Constrained Sequence Generation Using Stochastic Sequential Machines

Diana Marculescu, Radu Marculescu and
Massoud Pedram

CENG 96-16

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

March 1996

Constrained Sequence Generation Using Stochastic Sequential Machines

Diana Marculescu, Radu Marculescu, Massoud Pedram
Department of Electrical Engineering-Systems
University of Southern California, Los Angeles, CA 90089

Abstract

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

1. Introduction

In the past, time and area were the primary concerns of ICs designers. More recently, circuit testability was added as yet another important consideration during the design process. With the growing need for low-power electronic circuits and systems, power estimation and low-power optimization have become crucial tasks that must be also addressed. It is expected that, in the forthcoming years, power issues will receive increasing attention due to the widespread use of portable applications and the desire to reduce packaging and cooling costs of high-end systems.

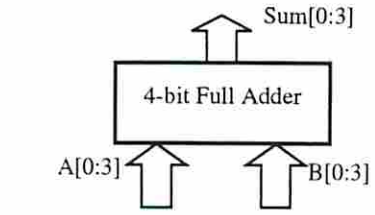
Power estimation techniques must be fast and accurate in order to be applicable in practice. Not surprisingly, these two requirements interfere with one another and at some point they become contradictory. General simulation techniques can provide sufficient accuracy, but the price to be payed is too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

A major 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 as *'the circuit problem'*, is by no means trivial. Indeed, a whole set of solutions have been proposed, ranging from approaches which build the global OBDDs [2][3] and therefore capture all internal dependencies, to efficient techniques which partially account for dependencies in an incremental manner [4]-[7].

Recently, some authors have pointed out the importance of correlations not only inside the target circuit, but also at the circuit inputs. If one ignores these correlations, the power estimation results can be impaired [8][9]. We will refer to this as *'the input problem'* which is important not only in power estimation, but also in low-power design.

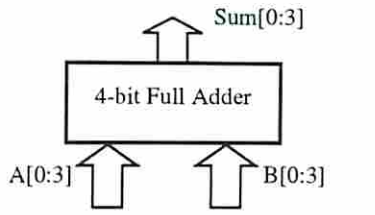
Let us consider a simple example to illustrate the input problem. Suppose that a 4-bit ripple-carry adder is fed successively by two input sequences S_1 and S_2, as it is shown in Fig.1(a). To estimate the total power consumption of the circuit (in a gate level implementation), we may sum over all the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

average switching activity per node (gate) is a key parameter that needs to be determined correctly.



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

(a)



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

(b)

Fig.1

The two sequences in Fig.1a, have the same signal probability on the input lines (p = 0.5), but are otherwise very different; whilst the average switching activity per bit is 0.5 for sequence S_1, it is 0.3 for sequence S_2. This difference in switching activity leads to: P_{S1} = 456 uW and P_{S2} = 365 uW 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 Fig.1b 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 uW, 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 [10], 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 [11] where authors use deterministic FSMs to model user-specified input sequences. Since the number of states in the FSM is equal to the length of the sequence to be modeled, the ability to characterize anything else but short input sequences is limited.

From a designer perspective, we may want to estimate the power consumption of the adder in Fig.1 in a context resembling as much as possible the one where this adder will be instantiated. For example, if the adder was designed to be an incrementor in the address calculation unit of a memory chip, then 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.

In summary, a number of issues appear to be important for power estimation and low-power synthesis. The *input statistics* which must be properly captured and the *length of the input sequences* which must be applied, are two such issues. From this perspective, the present paper shifts the focus from 'the circuit problem' to 'the input problem' and improves the state-of-the-art by proposing an original solution for *constrained sequence generation.*[1] Successful research on constrained sequence generation, will find application ground at least in three areas:

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

• *Sequence compaction:* This is basically the ability to construct a *representative sequence* (short enough to be efficiently simulated) equivalent as far as the total power consumption is concerned with the original sequence that reproduces the operating context in which the circuit is supposed to work. This

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

feature can make (circuit or gate-level) simulators a viable option for power analysis even for very large circuits and therefore deserves special attention; it is discussed and supported with examples in Sections 3 and 4.

• *Probability transformation:* This represents the ability to construct machines that convert a given set of input symbols, occurring with some fixed probabilities, into another one, which may have a completely different set of probabilities. Using the aforementioned formalism, a probability transformer when fed with input symbols $\{\alpha_1, \alpha_2,..., \alpha_n\}$ which occur with probabilities $\{p_1, p_2,...,p_n\}$, generates a new set of symbols $\{\beta_1, \beta_2,..., \beta_m\}$ with different, yet prescribed, probabilities $\{q_1, q_2,..., q_m\}$. For example, a probability transformer can take as input a highly correlated binary stream and produce an uncorrelated (random) output binary stream. Such a probability transformer, when placed in front of the circuit under consideration, can speed-up the probabilistic power estimation approaches as it eliminates the need for considering explicit input correlations which tend to slow down the estimation process. The input correlations will be instead captured by the structure of the probability transformer circuitry.

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 [12].

The mathematical foundation of our approach relies on the *stochastic sequential machines* (SSMs) theory and, without any loss in generality, emphasizes those aspects related to Moore-type machines. In this paper, we reveal a general procedure for SSM synthesis and describe a new framework for sequence characterization to match designer's needs for sequence generation or compaction. To address synthetically all three applications mentioned above, we focus on the basic task of synthesizing an SSM, able to generate constrained input sequences. Such a machine can be used not only in the stand-alone mode (as is the case for sequence compaction depicted in Fig.2a) but also in front of the target circuit (so as to

speedup the probabilistic power estimation as it is shown in Fig.2b).

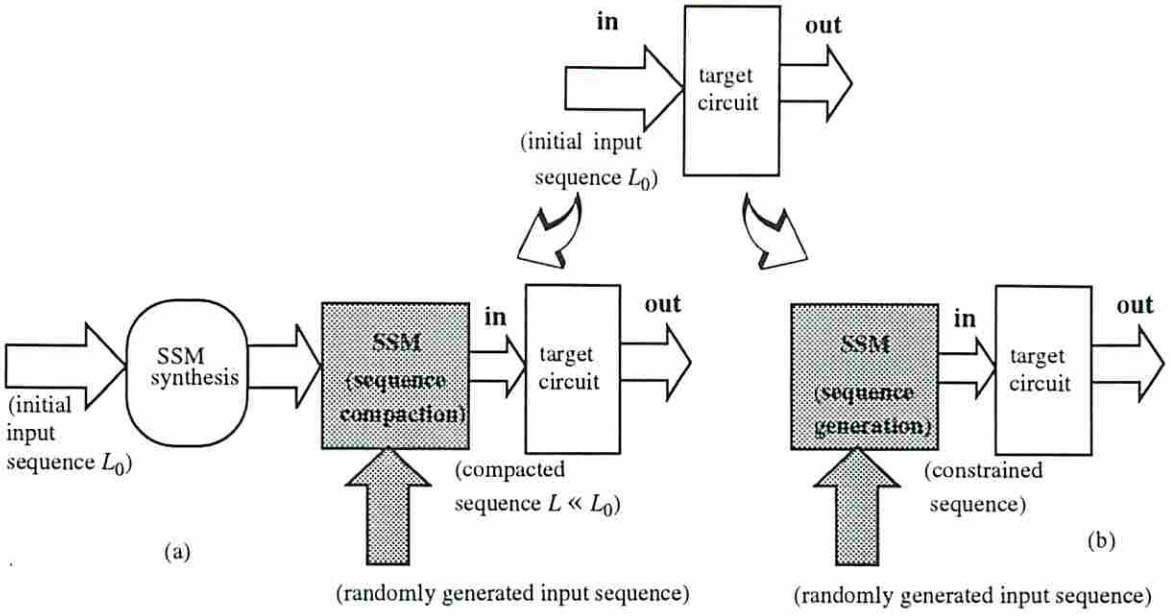


Fig.2

To conclude, both simulation-based approaches and probabilistic techniques for power estimation may benefit from this research. The issues brought into attention in this paper are new and represent a first step toward reducing the gap between the simulative and probabilistic techniques commonly used in power estimation. Finally, the concept of SSMs may find useful applications in other CAD-related problems.

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

2. Synthesis of SSMs

In this section, we review the concept of SSM and describe a basic procedure for synthesizing SSMs from their mathematical models. In what follows, we use the formalism and notations introduced in [19].

2.1. Stochastic machines: basic definitions

Definition 1: 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

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

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

Definition 2: Let $\mathcal{M}$ 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 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 (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 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 printing 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: A Moore-type SSM is a quintuple $\mathcal{M} = (S, X, Y, \{A(x)\}, \Lambda)$ where S, X, and Y are as in Definition 1, $\{A(x)\}$ is a finite set containing $|X|$ square stochastic matrices of order $|S|$ and Λ a deterministic function from S into Y.

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

Definition 4: Let $\mathcal{M}$ 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 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 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.

2.2. The synthesis procedure

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

Theorem 1 [13]: Any $m \times n$ stochastic matrix A can be expressed in the form $A = \sum p_i U_i$ where $p_i > 0$, $\sum p_i = 1$, and U_i are degenerate stochastic matrices (that is, matrix elements are 0 or 1 only), and the

number of matrices U_i in the expansion is *at most* $m(n\text{-}1) + 1$.

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 $[1/(1\text{-}p_1)]$ $[A\text{-}p_1U_1]$. ■

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

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

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

Let A be a stochastic matrix which can be expressed in the form $A = \sum p_i\ U_i\ (i = 1, 2,..., t)$ according to the above result. 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 1 (see Section 4).

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

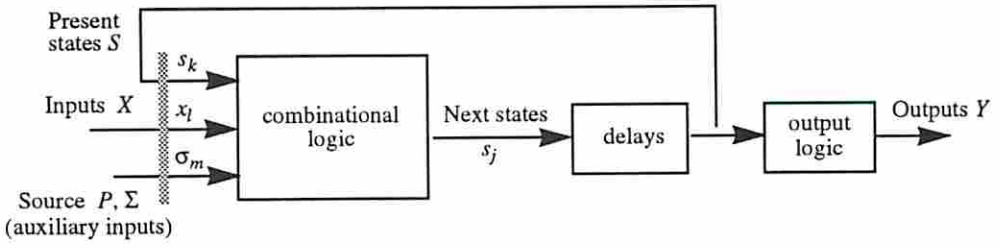


Fig.3

The combinational logic is constructed such that its output is s_j for input (x_l, σ_m, s_k) if and only if the

entry of matrix U_m in the row corresponding to (s_k, x_l) and the column corresponding to s_j equals 1. The output logic box is a combinational logic implementing the function Λ.

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

Example 3: Let's synthesize now the stochastic machine $\mathcal{M}$ defined in Example 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 1, we get $A = \frac{1}{2}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} + \frac{1}{4}\begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} + \frac{1}{6}\begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} + \frac{1}{12}\begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ 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 Fig.4.

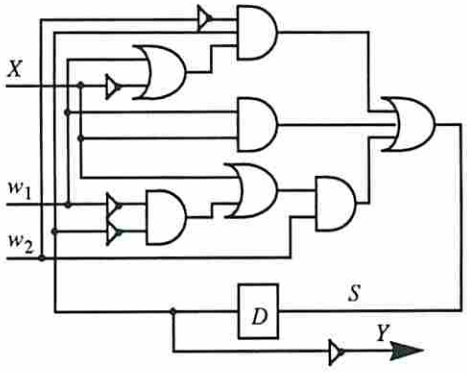


Fig.4

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

In summary, the basic synthesis procedure based on Theorem 1, involves essentially the synthesis of

a combinational circuit with feedback, and construction of information sources with prescribed probability distributions.

