} \cdot Q_{11} \cdot e = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} 0.85 & 0.0 & 0.149 \\ 0.1 & 0.65 & 0.249 \\ 0.1 & 0.8 & 0.0996 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0.9992$. Doing this for all entries, we get the following aggregation matrix: $A^{(0)} = \begin{bmatrix} 0.9992 & 0.0007 & 0.0001 \\ 0.00065 & 0.9992 & 0.00015 \\ 0.000053 & 0.00016 & 0.99978 \end{bmatrix}$ which is also a stochastic matrix. Its stationary distribution is given by $\xi^{(0)} = [0.2979 \quad 0.3369 \quad 0.3651]$ and thus, in step 5(a) we compute the row vector:

$z^{(1)} = \left(\xi_1^{(0)}\Phi_1^{(0)} \quad \xi_2^{(0)}\Phi_2^{(0)} \quad \xi_3^{(0)}\Phi_3^{(0)}\right) = [0.0993 \quad 0.0993 \quad 0.09993 \quad 0.1684 \quad 0.1684 \quad 0.1217 \quad 0.1217 \quad 0.1217]$.

In step 5(b) we solve for the block components of $\pi^{(1)}$. For example, $\pi_1^{(1)}$ is the solution of the equation:

$\pi_1^{(1)} = \pi_1^{(1)} \cdot Q_{11} + \sum_{j=2}^{3} z_j^{(1)} \cdot Q_{j1}$. Doing this for all components of $\pi^{(1)}$, we get as a first approximation for π: $\pi^{(1)}$ = [0.0874 0.0908 0.0396 0.1564 0.1174 0.1184 0.2731 0.1001]. After this step, the error of this estimate (defined as $\|\pi Q - \pi\|_2$[1]) is 9.3581e-6, that is, less than 10^{-5}. Continuing this process, the error becomes less than 10^{-10} after 3 iterations.

In practice, the KMS algorithm offers the attractive feature of working on individual blocks instead of the whole matrix Q_{XS}. As a consequence, its complexity per iteration is given by $O(n_{max}^3)$, where n_{max} is the maximum order over all diagonal blocks in the partitioned matrix Q_{XS}. Compared to the classical power method (which is also iterative in

1. By definition, $\|x\|_2 = \sqrt{\sum_j x_j^2}$.

nature) [19], the KMS algorithm has the significant advantage of being applicable to *any irreducible* MC (aperiodic or periodic) and also having a *higher rate of convergence* for NCD systems. In these cases, a few iterations will suffice for all practical purposes. We should point out that we can always trade-off space versus time complexity: if the partitioning is such that n_{max} is still too large, we can use a higher value for the threshold such that the size of the largest subset (i.e. n_{max}) becomes manageable. In this case, the convergence rate will be smaller and thus, the time needed for convergence will increase. Also, the size of the coupling matrix will be larger and hence step 4 in the KMS algorithm becomes critical. However, there is a solution to this latter problem: when solving the Chapman-Kolmogorov equations for the coupling matrix, we can apply the KMS algorithm in a recursive manner. This approach is called the *hierarchical KMS algorithm*.

5.5 Experimental results

In this section we provide our experimental results for some common benchmark circuits. In particular, we compare the probability distribution for the states where the order of the input sequence is assumed to be one, against the case where the actual order of the source is taken into consideration.

We provide in Table 5.1 the results obtained for stochastic complementation when Fibonacci sequences (that is, second-order sequences) are applied at the primary inputs of some *mcnc*'91 and ISCAS'89 benchmarks. We were forced to consider at this point only small benchmarks because, as pointed out in Section 5.4, the exact method based on stochastic complementation is, in general, computationally inefficient. For each example,

an appropriate dynamic Markov tree has been built and based on it, a sequence-driven reachability analysis has been performed. Using the obtained set of reachable combinations (*input, state*) (denoted by #(*x, s*)), and sparse matrix techniques, the matrix Q_{XS} has been built and further used to determine steady-state probabilities. The partitioning was determined by the same functional criterion as in Theorem 5.3. We also report for each benchmark the number of reached states (#*s*) and their corresponding probabilities..