3. Constrained Sequence Characterization

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

3.1. Sequence equivalence

In what follows, we associate with every Moore SSM its output sequence (of length $L \geq 1$), generated during its normal operation, and we will interchangeably refer to both SSM and its output sequence. The general problem of equivalence between stochastic machines is very complex and for an in-depth introduction the reader is referred to [19]. For practical purposes, we restrict our attention only to reduced stochastic machines of Moore-type.

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

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

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

3) For every state s_i of $\mathcal{M}$ 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:

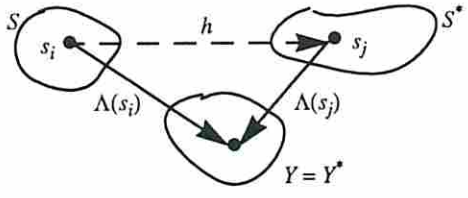


Fig.5

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

Definition 6: The output sequence Y generated by machine $\mathcal{M}$ 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 $\varepsilon = 0$, when $A = A^*$, which corresponds to exact equivalence). Differently stated, two output sequences are ε-equivalent if they are generated by SSMs characterized by nearly the same average transition probabilities, that is $\left|a_{ij}(x) - a_{ij}^*(x)\right| < \varepsilon$, 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^*.

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 [20] 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 [21]: Every stochastic matrix has 1 as a simple[2] 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 [21]. However, theorem 3 is very important for us because it makes possible to analyze perturbations on matrix A. To this effect, let's assume that the newly generated sequence is characterized by the matrix $A^* = [a_{ij}^*]$ where $a_{ij}^* = a_{ij} + \varepsilon_{ij}$ (ε_{ij} represents the error induced by pertutbations) 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,

---

1. This may appear, for instance, as a side effect if we apply Theorem 1 with limited precision.
2. This means that the multiplicity of the root $\lambda = 1$ in the equation $A^T \cdot p = \lambda \cdot p$ is one.

it is also a stochastic matrix and therefore, as stated in Theorem 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 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 $k_1\varepsilon + k_2\varepsilon^2 + \ldots$. ■

This theorem follows from the theory of algebraic functions developed in [22]. Since $\|a_{ij} - a_{ij}^*\| = 0(\varepsilon)$, it is easy to see that:

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

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

Proof: We have $P = \frac{V_{dd}^2}{2 \cdot T_{cycle}} \cdot \sum_{i,j,k} p_{i\to j} \cdot C_k \cdot n_{ij}^k$ where C_k is the output capacitance of gate k 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. Assuming that the input sequence is approximated by another input sequence such that the new set of transition probabilities satisfies $\|p_{i\to j} - p_{i\to j}^*\| = 0(\varepsilon)$, then the error made in the value of total power consumption is given by $|P^* - P| \le \frac{V_{dd}^2}{2 \cdot T_{cycle}} \cdot \sum_{i,j,k} |p_{i\to j}^* - p_{i\to j}| \cdot C_k \cdot n_{ij}^k = 0(\varepsilon)$. ■

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

Example 4: Let's consider the circuit in Fig.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 0.75).

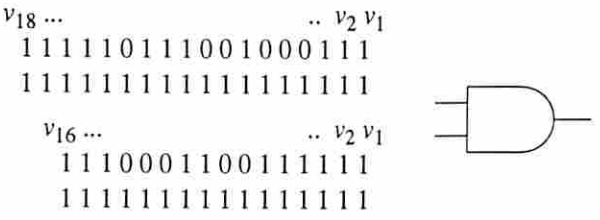


Fig.6

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 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 Fig.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 $|P - P^*| = \frac{C \cdot V_{dd}^2}{T_{cycle}} \cdot 0.08$.

4. Constrained Sequence Compaction

In practice, we may want to generate a fixed-length sequence satisfying a certain set of constraints or, more frequently, we may have from simulation a characteristic sequence for a target circuit and want to compact it into a new one by preserving its statistics. The first situation was considered in Section 2.2 for the synthesis of the stochastic machine $\mathcal{M}$. 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.

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 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:

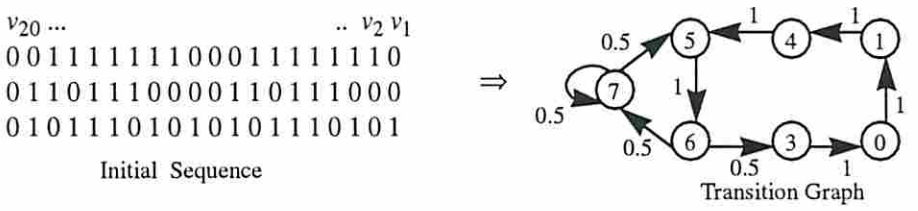


Fig.7

In the right side of Fig.7, we have the transition graph corresponding to this sequence. The 'state' nodes are labelled with the values that appear in the initial sequence (decimally encoded and read from top to bottom), while the labels on the edges are conditional probabilities captured by analyzing 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 5, we may deduce that $|S^*| = 7$ and $Y^* = Y = S = S^*$. The corresponding stochastic matrix for the initial sequence is shown below,

along with its decomposition from Theorem 1:

$$A = \begin{bmatrix} 0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0.5&0&0&0&0.5 \\ 0&0&0&0&0.5&0&0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} 0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0 \end{bmatrix} + 0.5 \cdot \begin{bmatrix} 0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1 \end{bmatrix} = 0.5 \cdot U_1 + 0.5 \cdot U_2 .$$

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

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 Fig.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).

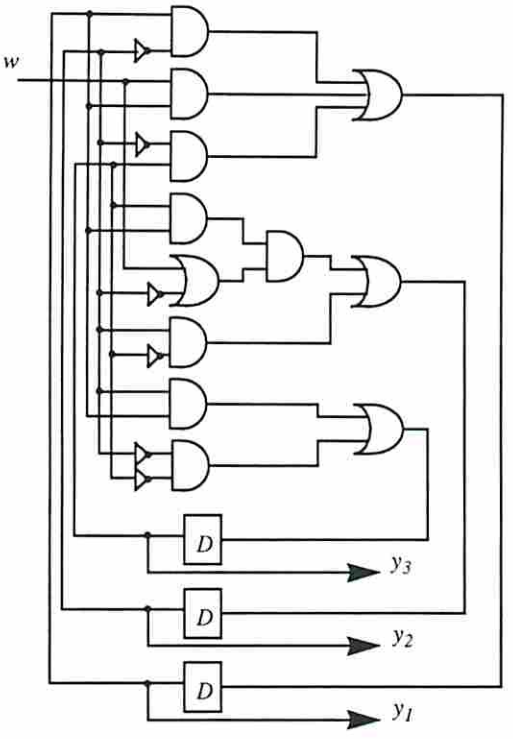


Fig.8

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 6 with $\varepsilon = 0$, we have $\|A - A^{*}\| = 0$, therefore, in this case, a compaction ratio of 2 has been achieved without any loss of information.

We should note that using another initial vector (e.g. '110' or '101') as initial state of the circuit in Fig.8, we would have obtained other output sequences, but all of them still satisfying the inequality from Definition 6.

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

- first, it allows the hardware synthesis of the machine $\mathcal{M}$;

- 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 1 and Theorem 2 */
  end while;
  /* let t be the number matrices U_i in the decomposition */
  {σ_i} = Encode_Symbols (t);
  /* construct the transition table for the SSM and synthesize the circuit */
  Table = Construct_Table ({U_i});
  Circuit = Construct_Circuit (Table);
  /* generate the new sequence providing corresponding values for the auxiliary inputs w_i */
  Gen_Sequence (Circuit);
end Generate_Sequence;
```

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:

$cost(x, y) = \sum_{i, j, k, l = 0, 1} |p(x_{i \to j} y_{k \to l}) - p(x_{i \to j}) \cdot p(y_{k \to l})|$ 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 ($\alpha = 0$). The above formulation involving the cost function is in fact a *min-cut partitioning* problem which is NP-complete in the general case [15]. Fortunately, excellent heuristics are available to solve this problem [16][17].

Example 6: Let us consider the input sequence in Fig.7, and let x, y, z be the 3 bits for representing v_1, v_2, ...,v_{20} that feed the simple circuit in Fig.9a; 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* (Fig.9c) assigning to each bit a vertex and weighting the edges with the cost function (α coefficients are $\alpha(x,y) = \alpha(y,z) = 1/2$, $\alpha(x,z) = 1/4$).

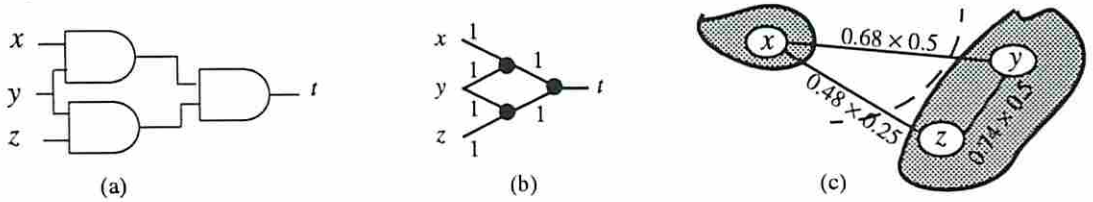


Fig.9

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

5. Practical Considerations and Experimental Results

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

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

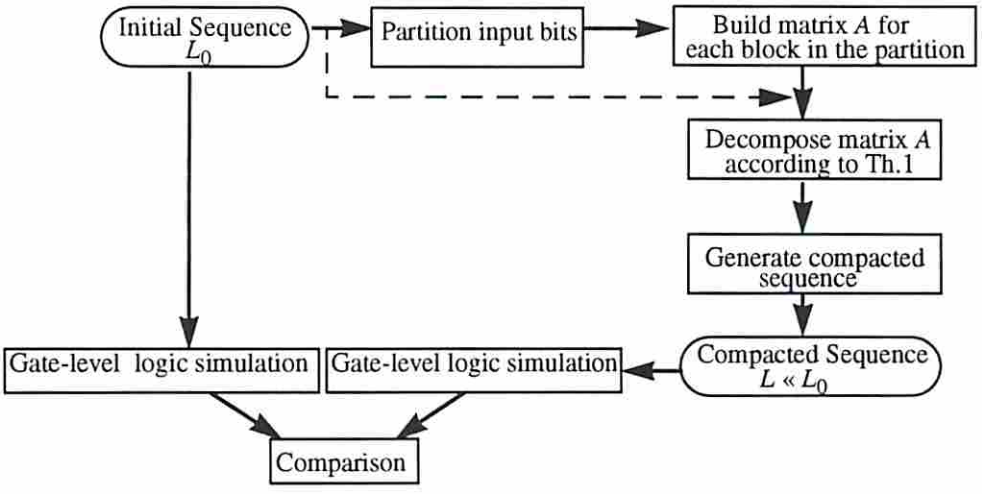


Fig.10

Our strategy is depicted in Fig.10 and follows the steps of the algorithm in Section 4. Basically, we verified our ability to generate and compact constrained input sequences which may be also used as power benchmarks in the design process. In all experiments, we target *lossy compression* [18], 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. To each block, we then associate a stochastic machine (SSM_1, SSM_2,...,SSM_b in Fig.11b). 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.

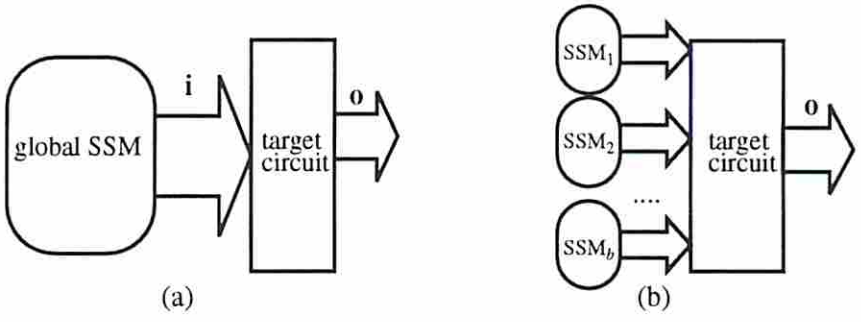


Fig.11

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

Once a partition is obtained, we simply apply the algorithm in Section 4 to each group of bits, that is, we build the matrix A (by preserving exactly the transition probabilities) and after that, decompose it into a set of degenerate matrices as stated in Theorem 1. A deterministic sequential machine is then constructed for each degenerate matrix in this decomposition. The next step does the actual generation of the output sequence (of length L): the resulting auxiliary inputs are excited by a random number generator satisfying the probability distribution from the decomposition process. From our experience, this strategy works well (less than 5% relative error on average) for pseudorandom and moderately biased input sequences. If the sequence to be compressed is a highly correlated one, then this approach will result in an error level of about 5-10% on average. In such cases, the global modeling of the SSM, if possible, (as depicted in Fig.11a) 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 Tables 1-2, 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 Fig.11b; this type of

sequence was obtained by randomly applying a set of logic operators (AND, OR, XOR) between the bits of a random sequence (thus increasing the correlation level among 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 C432 has 36 inputs which means a number of 9 blocks for $k = 4$ (and accordingly 9 stochastic machines SSM_1, SSM_2,..., SSM_9 in Fig.11b), and 6 blocks for $k = 6$. For two of the sequential circuits (i.e., s298 and s386) the partitioning step was unnecessary due to the small number of input bits (3 and 7, respectively).

Table 1: Total power (uW@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	

Table 2: Total power (uW@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	

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 1, instead of simulating 100,000 vectors with an exact power of 4875.60 uW, one can use only 1000 vectors with an estimate of 4861.80 uW ($k = 6$) or just 100 vectors with a power consumption estimated as 4974.00 uW ($k = 4$).

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

On the efficiency side, in Table 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 2) if this is satisfactory from a practical point of view.

Table 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 Tables 4-5, 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 Fig.11a 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 5.

Table 4: Total power (uW@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 5: Total power (uW@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
	Average. Error (%)	2.50	2.99	3.94

As reported in Tables 4-5, 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 [23]. 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 6 we present the results obtained for sequences of type 3 and compaction ratio $r = 200$.

Table 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 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.

5. Conclusion

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

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

References

[1] F. N. Najm, 'A Survey of Power Estimation Techniques in VLSI circuits', *IEEE Transactions on VLSI Systems,* vol.2, no.4, pp. 446-455, Dec.1994.

[2] A. Ghosh, S. Devadas, K. Keutzer, and J. White, 'Estimation of Average Switching Activity in Combinational and Sequential Circuits', in *Proc. ACM/IEEE Design Automation Conference*, pp. 270-275, June 1992.

[3] 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.

[4] 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.

[5] 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.

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

[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] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

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

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

[16] 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.

[17] 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.

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

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

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

[21] A. Katok and B. Hasselblatt, 'Introduction to the Modern Theory of Dynamical Systems', Cambridge University Press, 1995.

[22] J.H.Wilkinson, 'The Algebraic Eigenvalue Problem', Clarendon Press, 1988.

[23] 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.

Vector Compaction Using Dynamic Markov Models

Radu Marculescu, Diana Marculescu,
and Massoud Pedram

CENG 96-14

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

February 1996

Vector Compaction Using Dynamic Markov Models

Abstract

Evaluation of power dissipation is a critical step in the design of today's ICs. Power disssipation is strongly input pattern dependent and hence, to obtain accurate power values, one must simulate the circuit with a large number of input vectors that typify the application data. The goal of this paper is to present an effective and robust technique for compacting a large sequence of input vectors into a much smaller input sequence so as to reduce the circuit/ gate level simulation time by orders of magnitude and mantain the accuracy of the power estimates. In particular, this paper introduces and characterizes a family of dynamic Markov trees that can model complex spatiotemporal correlations which occur during power estimation both in combinational and sequential circuits. This new framework is very effective and flexible: the Markov model itself is derived through a one-pass traversal of the initial sequence and it can be used after that with any available simulator to derive power consumption. As the results demonstrate, large compaction ratios of 1-2 orders of magnitude can be obtained without significant loss (less than 3% on average) in the accuracy of power estimates.

1. Introduction

CAD tools have played a significant role in the efficient design of the high-performance digital systems. In the past, time and area were the primary concerns of the CAD community during the optimization phase. More recently, circuit testability was added as yet another important consideration during the design process. With the growing need for low-power electronic circuits and systems, power analysis and low-power synthesis have become crucial tasks that must also be addressed. 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.

Power estimation is in general a difficult problem; the key task in this process is the accurate and fast estimation of average switching activity. To date, both simulative [1]-[4] and nonsimulative approaches [5]-[10] have been tried, each one having its own advantages and limitations [11]. More specifically, general simulation techniques provide sufficient accuracy, but at high computational cost; it is simply expensive to simulate thousands of vectors. On the other hand, nonsimulative approaches (best represented by probabilistic power estimation techniques) are in general faster, but less accurate than those based simulation; usually, the input correlations and the reconvergent fan-out in the target circuit make things very complicated and simplifying assumptions (like input independence) become mandatory. During the last years, the gap between simulative and nonsimulative approaches remained basically the same, despite remarcable advances made in both directions.

As a conclusion, a number of issues appear to be important for power estimation and low-power synthesis. The *input statistics* which must be properly captured and the *length of the input sequences* which must be applied are two such issues. 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 [12], are of little or no help

here. The reason is that more elaborate set of input statistics must be preserved or reproduced during sequence generation for use by power simulators. One such attempt is [13] 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 severely limited. A more elaborate and effective technique was presented in [14] where, based on stochastic sequential machines, the authors succeed in compacting large sequences without significant loss in accuracy. However, in the present research, the limitations of that approach are pointed out and overcome by the proposed technique.