Table 5.1. Stochastic complementation for second-order input sequences

Circuit	PIs/FFs	#(*x*,*s*)	#*s*	State probability distribution
bbara	4/4	28	3	[0.5 0.25 0.25]
bbtas	2/3	18	6	[0.0556 0.3333 0.3333 0.0556 0.0556 0.1667]
dk17	2/3	6	2	[0.6667 0.3333]
donfile	2/5	9	5	[0.1667 0.1667 0.1667 0.1667 0.3333]
s400	3/21	12	3	[0.6667 0.1667 0.1667]
s526	3/21	17	9	[0.3333 0.0833 0.0833 0.0833 0.0833 0.0833 0.0833 0.0833 0.0833]

In Table 5.2, we report our results obtained when applying the KMS algorithm for an extended set of benchmarks. Once again, the input sequences were generated using Fibonacci series. For comparison, we also provide the results when the input is considered by default as having order one. For each case, the sequence-driven reachability analysis is carried out as above and then, using the Q_{XS} matrix, the KMS algorithm is applied using a numerical partitioning criterion with a threshold of 0.001. In all cases, we report the number of (*input, state*) combinations reached and the number of reached states. The number of iterations needed to converge for an error less than 10^{-5} was ≤ 3 for the second-order model and between 7 and 15 for the first-order model. The average run-time per iteration was 2 sec. for each block-matrix on a Sparc 20 workstation. For the larger sized matrices, the hierarchical KMS algorithm was used. For comparison, for first-order

models we also provide the maximum and mean percentage errors obtained when comparing the results to the actual second order model (MAX% and MEAN%).

Table 5.2. First-order vs. the second-order model using the KMS alg.

		Second-order		First-order			
Circuit	PIs/FFs	#(*x*,*s*)	#*s*	#(*x*,*s*)	#*s*	MAX%	MEAN%
bbara	4/4	28	3	86	10	49.40	35.91
bbtas	2/3	18	6	20	6	55.70	24.28
dk17	2/3	6	2	18	7	37.48	29.96
donfile	2/5	9	5	14	6	52.70	26.76
ex1	9/5	769	11	1596	11	50.00	18.03
planet	7/6	192	34	3527	48	513.79	67.14
sand	11/5	977	32	17169	32	103.57	24.49
s1196	14/18	1536	329	1918	342	133.71	4.17
s1238	14/18	1538	330	1919	343	134.82	4.21
s1494	8/6	384	6	1372	40	28.69	8.92
s526	3/21	17	9	12155	4137	97.17	43.03
s820	18/5	680	8	3056	24	24.78	10.78

Table 5.3. The impact of the order of the input sequence

	Total power (μW@20MHz)	
Circuit	Second order	First order
bbara	747.38	786.53
bbtas	337.75	345.78
dk17	1439.71	1313.78
donfile	3020.55	2943.15
ex1	3165.66	3082.86
planet	8550.92	6429.16
sand	7883.13	8162.71
s1196	7027.45	7074.17
s1238	7755.56	7648.88
s1494	5367.42	5082.92
s526	1292.74	1342.12
s820	4362.78	4275.08

As we can see, considering the input of the target FSM as having only one-step temporal correlations may significantly impair the ability of predicting the correct number of reached states. In addition, for the subset of states correctly found as being reached, there is a significant error in the value of steady-state probabilities and total power consumption. For example, when excited using a second-order type of input, benchmark *planet* has a number of 34 reached states, whereas if the order is (incorrectly) assumed to be one, the number of reached states becomes 48. Moreover, the error in predicting the steady-state probability can be as high as 513% for the first-order model. Generally speaking, a lower order model covers all the states from the original one, but it may also introduce a significant number of extra states and, furthermore, the quality of the results in estimating the steady-state probabilities is seriously impaired.

Since the values of these probabilities strongly affect the total power values, we also show the impact of these results on probabilistic power estimation techniques. Knowledge of the steady-state probability distribution for inputs and states and the conditional probabilities from matrix Q_{XS}, allow us to reduce the problem of power estimation for sequential circuits to the one for combinational circuits; therefore techniques like [36][53] can be successfully applied. We provide in Table 5.3 a comparison between the values of power estimated when the order of the input sequences was considered arbitrarily as being one vs. those determined when the actual order has been taken into account. As we can see, underestimating the actual order of the input sequence, strongly impacts the accuracy of the values of total power consumption; the error introduced can be as much as 25% for circuit *planet*.

5.6 Summary

In this chapter we investigated from a probabilistic point of view the effect of finite-order statistics of the input sequence on FSMs behavior. In particular, the effect of temporal correlations longer that one clock-cycle was analyzed for steady-state and transition probabilities calculations. The results presented in this chapter can be used in low-power design, synthesis and verification and represent an important step towards understanding the FSM behavior from a probabilistic point of view.

Chapter 6 Conclusion and Future Direction

6.1 Thesis summary

With the growing need for low-power electronic circuits and systems, power estimation and low-power optimization have become crucial tasks that must be addressed in addition to circuit area, speed, cost and reliability. It is expected that, in the forthcoming years, power issues will receive increasing attention due to the widespread use of portable applications and the desire to reduce packaging and cooling costs of high-end systems. Thus, there is a whole range of applications that will benefit from techniques and mechanisms of *reducing* power consumption. This, however, requires availability of proven methodologies and techniques for power *analysis* and *estimation*. The present thesis offers an answer for the problem of accurately, yet efficiently, estimating the average power consumption in CMOS digital circuits. The techniques described in the present research can be applied for circuits specified at different levels of abstraction, from circuit/gate level to RT/behavioral level.

In Chapter 3 we proposed a novel technique for estimating the dynamic power consumption at circuit and gate level. More precisely, based on the concept of stochastic sequential machine, we presented an effective technique for compacting a long sequence of vectors into a much shorter input sequence so as to reduce the simulation time by orders of magnitude. The mathematical foundation of the approach relies on probabilisitc automata theory based on which a general synthesis procedure for SSM synthesis was presented. The newly synthesized SSM can be used in a stand-alone mode for constrained sequence generation or compaction, providing up to three orders of magnitude compaction ratios without much loss in accuracy (6% on average).

Chapter 4 addressed the problem of estimating power consumption at higher levels of abstraction. For this purpose, we proposed two new information-theoretic measures for estimating the average switching activity per module. Based on the concepts of entropy and informational energy, each module is characterized in terms of its average switching activity. The approach requires information about the functionality of the module or (if more detailed information is available) about its structure (i.e., topology, distribution of gates per level etc.). Simple closed form expressions were derived for the average switching activity per module, which can be further used in conjunction with capacitance estimates to deduce total power consumption. Because the approach is not simulative, it is extremely fast, yet it produces sufficiently accurate power estimates (10% error on average).

Finally, Chapter 5 investigated, from a probabilistic point of view, the effect of finite-order statistics of the input stream on the finite-state machine (FSM) behavior. As the main theoretical contribution, we extended the previous work done on steady-state probability calculation in FSMs to handle high-order correlations on the primary inputs. Formally, we proved that assuming temporal independence or using first-order temporal models for the external input stream is insufficient and may induce significant inaccuracies. More precisely, not only the set of reachable states may be incorrectly determined, but also the steady-state probabilities can be off by more than 100% from the correct ones. To correctly solve the problem, we proposed constrained reachability analysis to extract the exact set of reachbale states, and then, based on aggregation/disaggregation techniques, the correct steady-state probabilities are found.

The results presented in this thesis represent a significant step towards a better understanding of the behavior of digital circuits from a probabilistic point of view. They are applicable to not only power estimation, but also other areas where probabilistic modeling is appropriate.

6.2 Future work

Following a natural trend, the interests of researchers have lately shifted to the investigation of high-level power modeling, estimation, synthesis and optimization techniques. However, the research done in behavioral and RT-level power exploration is still in its infancy. As designs become larger and more complex, design reuse becomes much more prevalent and this leads to a requirement for efficient, high-level power models for complex macrocells (also referred to as IP blocks). These models must be capable of being utilized at various abstraction levels (since it will not be feasible to verify the performance of such large designs at transistor level in a reasonable amount of time). A starting point is presented in Chapter 4 where two measures were proposed to characterize circuits at different levels of abstraction.

In addition to designing power efficient hardware, application software will also be judged in terms of power efficiency. Thus, an increase in the level of abstraction at which energy estimation or reduction should be considered is expected. More precisely, energy measures at system and software (or algorithmic) levels are desired and they should be the main target of future CAD research. As a starting point, finding good theoretical bounds

for the switching activity in finite-state machines specified at high-levels of abstraction is of great interest [34].

With the increasing usage of Digital Signal Processor (DSP) and Application Specific Instruction Set Processor (ASIP) architectures, embedded systems have gained importance in the design automation community. Most of the work done in the area of code generation for embedded systems has mainly targeted code size and timing constraints. This is because the classical techniques for code optimization have not been able to generate code that efficiently uses the features of DSP or ASIP architectures and thus, most of the research has been done to overcome this shortcoming. However, very little has been done in code generation that uses power or energy as a supplementary constraint for optimization. In conjunction with finding new energy metrics at the software level, future research should concentrate on using these type of metrics to intelligently generate and/or optimize code for low power in embedded systems.