The present paper improves the-state-of the art by providing an original solution for vector compaction problem which potentially reduces the gap between simulative and nonsimulative approaches. Traditionally, data compression techniques were aimed to lossless compression, that is the ability to encode a body of data, transmit it eventually over a communication line and finally, decode it uniquely, without any loss of information during this process. Our objective in this paper is slightly different: having an initial sequence (assumed representative for some target circuit), we target *lossy compression* [15], that is the process of transforming an input sequence into a smaller one, such that the new body of data represents a *good approximation* as far as total power consumption is concerned. We use throughout the paper the term 'compaction' instead of 'compression', because we think it better suits our intentions.

The foundation of our approach is probabilistic in nature; it relies on *adaptive* (*dynamic*) *modeling* of binary input streams as first-order Markov sources of information and is applicable both to combinational and sequential circuits. The adaptive modeling technique itself (best known as Dynamic Markov Chain or DMC modeling) was introduced very recently in the literature on data compression [16] as a candidate to solve various compression problems. From the very begining, this technique looked very promising and indeed, in most practical situations, has been more effective than any other compression technique available to date. However, the original model introduced in [17] is not completely satisfactory for our purpose. In this paper, we thus extend the initial formulation to manage not only correlations among adjacent bits that belong to the same input vector, but also correlations between successive input patterns.

As demonstrated and supported by practical evidence, this new framework is extremely effective in power estimation. The basic idea is illustrated in Fig.1. To evaluate the total power consumption of a target circuit for a given input sequence L_0 (Fig.1a), we derive first the Markov model of the input sequence through a one-pass traversal technique and after that, having this compact representation, we generate a much shorter sequence L, equivalent with L_0, which can be used with any available simulator to derive accurate power estimates (Fig.1b).

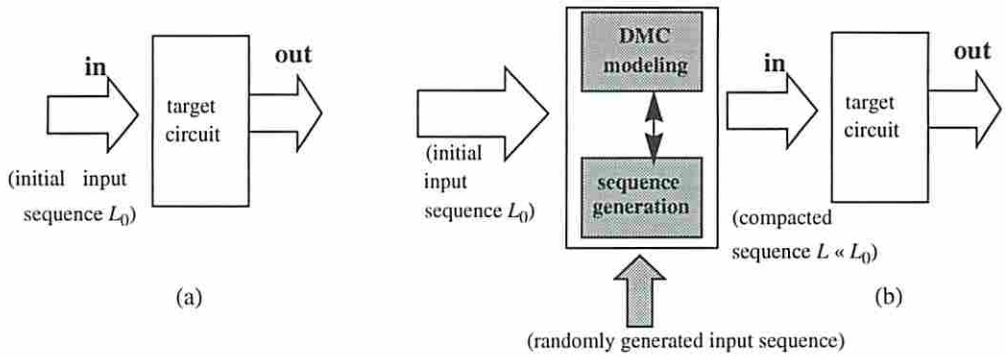


Fig.1

We point out here that the present approach can be used without any difficulty to generate benchmark data for power estimation, that is, input sequences with different lengths that satisfy a set of user-prescribed characteristics in terms of word-level transitions or conditional probabilities.

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

The paper is organized as follows: Section 2 reviews the basic concepts of DMC modeling technique. Section 3 formalizes the power-oriented vector compaction problem and discusses parameters which makes this approach effective in practice. Section 4 presents a DMC-based procedure for vector compaction. In sections 5 and 6, we give some practical considerations and experimental results, respectively. Finally, we conclude by summarizing our main contribution.

2. Background on Dynamic Markov models

Modeling for data compaction in digital systems involves essentially the derivation of certain source-string events and their contexts (the set of bits or words surrounding some bit or word under consideration), which uniquely describe the original source string. We consider therefore the *model* as having two parts: 1) the *structure* which is the set of events and their contexts and 2) the *parameters* which are probabilities assigned to the events. The structure is intended to capture the characteristics of the entire set of sequences under consideration while the parameters are tailored to each individual sequence.

Without loss of generality, in what follows we restrict ourselves to finite binary strings, that is, finite sequences consisting only of 0's and 1's. The set of events of interest is the set S of all finite binary sequences on k bits. A particular sequence S_1 in S consists of vectors v_1, v_2,..., v_n (which may be distinct or not), each having a positive occurence probability[1]. Indices 1, 2,..., n represent the discrete time steps when a particular vector is applied to a

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

target circuit. Imposing a total ordering among bits, such a sequence may be conveniently viewed as a binary tree (for reasons that will become obvious later on, let's call it DMT_0 from *Dynamic Markov Tree of order zero*) where nodes at level j correspond to bit j ($1 \leq j \leq k$) in the original sequence; each edge that emerges from a node is labelled with a positive count (and therefore with a positive probability) that indicates how many times the substring from the root to that particular node, occured in the original sequence. For clarity, let's consider the following example.

Example 1: For the following 4-bit sequence consisting of 8 non-distinct vectors: (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8) = (0000, 0001, 1001, 1100, 1001, 1100, 1001, 1100) the construction of the tree DMT_0 is shown step-by-step in Fig.2a. Obviously, the whole Markov tree that models this sequence must have four levels because the original sequence is a 4-bit sequence. Without any loss in generality, we assume a left-to-right order among bits that is, the leftmost bit in any vector v_1 to v_8 is considered as being bit number one (and consequently represented at level one in DMT_0 as shown in Fig.2a), the next bit is considered as being bit number two and so on. Every time when a vector is completely scanned (that corresponds to reaching the level four in the tree), we come back to the root and start again with the next vector in the sequence. While the input sequence is scanned, the actual counts on the edges are dynamically updated (as shown in Fig.2a for the first three vectors) such that, for this particular example, they finally become as indicated in Fig.2b. The Markov tree in Fig.2b contains in a compact form all the spatial information about the original sequence v_1, v_2,..., v_8. We point out that this sparse structure is possible only by using the *dynamic (adaptive)* fashion of growing the tree DMT_0 just illustrated. Another approach would have been to consider a static binary tree capable to model any 4-bit sequence and just to update the counts on the edges while scanning the original sequence (Fig.3). By doing so, we would end up with the obvious disadvantage of having 15 instead of 9 nodes in the structure for the same amount of information; this reason alone is sufficient for considering from now on only dynamically grown models.

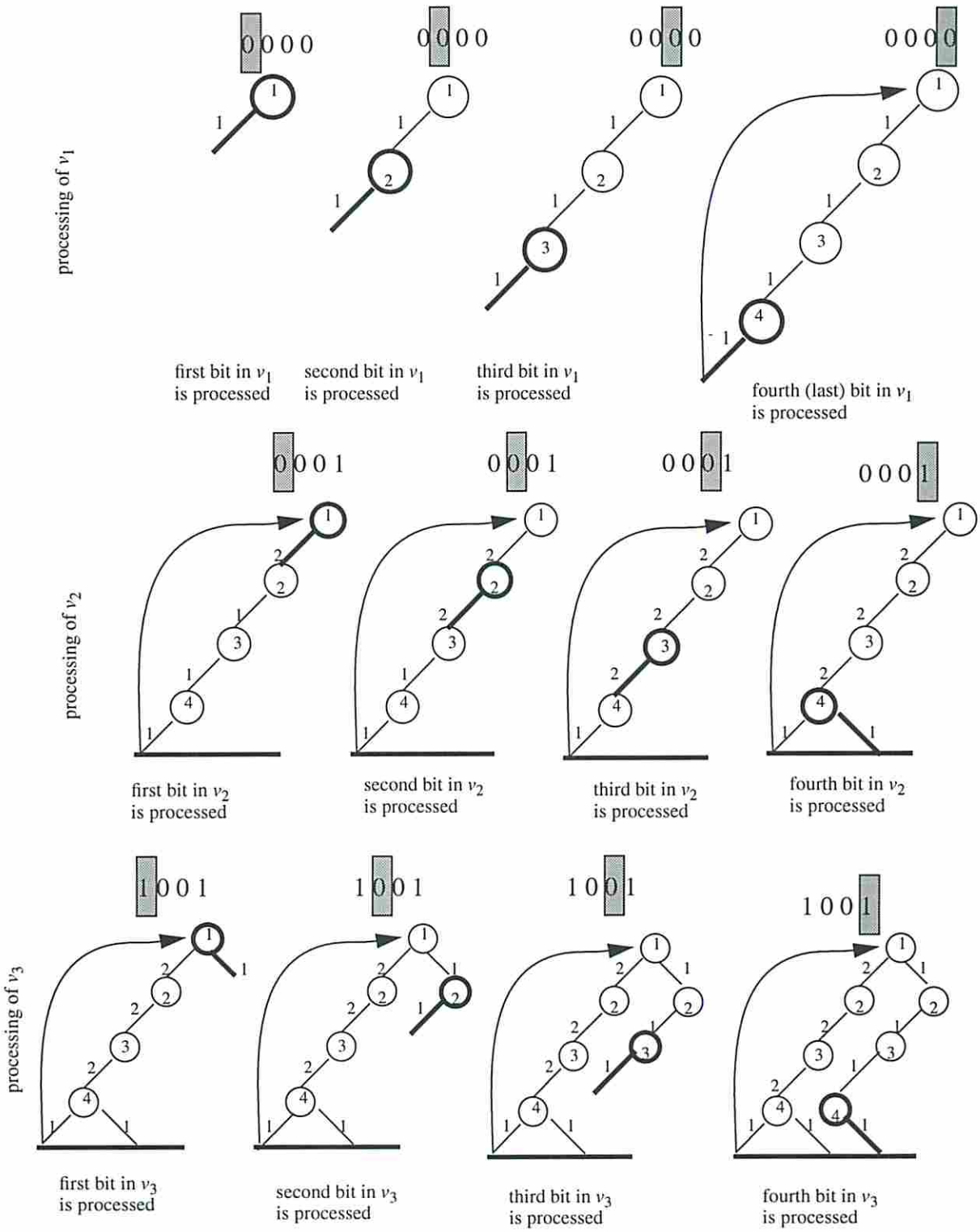

processing of v_1
0 0 0 0
first bit in v_1 is processed
second bit in v_1 is processed
third bit in v_1 is processed
fourth (last) bit in v_1 is processed
processing of v_2
0 0 0 1
first bit in v_2 is processed
second bit in v_2 is processed
third bit in v_2 is processed
fourth bit in v_2 is processed
processing of v_3
1 0 0 1
first bit in v_3 is processed
second bit in v_3 is processed
third bit in v_3 is processed
fourth bit in v_3 is processed

Fig.2 (a)

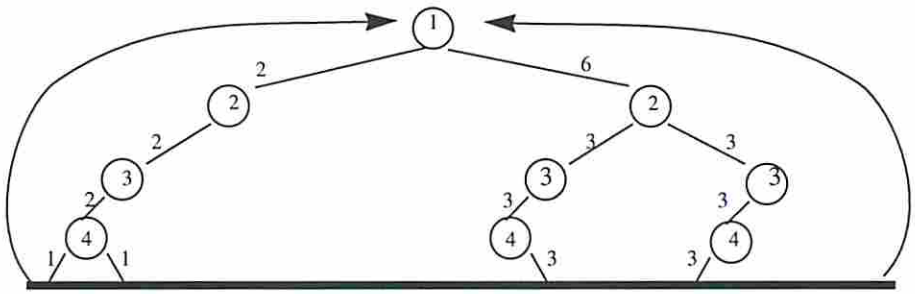


Fig.2(b)

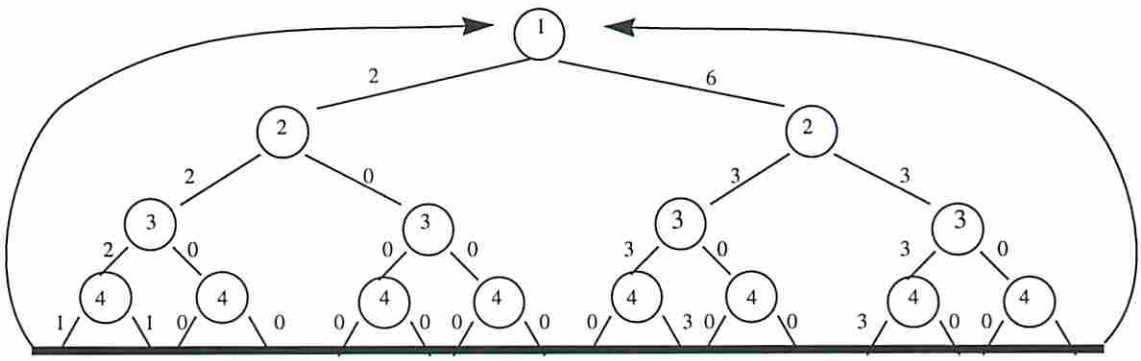


Fig.3

Definition 1. We define the *information source*, to be the pair <*S,P*>, where *P* is a function from *S* into [0,1] satisfying the condition:

$$P(v) = \sum_{x \in S} P(vx) \qquad (1)$$

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

The above condition, simply states that the sum of the counts attached to the immediate successors of node v equals its own value $P(v)$. As we can easily see in Fig.2, condition (1) is satisfied at every node in this representation[1]. In addition, based on the counts of the terminal edges, we may easily compute the probability of occurence for a particular vector in the sequence. For instance, the probability of occurence for string '1001' is 3/8 (because the count on the terminal edge that corresponds to '1001' is 3 and the length of the sequence is 8) while the probability of string '1111' is zero, '1111' being a 'forbidden' vector for this particular sequence.

Traditionally, data compression algorithms (e.g. Huffmann, Ziv-Lempel, Cleary-Witten) [16] have been targeted to work at byte- or word-level. In contrast, DMC is normally used to process one bit of the input at a time rather than one symbol at a time. In principle, there is no reason why a symbol-oriented version of the method can not be used. However, for practical reasons (less storage required and more efficiency in computations), the bit version of DMC is preferred.

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

3. Power-oriented data compaction

In this section, first we review the main concerns about preserving the input statistics to obtain accurate power estimates. Second, we present a new approach based on DMC by generalizing the model introduced in Section 2.

3.1 Problem formulation

Input pattern dependence has a dramatic impact on power dissipation estimates. If one ignores the input statistics (which give the actual correlations among the primary inputs), power estimation results can be seriously impaired. Capturing only signal probabilities at the primary inputs of the circuit is not enough for accurate estimates therefore, for power estimation purposes, it is critical to distinguish between sequences which exhibit the same signal probabilities on different bit lines, yet showing very different spatial and temporal correlations.

Assuming that a gate level implementation is available, to estimate the total power dissipation, one can sum over all the gates in the circuit the average power dissipation due to the capacitive switching currents, that is: $P_{avg} = \frac{f_{clk}}{2} \cdot V_{DD}^2 \cdot \sum_n (C_n \cdot sw_n)$ where f_{clk} is the clock frequency, V_{DD} is the supply voltage, C_n and sw_n are the capacitance and the average switching activity of gate n, respectively. From here, the average switching activity per node (gate) is the key parameter that needs to be correctly determined, mostly if we are interested in a node-by-node basis power estimation. However, this parameter is highly sensitive to the input statistics, namely it depends significantly on transition and conditional probabilities among different signal lines.