References

[1] V. Agrawal, 'An Information Theoretic Approach to Digital Fault Testing,' in *IEEE Trans. on Computers*, vol.C-30, no.8, pp. 582-587, Aug. 1981.

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

[3] L. Brillouin, 'Science and Information Theory,' Academic Press, New York, 1962.

[4] R. E. Bryant, 'Graph Based Algorithms for Boolean Function Manipulation,' in *IEEE Trans. on Computers*, vol.C-35, no.8, pp. 677-691, Aug. 1986.

[5] 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.

[6] K.-T. Cheng, V. Agrawal, 'An Entropy Measure for the Complexity of Multi-Output Boolean Functions,' in *Proc. ACM/IEEE Design Automation Conference*, pp. 302-305, June 1990.

[7] 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.

[8] R.W. Cook, M.J. Flynn, 'Logical Network Cost and Entropy,' in *IEEE Trans. on Computers*, vol.C-22, no.9, pp. 823-826, Sept. 1973.

[9] O. Coudert, C. Berthet, and J.C. Madre, 'Verification of Sequential Machines Based on Symbolic Execution,' in *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, 1989.

[10] P.J. Courtois, 'Decomposability: Queueing and Computer System Applications,' Academic Press, New York, 1977.

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

[12] 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.

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

[14] 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.

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

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

[17] G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, ‘Probabilistic Analysis of Large Finite State Machines,’ in *Proc. ACM/IEEE Design Automation Conference*, pp. 270-275, June 1994.

[18] G. Hachtel, M. Hermida, A. Pardo, M. Poncino and F. Somenzi, ‘Reencoding Sequential Circuits to Reduce Power Dissipation,’ in *Proc. ACM/IEEE Intl. Conf. Computer-Aided Design*, pp. 70-73, Nov. 1994.

[19] G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, ‘Markovian Analysis of Large Finite State Machines,’ in *IEEE Trans. on CAD*, vol.15, no.12, pp. 1479-1493, Dec. 1996.

[20] L. Hellerman, ‘A Measure of Computational Work,’ in *IEEE Trans. on Computers*, vol.C-21, no.5, pp. 439-446, 1972.

[21] 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.

[22] B. Kapoor, ‘Improving the Accuracy of Circuit Activity Measurement,’ in *Proc. ACM/IEEE Design Automation Conference*, pp. 734-739, June 1994.

[23] J.G. Kemeny, J.L. Snell, and A.W. Knapp, ‘Denumerable Markov Chains,’ D.Van Nostrand Company, Inc., 1966.

[24] 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.

[25] R. Koury, D.F. McAllister, and W.J. Stewart, ‘Methods for Computing Stationary Distributions of Nearly-Completely-Decomposable Markov Chains,’ *SIAM Journal of Algebraic and Discrete Mathematics*, vol.5, no.2, pp. 164-186, 1984.

[26] P. Landman, J. Rabaey, ‘Power Estimation for High Level Synthesis,’ in *Proc. European Design Automation Conferenc*e, pp. 361-366, Feb.1993.

[27] P. Landman and J. Rabaey, ‘Black-Box Capacitance Models for Architectural Power Analysis,’ in *Proc.Intl. Workshop on Low Power Design*, pp. 165-170, April 1994.

[28] D. Marculescu, R. Marculescu, and M. Pedram, ‘Information Theoretic Measures for Energy Consumption at Register Transfer Level,’ in *Proc. of Intl. Symposium. on Low Power Design*, Dana Point, CA, pp. 81-86, April 1995.

[29] D. Marculescu, R. Marculescu, and M. Pedram, ‘Information Theoretic Measures for Power Analysis,’ in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* (Special Issue on Low Power Design), vol.15, no.6, pp. 599-610, June 1996.

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

[31] D. Marculescu, R. Marculescu, and M. Pedram, 'Stochastic Sequential Machine Synthesis with Application to Constrained Sequence Generation,' to appear in *ACM Trans. on Design Automation of Electronic Systems*.

[32] D. Marculescu, R. Marculescu, and M. Pedram, 'Sequence Compaction for Probabilistic Analysis of Finite State Machines,' in *Proc. ACM/IEEE Design Automation Conference*, pp. 12-15, June 1997.

[33] D. Marculescu, R. Marculescu, and M. Pedram, 'Trace-Driven Steady-State Probability Estimation with Application to Power Estimation,' in *Proc. Design Automation and Test in Europe Conf.*, pp. 774-779, Feb. 1998.