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

$$\left| P(v^- = X \wedge v^+ = Y) - P(u^- = X \wedge u^+ = Y) \right| < \varepsilon \qquad (2)$$

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

3.2 A DMC-based approach

An attempt to solve the vector compaction problem for power estimation was recently presented in [14]. In that paper, the authors use elements from probabilistic automata theory to synthesize stochastic machines which can be used in a standalone mode for sequence compaction. From a practical point of view, however, this approach has two inherent limitations:

- 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 and one should therefore allow limited precision in the calculations to simplify the decomposition process.

• The compaction technique on stochastic machines is a multiple-step compaction technique. An initial pass through the sequence is performed to extract the statistics of interest; after that, the stochastic machine is synthesized and then the new sequence is generated. This is especially disavantageous for large sequences when the on-line computer memory and time requirements become prohibitive.

The disavantages mentioned above can be eliminated by using DMC modeling. To this end, in what follows we introduce an original framework for power-oriented data compaction.

From Section 3.1, it follows that the spatiotemporal correlations that characterize a particular sequence are the key factor in power estimation. Differently stated, not only a particular vector v_i in a given sequence is important, but also its relative position in that sequence matters. More precisely, different interleavings among the vectors belonging to the same initial set (v_1, v_2,..., v_n) (e.g.(v_1,...,v_i, v_j, v_k,...,v_n), (v_1, ...,v_i, v_k, v_j, ...,v_n) or (v_1, ...,v_k, v_i, v_j, ...,v_n)) define completely different input sequences. Coming back to the model presented in Section 2, we observe that DMT_0 alone cannot capture this property; we say that DMT_0 has *no memory* and therefore the relative order of vectors in the initial sequence is irrelevant for DMT_0's construction. Obviously, DMT_0 is a poor structure for S because it does not preserve properly the order of events. In Fig.2b for instance, the value of 3/8 is the probability to see the particular string (state) '1001' in the original sequence but this gives us no indication at all about the sequencing of this vector relative to another one, say '0001'.

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

Example 2: For the same sequence in Example 1, suppose we want to construct its corresponding tree DMT_1. We begin as in DMT_0 and for each leaf that represents a valid combination in the original sequence, we construct a new tree (having the same depth as DMT_0) which is meant to preserve the *context* in which the next combination occurs. For instance, the vector v_2 = 0001 follows immediately after v_1 = 0000; consequently when we reach the node that corresponds to v_1 (the leftmost path in Fig.4a), instead of going back to the root (and therefore 'forgetting' the context), we start to build a new tree (rooted at the current leave of DMT_0) as indicated in Fig.4a. Basically, we added a new path which corresponds to '0001'. The newly constructed tree will preserve the context in which v_2 = 0001 occurred that is, immediately after v_1 = 0000 (denoted by $v_1 \rightarrow v_2$). After processing the pair (v_1,v_2), we come back to the root and continue with (v_2,v_3) as shown in Fig.4b; v_2 alone leads us to the second leftmost edge of DMT_0 from where, to construct DMT_1, we have to add the path '1001' which corresponds to v_3. In this way, we indicate the sequenting between v_2 and v_3 that is, $v_2 \rightarrow v_3$.

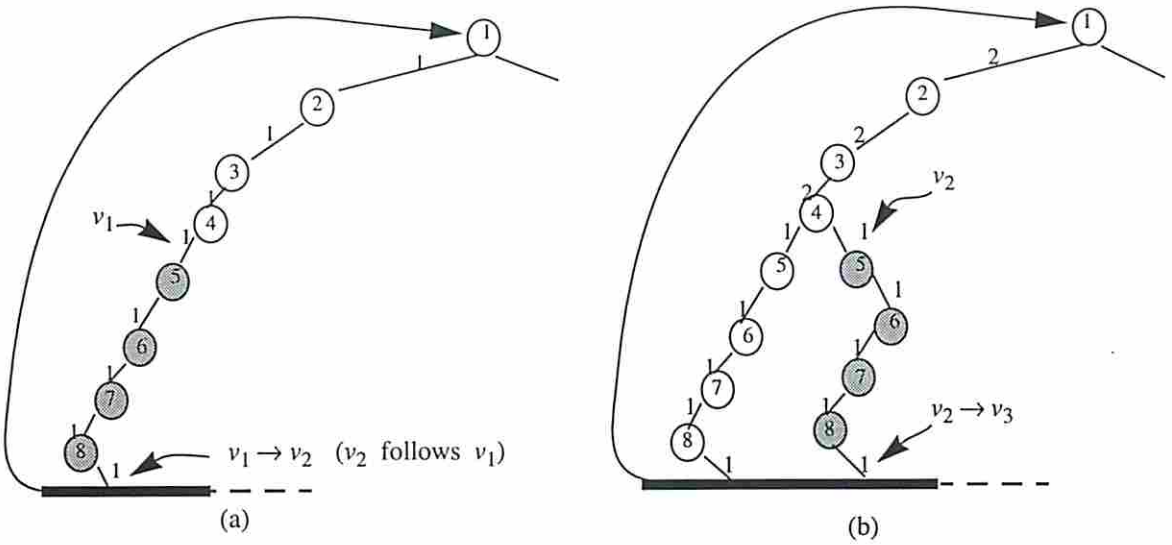


Fig.4

What is important to note here, is that *all* vectors in the original sequence are processed that is, *none of them is skipped* during the construction of DMT_1. This is the theoretical basis for accurate modeling of the input sequences as first-order Markov sources of information. Similarly, continuing this process for all leaves in DMT_0 in Fig.2b, we end up by building the whole tree DMT_1 as shown in Fig.5.

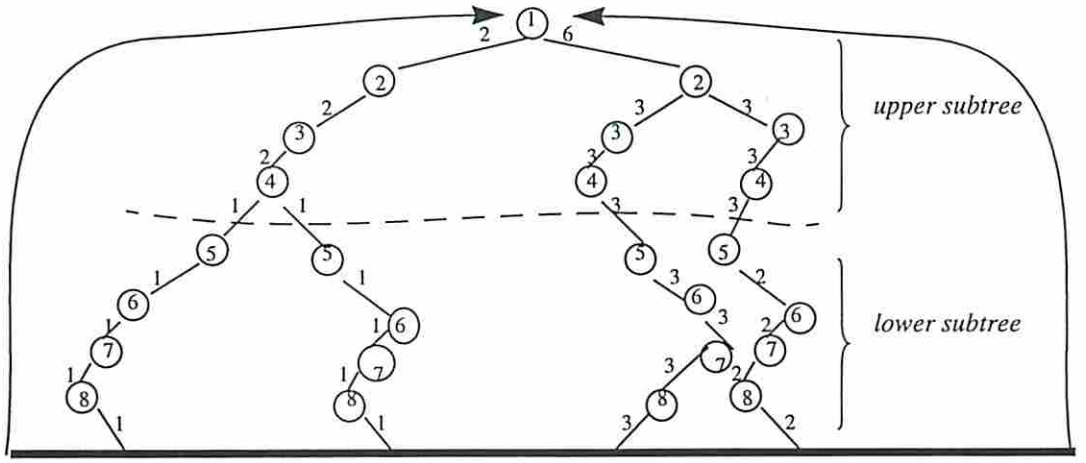


Fig.5

In Fig.5, we separated by a dashed line the two subtrees that constitute DMT_1. The upper subtree (levels 1 to 4) represents DMT_0, that is, it sets up the state probabilities for the sequence; the lower subtrees (levels 5 to 8), give the actual sequencing between any two successive vectors. To keep the counts in these subtrees consistent, while we traverse the lower subtrees and update the counts on their edges, we also accordingly increment the counts on the paths in the upper subtree (in fact, all vectors except the first and the last are processed exactly twice, once in the upper DMT_0 and next in the lower DMT_0). In practice the counts of these two subtrees may differ by one, due to the finite length of the sequences. This aspect can be seen in Fig.5 at level 5 on the rightmost path at the border between

the upper and lower subtree. A practical solution to this issue is to consider the input sequence as being cyclic that is, to link the last combination in the sequence with the very first one or simply add two dummy states, one before and one after the initial sequence.

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

Proposition 1 [19]. We write the probability of a vector string $v = v_1 v_2 ... v_n$ as follows:

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

where the conditional probabilities are uniquely defined by: $P(x|v) = P(vx) / P(v)$.

This property, used in connection with the counts on the edges, allows a quick calculation of the transitions probabilities that characterize a particular sequence. For example, if we want to calculate the transition probability '1001' $\rightarrow$ '1100' we have from Proposition 1 $P(v) = P(v_1 v_2) = P(v_1) \cdot P(v_2 | v_1) = 3/8$ which is exactly the count on the path '10011100' in the tree DMT_1 divided by the sequence length.

Theorem 2. Any sequence in S can be modeled as a first-order Markov source using the structure DMT_1 and parameters P. We call this process Dynamic Markov Chain (DMC) modeling.

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

Theorem 3. The structure DMT_1 and parameters P are equivalent to a stochastic sequential machine.

Sketch of proof: DMT_1 defines a Markov source (based on Theorem 2). Any Markov source is characterized by a stochastic matrix A. According to the decomposition Theorem 1 given in [14], this matrix is uniquely associated to a stochastic machine (a finite-state machine with randomly generated inputs). Thus, DMT_1 is equivalent to a SSM. ■

Generally speaking, the theory of stochastic sequential machines is far more developed than the theory of DMC modeling. However, the DMC modeling technique based on DMT_1 seems to be more effective as it offers a much more compact structure and generally outperforms the compaction techniques based on stochastic machines. Specifically:

- Sequence compaction based on DMT_1 avoids the time consuming decomposition process necessary in stochastic machines' synthesis.
- Using DMT_1 one can avoid the need for partitioning the input vectors into groups of bits. Therefore, one can expect to improve the accuracy, especially in those cases when the input patterns are highly correlated.
- DMT_1 is constructed dynamically (new nodes are added only 'on demand') therefore it offers a much more compact data structure than matrix A does.

The DMC modeling technique can be successfully used to model such complex spatiotemporal correlations. The structure DMT_1 just introduced is general enough to capture completely the correlations among all bits of the same

input vector and also between successive input patterns. Indeed, the recursive construction of DMT_1 by considering successive bits in the upper and lower subtrees completely captures the word-level (spatial) correlations for each individual input vector in the original sequence. Furthermore, cascading lower subtrees for each path in the upper subtree, gives the actual sequencing (temporal correlation) between successive input patterns. This model captures completely spatial correlations and first-order temporal correlations. However, it has conceptually no inherent limitation to be further extended to capture temporal dependencies of higher orders. For instance, if we continue to define recursively DMT_2 (as a function of DMT_1), we can basically capture second-order temporal correlations (Fig.6).

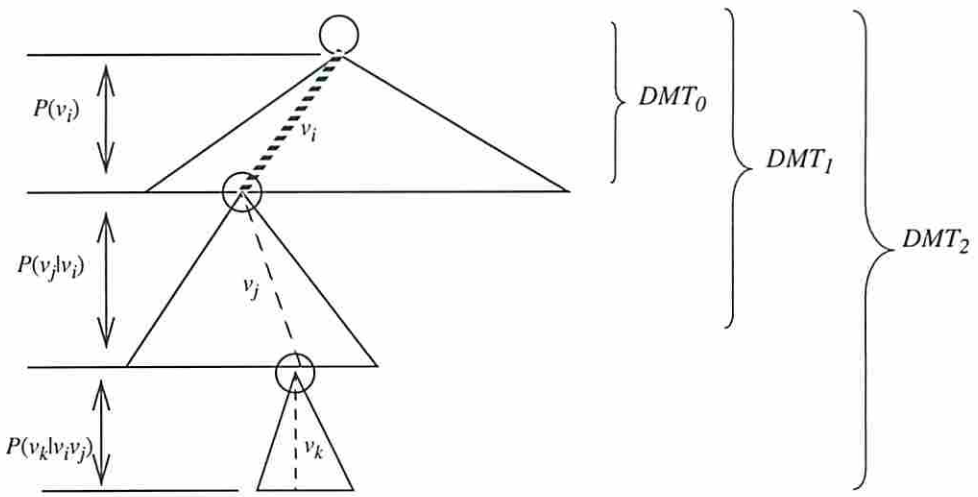


Fig.6

Theorem 4. The general structure DMT_n and parameters P can model spatiotemporal correlations of order n.

Sketch of proof: Let $v = v_1\ v_2\ ...\ v_n$ be a string in DMT_n (the substring v_i belongs to the i-level tree). Using Proposition 1, we have $P\ (v_n \mid v_1\ v_2\ ...\ v_{n-1}) = P\ (v_1\ v_2\ ...\ v_n)\ /\ P\ (v_1\ v_2\ ...\ v_{n-1})$ and thus the lag-n Markov chain characterizing the input can be fully modeled by the DMT_n structure. ■

4. A DMC-based vector compaction procedure

A practical procedure to construct DMT_1 and generate the compacted sequence is given in Fig.7a. During a one-pass traversal of the original sequence (when we extract the bit-level statistics of each individual vector $v_1,v_2...,v_n$ and also those statistics that correspond to pairs of consecutive vectors (v_1v_2), $(v_2v_3),...,(v_{n-2}v_{n-1}),(v_{n-1}v_n)$) we grow simultaneously the tree DMT_1. We continue to grow DMT_1 as long as the Markov model is smaller than a user-specified threshold (*model_size*), otherwise we just generate the new sequence up to that point and discard (flush) the model. A new Markov model is started again and the process is continued up to the end of the original sequence. The *generate_seq* procedure called by the DMC program is detailed in Fig.7b. Each generation phase is driven by the user-specified compaction parameter *ratio* that is, in order to generate a total of $m = n/ratio$ vectors, we have to keep

the same compaction ratio for every dynamically grown Markov model.