[34] D. Marculescu, R. Marculescu, and Massoud Pedram, 'Theoretical Bounds for Switching Activity Analysis in Finite-State Machines', to appear in *Proc. of Intl. Symposium. on Low Power Electronics and Design*, Monterey, CA, August 1998.

[35] R. Marculescu, D. Marculescu, and M. Pedram, 'Switching Activity Analysis Considering Spatiotemporal Correlations,' in *Proc. IEEE/ACM Intl. Conference on Computer Aided Design*, pp. 294-299, Nov. 1994.

[36] R. Marculescu, D. Marculescu, and M. Pedram, 'Efficient Power Estimation for Highly Correlated Input Streams,' in *Proc. ACM/IEEE Design Automation Conference*, pp. 628-634, June 1995.

[37] R. Marculescu, D. Marculescu, and M. Pedram, 'Adaptive Models for Input Data Compaction for Power Simulators,' in *Proc. IEEE Asian-South Pacific Design Automation Conference*, Japan, pp. 391-396, Jan. 1997.

[38] R. Mehra, J. Rabaey, 'High Level Estimation and Exploration,' in *Proc. Intl. Workshop on Low Power Design*, pp.197- 202, April 1994.

[39] C.D. Meyer, 'Stochastic Complementation, Uncoupling Markov Chains and the Theory of Nearly Reducible Systems,' *SIAM Reviews*, vol.31, no.2, pp. 240-272, 1989.

[40] 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.

[41] 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.

[42] F. N. Najm, 'Towards a High-level Power Estimation Capability,' in *Proc. of Intl. Symposium. on Low Power Design*, Dana Point, CA, pp. 87-92, April 1995.

[43] 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.

[44] M. Nemani and F. N. Najm, 'Towards a high-level power estimation capability,' in *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, vol.15, no.6, pp. 588-598, June 1996.

[45] O. Onicescu, ‘Theorie de l’information. Energie informationelle,’ Paris, C.R. Acad.Sci., Tome 263, 1966.

[46] O. Onicescu, V. Stefanescu, ‘Elements of Informational Statistics with Applications,’ in Romanian, Bucharest 1979.

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

[48] K. Parker and E.J. McCluskey, ‘Probabilistic Treatment of General Combinational Networks,’ in *IEEE Trans. on Computers*, vol. C-24, no.6, pp. 688-670, June 1975.

[49] P. Paulin, and J. Knight, ‘Force-directed Scheduling for the Behavioral Synthesis of ASICs,’ *IEEE Transactions on CAD*, vol.CAD-8, no.6, pp. 661-679, July 1989.

[50] A. Paz, ‘Introduction to Probabilistic Automata,’ Academic Press 1971.

[51] M. Pedram, ‘Power Minimization in IC Design: Principles and Applications,’ in *ACM Transactions on Design Automation of Electronic Systems*, vol.1, no.1, pp.1-54, Jan.1996.

[52] J. Savir, G.S. Ditlow, and P.H. Bardell, ‘Random Pattern Testability,’ in *IEEE Trans. on Computers*, vol.C-33, no.1, pp. 79-90, Jan. 1984.

[53] P. Schneider, U. Schlichtmann, and K. Antreich, ‘Decomposition of Boolean Functions for Low Power Based on a New Power Estimation Technique,’ in *Proc. Intl. Workshop on Low Power Design*, pp. 123-128, April 1994.

[54] H.A. Simon and A. Ando, ‘Aggregation of Variables in Dynamic Systems,’ *Econometrica*, vol.29, pp. 111-138, 1961.

[55] G.W. Stewart, ‘Introduction to Matrix Computations,’ Academic Press, New York, 1973.

[56] J. Storer, ‘Data Compression: Methods and Theory,’ Ch.1, Computer Science Press, 1988.

[57] K. Thearling, J. Abraham, ‘An Easily Computed Functional Level Testability Measure,’ in *Proc. Intl. Test Conference*, pp. 381-390, 1989.

[58] H.J. Touati, H. Savoj, B. Lin, R.K. Brayton, and A. Sangiovanni-Vincentelli, ‘Implicit State Enumeration of Finite State Machines Using BDDs,’ in *Proc. ACM/IEEE Intl. Conf. Computer-Aided Design*, pp. 130-133, Nov. 1990.

[59] 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.

[60] 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.

[61] 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

[62] N. Weste and K Eshraghian, ‘Principles of CMOS VLSI Design: A Systems Perspective,’ 2nd Edition, Addison-Wesley Publishing Co., 1994.

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