```
procedure DMC (input_file, ratio, model_size) {
 initial_state = new_state ();
 symbol = read_input (input_file);
 update_tree (symbol, upper_tree, initial_state);
 crt_state = last_state (symbol, upper_tree);
 while (!EOF (input_file)) {
  symbol = read_input (input_file);
  if (number_of_states < model_size) {
   update_tree (symbol, lower_trees, crt_state);
   update_tree (symbol, upper_tree, initial_state);
   crt_state = last_state (symbol, upper_tree);
  }
  else {
   generate_seq (upper_tree, lower_trees, ratio);
   flush_model (upper_tree, lower_trees);
   initial_state = new_state ();
   symbol = read_input (input_file);
   update_tree (symbol, upper_tree, initial_state);
   crt_state = last_state (symbol, upper_tree);
  }
 }
 generate_seq (upper_tree, lower_trees, ratio);
}
```

(a)

```
procedure generate_seq (upper_tree, lower_trees, ratio) {
 crt_symbol = generate_random ();
 lower_tree_node = last_node (upper_tree, crt_symbol);
 upper_tree_node = root (upper_tree);
 do {
  for each bit in the current vector {
   generate '0' or '1' to maximize the decrease in
   absolute error;
   if ('0' is generated) {
    lower_tree_node = left (lower_tree_node);
    upper_tree_node = left (upper_tree_node);
   }
   else {
    lower_tree_node = right (lower_tree_node);
    upper_tree_node = right (upper_tree_node);
   }
  }
  lower_tree_node = upper_tree_node;
  upper_tree_node = root (upper_tree);
 }
 while there are still vectors to be generated;
}
```

(b)

Fig.7

In all our experiments we used the DMC modeling technique based on the structure DMT_1. We also note that this strategy does note allow 'forbidden' vectors that is, those combinations that did not occur in the original sequence, will not appear in the final compacted sequence either. This is an essential capability needed to avoid 'hang-up' ('forbidden') states of the circuit during simulation process for power estimation.

Example 3: Assume that we are given the following 3-bit sequence consisting of 17 non-distinct vectors: (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}) = (001, 100, 001, 110, 111, 111, 101, 110, 011, 000, 101, 001, 100, 000, 110, 110, 011); our objective is to compact this sequence with a compaction ratio of 2.

We start building the Markov model that characterizes the initial sequence. For clarity, the construction of the tree DMT_1 is shown in Figs.8-9 for two different scenarios. First, in Fig.8, we assume that the parameter *model_size* is set by the user to the value 35; this means that the model can be grown dynamically (without any need for flushing) until this limit is reached.

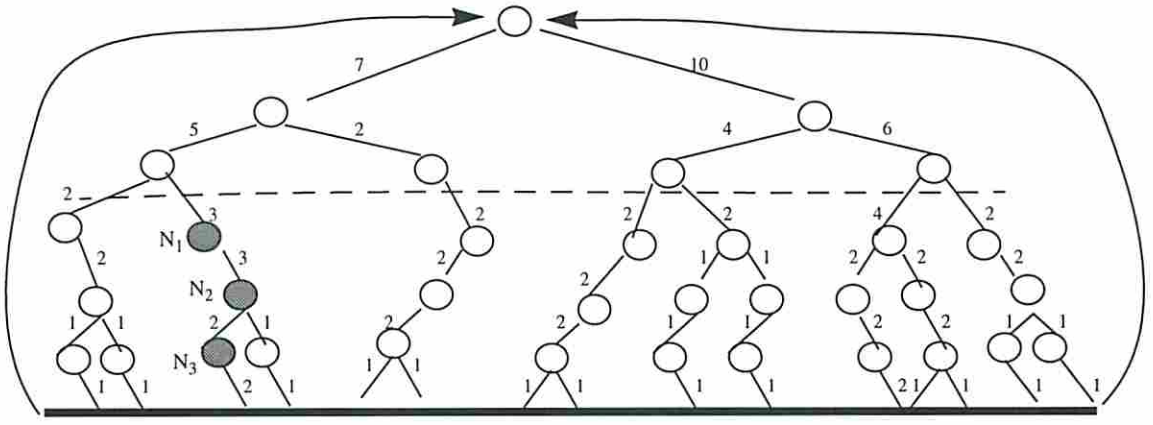


Fig.8

Once we built the Markov tree in Fig.8, we start the procedure *generate_seq* with parameter *ratio* = 2 and generate a subset of 8 vectors which best approximate the original sequence. To this effect, we use a modified version of the *dynamic weighted selection algorithm* [20]. In that approach, a similar structure with DMT_0 is built; more precisely, a full tree having on the leaves the symbols that need to be generated. The counts on the edges are dynamically updated and the symbols are generated according to their probability distribution. For this, a single random number generator is required in order to divide the interval [0,1] into subintervals that correspond to symbols' probabilities. At each level, the random number is compared to the left probability: if lower, a zero value is generated; if greater, a one value is generated and the number is decreased by the left probability. In our case, this strategy is used only to generate the first vector. After that, to ensure a minimal level of error, we use an *error controlling mechanism* in a greedy fashion. More precisely, at each level in the lower Markov tree, in order to decide whether a zero or one has to be generated, we compute the transition probabilities for both alternatives and choose the one that minimizes the absolute error accumulated up to that point . Simultaneously, the upper tree is parsed from the root to the leaves, according to the bits generated in the lower subtree. The procedure is then resumed until the needed number of vectors is generated.

In our example, if we assume that $x = 0.23$ is the first randomly generated number, based on the tree in Fig.8, since $0.23 < 7/17$ we take the left edge, generate a value 0 and x remains unchanged. At the second level, $x = 0.23 < 5/17$ so again we generate a '0' and leave x unchanged. Now $x = 0.23 > 2/17$ so a '1' is generated and x becomes $x = 0.23 - 2/17 = 0.11$. For the lower subtree rooted at the node denoted by the vector '001' (that is , we parse the upper subtree according to the already generated bits 0, 0, 1), to produce the second vector, we use the error controlling mechanism. Specifically, at node N_1 in Fig.8, the only choice is to take the right edge, generating a '1'. Next, at node N_2, the absolute error made for the transition probabilities becomes $|2/17-1/8| + |1/17 - 0| = 0.066$ if we take the left edge, and $|2/17 - 0| + |1/17 - 1/8| = 0.183$ if we take the right edge (8 is the length of the sequence to be obtained). The first choice is preferred and therefore a '0' is generated. At the last level, at node N_3, the decision is quite simple as we have only one descendent. Thus, after the first vector '001', we generate '101' as the second vector. The generation procedure continues for the lower subtree rooted at the node denoted by the vector '101' until the desired length $m =$

n/ratio is achieved. Despite its locality, this decision strategy performs very well in practice; as we'll see in the experimental part, the overall level of error is very small in all practicas cases.

In the second scenario, illustrated in Fig.9, the *model_size* parameter is set by the user as being 30 therefore the tree in Scenario 1 cannot be grown as such because the limit of 30 nodes is reached before the whole sequence is scanned. As a consequence, once we reach this limit (this actually happens immediately after processing the subsequence v_1, v_2, ..., v_9), we stop growing the tree and call *generate_seq* procedure with parameter *ratio* = 2 (Fig.9a). This will produce a subsequence of 4 vectors which best approximate the first 'segment' (v_1, v_2, ..., v_9) of the original sequence. After that we flush the model (keeping only the very last processed vector v_9) and start a new Markov tree as shown in Fig.9b. When the whole sequence is exhausted, based on this new Markov tree, we generate a new subset of 4 vectors which best approximate the second 'segment' (v_{10}, v_{11}, ..., v_{17}) of the original sequence.

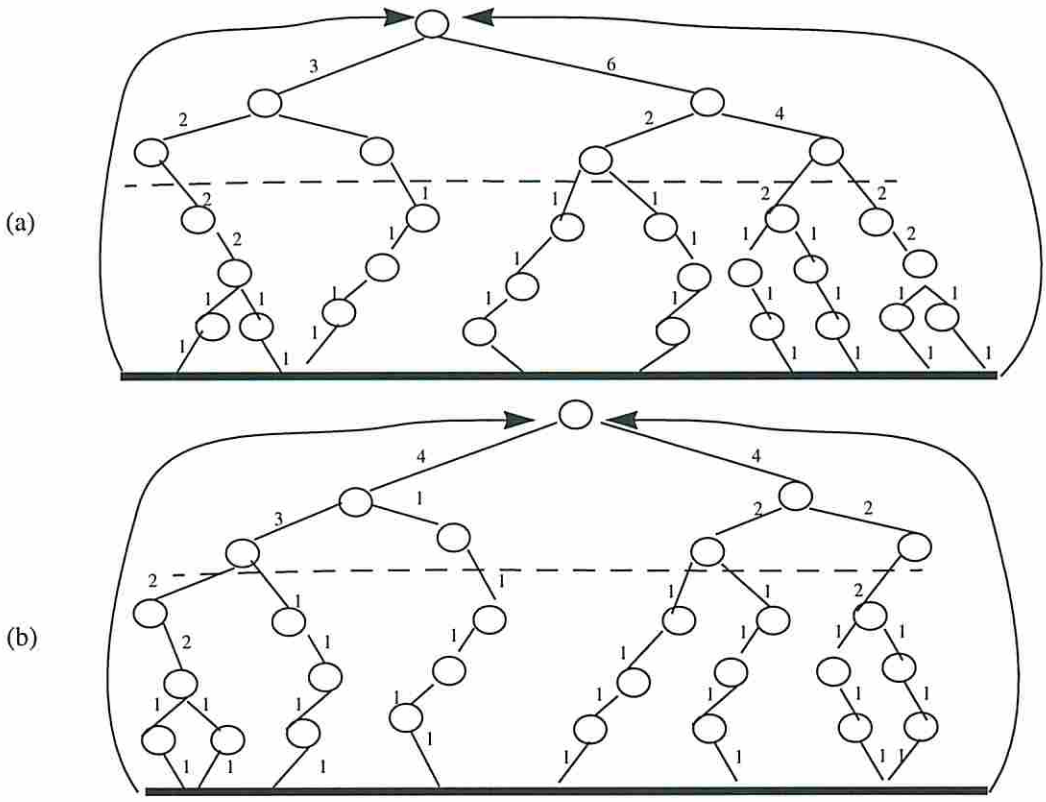


Fig.9

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

5. Practical considerations

5.1 Complexity related issues

The DMC modeling approach offers the significant advantage of being a *one-pass adaptive technique*. As a one-pass technique, there is no requirement to save the whole sequence in the on-line computer memory. Starting with an

initial empty tree DMT_1, while the input sequence is scanned incrementally, both the set of states and the transition probabilities change dynamically making this technique highly adaptive.

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

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

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

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

We point out that both these observations can be efficiently exploited only by a probabilistic technique such as DMC modeling; a deterministic technique (e.g. [13]) has no such inherent capability and therefore cannot avoid all the difficulties that arise from this type of complexity.

However, a natural question still remains: when should this growing process be halted ? If it is not halted, there is no bound on the amount of memory needed. On the other side, if it is completely halted we lose the ability to adapt if some characteristics of the source message change. A practical solution is to set a limit on the number of states in the DMC [17] as we actually did in Example 3. When this limit is reached, the Markov model is flushed and a new model is started. Although this solution may appear as too drastic, in practice it performs very well. The intuition behind this property is the capability of DMC model to adapt very fast to changes that occur while the input is scanned. A less extreme solution to limit model growing is also possible; we can keep a backup buffer that retains the last p vectors emitted by the source and whenever the model should be discarded, we may reuse this information to avoid starting the new model from the scratch.

5.2 Accuracy related issues

To see how the flushing technique affects the accuracy, let's assume that an input sequence of length n is modeled by the *DMC* approach. Suppose that during the building of the Markov model, flushing occurs after the first n_1 vectors, then after the next n_2 vectors, and so on. If the number of flushes is f, then $n_1 + n_2 + ... + n_f = n$. Let $v_i\,(u_i)$ be a vector from the initial (compacted) i-th subsequence (obtained due to successive flushes) and $v\,(u)$ a vector from the initial (compacted) sequence. We note that:

$$P(v^- = X \wedge v^+ = Y) = \frac{\sum_{i=1}^{f} n_i \cdot P(v_i^- = X \wedge v_i^+ = Y)}{n} \text{ and}$$

$$P(u^- = X \wedge u^+ = Y) = \frac{\sum_{i=1}^{f} \frac{n_i}{r} \cdot P(u_i^- = X \wedge u_i^+ = Y)}{\frac{n}{r}} \text{ where } r \text{ is the compaction ratio.}$$

If the i-th subsequence is approximated with an error less than ε_i, then the accuracy for the whole sequence is

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

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

An extreme case may occur when all patterns in the original sequence are distinct that is, there are no two identical vectors in the whole sequence. In such a case all terminal edges in the upper subtree of DMT_1 will have only one successor in the lower subtree, therefore the probabilistic choices during the generation phase will degenerate into a pure deterministic generation of a subsequence from the initial sequence. To avoid this, we propose to use a random pairwise generation phase that is, to select randomly the first vector in the upper subtree of DMT_1 and generate deterministically the second vector from the lower subtree of DMT_1. An alternative solution would be to partition the initial sequence into disjoint groups of bits and apply the DMC modeling technique to each individual group separately. As partitioning criteria we may use either functional considerations (e.g. separation between control bits and data-path bits) or the level of correlations among individual bits as was originally proposed in [14]. By partitioning, we increase the chances for repeated patterns in each group of bits therefore making probabilistic arguments more reasonable. The disadvantage however, is that this solution will ignore correlations across group boundaries.

6. Experimental results

The overall strategy is depicted in Fig.10.

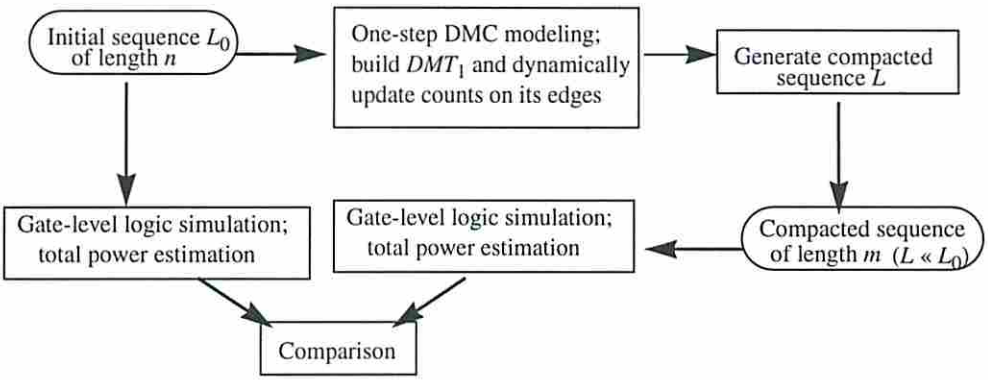


Fig.10

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

Starting with an k-bit input sequence of length n, we perform a one-pass traversal of the original sequence and simultaneously build the basic tree DMT_1; during this process, the frequency counts on DMT_1's edges are dynamically updated.

The next step in Fig.10 does the actual generation of the output sequence (of length m). As explained in Section 4, to generate the new sequence we use a modified version of the dynamic weighted selection algorithm presented in [20]. If the initial sequence has the length n and the new generated sequence has the length $m < n$ then the outcome of this process is a compacted sequence, equivalent to the initial one as far as total power consumption is concerned; we say that a *compaction ratio* of $r = n/m$ was achieved.

Finally, a validation step is included in the strategy; for short sequences we used the commercial tool PowerMill [2] whilst for long sequences we resorted to an in-house gate-level logic simulator developed under SIS. The total power consumption of some ISCAS'85 and ISCAS'89 benchmarks has been measured for the initial and the compacted sequences, making it possible to assess the effectiveness of the compaction procedure (under both zero- and real-delay models).

In Tables 1-2, we provide only the real-delay results for two types of initial sequences. Sequences of type 1 are large input streams having the same initial length n =100,000 and being then prime candidates for compaction; type 1 refers to biased sequences obtained by doing bit-level logical operations on ordinary pseudorandom sequences. The sequences of type 2 (having the length 4,000) are highly biased sequences obtained from real industry applications. As shown in Table 1, sequences of type 1 were compacted with two different compaction

ratios (namely r = 50 and 100); we give in this table the total power dissipation measured for the initial sequence (column 3) and for the compacted sequence (columns 4, 5). In the last column, we give the time is seconds (on a Sparc 20 workstation with 64 Mbytes of memory) necessary to read and compress data with DMC modeling. Since the compaction with DMC modeling is linear in the number of nodes in the structure DMT_1, the values reported in the last column are far less than the actual time needed to simulate the whole sequence. During these experiments, the number of states allowed in the Markov model was 20,000 on average.

Table 1: Total Power (uW@20MHz) for sequences of type 1

Circuit	Number of inputs	Power for initial seq.	Power for r = 50	Power for r = 100	Time for DMC (sec)
C432	36	1816.32	1838.89	1779.60	42
C499	41	3697.84	3546.65	3622.26	48
C880	60	3314.07	3229.85	3329.31	75
C1355	41	3205.27	3044.20	3109.18	48
C3540	50	10876.22	9910.08	10687.32	61
C6288	32	110038.69	114199.50	109077.42	37
s344	9	751.58	748.54	719.53	10
s386	7	818.11	844.58	848.80	8
s838	34	1052.05	1061.73	1091.14	41
s1196	14	3687.47	3702.32	3580.63	16
s9234	36	9192.75	9157.31	9209.75	43
		Average relative error (%)	2.80	2.93	

As we can see, the quality of results is very good even when the length of the initial sequence is reduced by 2 orders of magnitude. Thus, for C432 in Table 1, instead of simulating 100,000 vectors with an exact power of 1816.32 uW, one can use only 2000 vectors with an estimate of 1838.89 uW or just 1000 vectors with a power consumption estimated as 1779.60 uW. This reduction in the sequence length has a significant impact on speeding-up the simulative approaches where the running time is proportional to the length of the sequence which must be simulated.

The sequences of type 2 were compacted for two compaction ratios (r = 5 and r =10) using PowerMill [2]; to asses the potential of efficiency of the approach, for both original and compacted sequences, we report also the actual running time required by PowerMill to provide power estimates. The number of states allowed for the Markov model construction, was 5,000 on average; the CPU time for DMC modeling was below 3 seconds in all cases.

Table 2: Total Current (mA) for sequences of type 2

		Initial sequence		Compacted sequence		
Circuit	Number of inputs	Current (mA)	Time to simulate (sec)	Current (mA) r = 5	Current (mA) r = 10	Time to simulate (sec) r = 10
C432	36	0.4135	1186	0.4352	0.4066	120
C499	41	0.8188	2675	0.8337	0.8290	235
C880	60	0.7907	2289	0.7995	0.8023	274
C1355	41	1.1375	2993	1.1549	1.1461	284
C1908	33	1.2976	4034	1.2821	1.2833	367
C3540	50	3.4490	9467	3.3582	3.5719	1082
C6288	32	14.5749	88032	14.8020	13.3095	5005
			Average relative error (%)	2.15	2.63	

As it can be seen in Table 2, the average relative error is below 3% while the speed-up in power estimation is about one order of magnitude on average. For example, using the original sequence of 4000 vectors, PowerMill took for C432 about 1186 seconds to estimate a total current of 0.4135 mA. On the other side, using the sequence generated with DMC of only 400 vectors (r = 10), PowerMill estimated a total current of 0.4066 mA in only 120 seconds. This speed-up in power estimation becomes more significant for larger modules (e.g.C6288). We note also, that the results presented both tables 1 and 2, are significantly better than those reported in [14] in terms of accuracy, running time and memory requirements.

Finally, we compare our results with simple random sampling of vector pairs from the original sequences [21]. In simple random sampling, we performed 1,000 simulation runs with 0.99 confidence level and 5% error level on each circuit[1]. We report in Table 3 the maximum and average number of vector pairs needed for total power values to converge [11]. We also indicate the percentage of error violations for total power values, using as thresholds 5%, 6% and 10%. Using different seeds for the random number generator (and therefore choosing different initial states in the sequence generation phase), we run a set of 1,000 experiments for the DMC technique. In Table 4, we give the DMC results for the same thresholds as those used in simple random sampling.

Table 3: Results obtained for Simple Random Sampling

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

Table 4: Results obtained for DMC Approach

		Error violations (%)		
Circuit	Number of vectors	> 5%	> 6%	>10%
C432	2000	6.7	1.9	0.0
C499	800	0.3	0.0	0.0
C880	2000	1.4	0.1	0.0
C1355	800	0.2	0.0	0.0
C1908	800	1.9	1.2	0.0
C6288	2000	0.0	0.0	0.0

Once again, the results obtained with DMC modeling technique score very well and prove the robustness of the present approach. As we can see, using fewer vectors, the accuracy of DMC is higher than the one of simple random sampling in most of the cases. Noteworthy examples are benchmarks C499, C880, C6288 where less than 60% of the maximal number of vector pairs needed in random sampling for convergence are sufficient for DMC to achieve higher accuracy and confidence levels.

7. Conclusion

In this paper, we addressed the vector compaction problem from a probabilistic point of view. Based on dynamic Markov Chain modeling, we proposed an original approach to compact an original sequence into a much shorter equivalent one, which can be used after that with any available simulator to derive power estimates in the target circuit.

The mathematical foundation of this approach relies in Markov models; within this framework a family of

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

dynamic Markov trees is introduced and characterized as an effective and flexible way to model complex spatiotemporal correlations which occur during power estimation. The results obtained both on combinational and sequential benchmarks show that large compaction ratios of 1-2 orders of magnitude can be obtained without much loss in accuracy in total power estimates.

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

References

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

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

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

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

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

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

[7] 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.

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

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

[10] D. Marculescu, R. Marculescu, and M. Pedram, 'Information Theoretic Measures for Energy Consumption at Register Transfer Level', in *Proc. Intl. Workshop on Low Power Design,* pp. 81-86, April 1995.

[11] 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.

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

[13] 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.

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

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

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

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

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

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

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

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

How to Minimize Energy Using Multiple Supply Voltages

Jui-Ming Chang and Massoud Pedram

CENG-96-13

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

May 1996

Abstract

We present a dynamic programming technique for solving the multiple supply voltage scheduling problem in both non-pipelined and functionally pipelined data-paths. The scheduling problem refers to the assignment of a supply voltage level (selected from a fixed and known number of voltage levels) to each operation in a data flow graph so as to minimize the average energy consumption for given computation time or throughput constraints or both. The energy model is accurate and accounts for the input pattern dependencies, re-convergent fanout induced dependencies, and the energy cost of level shifters. Experimental results show that using four supply voltage levels on a number of standard benchmarks, an average energy saving of 40.48% (with a computation time constraint of 1.5 times the critical path delay) can be obtained compared to using a single supply voltage level.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One driving factor behind the push for low power design is the growing class of personal computing devices as well as wireless communications and imaging systems that demand high-speed computations and complex functionalities with low power consumption. Another driving factor is that excessive power consumption has become a limiting factor in integrating more transistors on a single chip. Unless power consumption is dramatically reduced, the resulting heat will limit the feasible packing and performance of VLSI circuits and systems.

The behavioral synthesis process consists of three phases: allocation, assignment and scheduling. These processes determine how many instances of each resource are needed (allocation), on what resource a computational operation will be performed (assignment) and when it will be executed (scheduling). Traditionally, behavioral synthesis attempts to minimize the number of resources to perform a task in a given time or minimize the execution time for a given set of resources. It is necessary to develop behavioral synthesis techniques that target lower power dissipation in the circuit.

The most effective way to reduce power consumption is to lower the supply voltage level for a circuit. Reducing the supply voltage however increases the circuit delay. Chandraskan et. al. [ChPo92] compensate for the increased delay by shortening critical paths in the data-path using behavioral transformations such as parallelization or pipelining (If the circuit is already

pipelined, then re-timing may be applied). The resulting circuit consumes lower average power while meeting the global throughput constraint at the cost of increased circuit area.

More recently, the use of multiple supply voltages on the chip is attracting attention. This has the advantage of allowing modules on the critical paths to use the highest voltage level (thus meeting the required timing constraints) while allowing modules on non-critical paths to use lower voltages (thus reducing the energy consumption). This scheme tends to result in smaller area overhead compared to parallel architectures. For this scheme to work, we will however need to insert level-shifters between connected modules that operate at different supply voltage levels. The area and energy costs of these level shifters must be taken into account when comparing a multiple-supply voltage design with that of a fixed-supply voltage design.

There are however a number of practical problems that must be overcome before use of multiple supply voltage becomes prevalent. These problems include routing of multiple supply voltage lines, area/delay overhead of required level converters, and lack of design tools and methodologies for multiple supply voltages. The first issue is an important concern which should be considered by any designer who wants to use multiple supply voltages. That is, there is a trade-off between lower energy dissipation and higher routing cost. If however energy dissipation is a critical design issue for the designer, he may choose to accept the higher routing overhead (or add to the cost by increasing the number of routing layers) in return for significantly lower energy dissipation. The remaining issues (that is, level shifter cost and lack of tools) are addressed in this paper. That is, we will show that the area/delay overhead of level shifters is relatively small and will present an effective algorithm for using multiple supply voltages during behavioral synthesis.

In this context, an important problem is to assign a supply voltage level (selected from a finite and known number of supply voltage levels) to each operation in a data flow graph (DFG) and schedule various operations so as to minimize the energy consumption under given timing

constraints (i.e., total computation time for non-pipelined designs or throughput constraint for pipelined designs). We will refer to this problem as the *multiple-voltage scheduling* problem or the MVS problem for short.

In this paper, we tackle the problem in its general form. We will show that the MVS problem is NP-hard even when only two points exist on the energy-delay curve for each module (these curves may be different from one module to another), and then propose a dynamic programming approach for solving the problem. This algorithm which has pseudo-polynomial complexity (cf. Chapter 4.4) produces optimal results for trees, but is not the optimal for general directed acyclic graphs. We will show that energy minimization given the total computation time or throughput constraints is equivalent to minimizing the average power dissipation. The dynamic programming technique is then generalized to handle functionally pipelined designs. This is the first time that the use of multiple supply voltages in a *functionally pipelined* design is considered. We will present a novel *revolving schedule* for handling these designs.

The report is organized as follows. In Chapter 2, we summarize related work. In Chapter 3, we describe timing and energy consumption models for non-pipelined designs. In Chapter 4, we present a dynamic programming approach for solving the multiple-voltage scheduling problem for the tree-like DFG's and then for general DFG's. In Chapter 5, we extend the approach to functionally pipelined designs. Experimental results and concluding remarks are provided in Chapters 6 and 7.

Chapter 2

Related Problems

The Multiple-voltage scheduling problem (MVS) as described above is closely related to the *circuit implementation problem* as defined in [LiLi92]. The problem is to minimize the total gate area in a circuit by selecting a gate implementation for each circuit node while meeting a timing constraint. It was shown in [LiLi92] that even under a fanout (load) independent delay model, with two implementations per circuit node, equal signal arrival times at inputs, and chain-like circuit structure, the problem of finding a solution where circuit area (energy) $\leq \alpha$ and signal arrival time $\leq \beta$ is *NP-complete.* We will show (cf. Chapter 4) that the MVS problem for minimum energy is also *NP-complete.*

Another similar problem is that of delay constrained technology mapping [ToMo90] [ChPe92] [TsPe94]. Our method for solving multiple voltage scheduling is similar to the method used in delay constrained technology mapping [ChPe92] [TsPe94]. In these works, the authors cover a subject graph by a library of pattern graphs with the goal of minimizing area/power while satisfying given timing constraints. The approach consists of two steps: First, the delay functions (which capture arrival time-energy trade-offs) are generated at all nodes of the circuit using a post-order traversal. In the second step, a pre-order traversal is performed to determine the gate mapping of each node based on the user-specified required times at the circuit outputs.

The MVS problem was tackled in [RaSa95] where the authors proposed an algorithm for

minimizing the energy consumption of a non-pipelined design while meeting the computation time constraint. The authors assume that delay vs. supply voltage curves for all modules in the design library are given and propose an iterative improvement algorithm for solving the problem. The approach is optimal for general directed acyclic graphs. However, the authors make a number of simplistic and rather unrealistic assumptions (e.g., the assumption that the difference of squares of the consecutive voltages on the delay vs. voltage curve is fixed; the independence of energy consumption of a module from data activity at its inputs; identical latency (absolute delay) vs. supply voltage curves for all modules in the circuit including adders and multipliers). The first assumption enables the authors to reduce the problem of $Min\ \sum_{i \in modules} E_i$ under given computation time constraint where E_i is the energy consumption of module i to $Max \sum_{i \in modules} d_i$ where d_i is the delay of module i for the corresponding voltage assignment. However, if the voltage vs. delay follows the curve that they are based on, the first assumptions will never be satisfied. If the assumptions made in [RaSa95] do not hold for a given problem instance, then their proposed algorithm will produce a suboptimal solution without any performance guarantee.

Usami and Horowitz [UsHo95] proposed a technique to reduce the energy consumption in a circuit by making use of two supply voltage levels. The idea is to operate gates on the critical paths at the higher voltage level and the gates on the non-critical path at the lower voltage level. In this manner, the energy consumption is minimized without affecting the circuit speed.

Power Profiler [MaKn95] primarily uses a *genetic search algorithm* to solve the multiple voltage scheduling problem. The algorithm has exponential worst-case complexity and hence the results are suboptimal for large problem instances where computation time is bounded due to practical considerations. Power Profiler does not address conditional branches and its energy model does not support input data dependency. Finally, effects of level shifter is ignored and the case of functionally pipelined design is not addressed.

Johnson and Roy presented an ILP based formulation for the multiple voltage scheduling

problem for non-pipelined design in [JoRo96]. The formulation is general, handles timing and resource constraints, and accounts for the cost of level shifters. However, it does not address conditional branches; nor does it consider functional pipelining. The energy model again is a data-insensitive model which ignores the effect of input activities on the energy dissipation of a module. Finally, it has exponential worst-case complexity and (as can also be seen in the experimental results section of [JoRo96]) cannot handle large examples.

The inability to handle functionally pipelined data-path seriously limits the applicability of the above techniques. In functionally pipelined data-path, lowered energy does not necessarily imply lower performance. This is because in a functionally pipelined data-path, performance is described by the throughput of the functional pipeline, and not the total computation time for each data sample. Our method enables the designer to use longer total computation time for each data sample and thus assign lower voltage levels to modules while maintaining throughput of the functional pipeline without any loss in performance. The result is of course lower energy dissipation.

In comparison to previous work, our algorithm is able to find the minimal energy solution under timing and/or throughput constraints, handles functional pipelining, explicitly supports the conditional branches, uses an energy model that takes different input data switching activities into consideration, and has pseudo-polynomial time complexity.

Chapter 3

Energy-delay Curves

We assume there are latches on the inputs of all modules to synchronize the input arrival times, and no multiple module activations per cycle occurred.

3.1 The timing model

Let c-step denote a control step (clock cycle), the basic unit of time used in the DFG in behavioral level. When the supply voltage level of a module is lowered, the delay increases. Let c-step denote the basic unit of time used in the DFG. For a given length of a c-step, t_c, an operation may thus become a *multi-cycle operation.*

Each multi-cycle operation starts its execution on the boundary of a c-step, but it may finish its execution within a c-step. We do not handle operation chaining in our new method for several reasons. Chaining can be done if the length of the c-step is large. But if we do not perform chaining in every possible c-step, the chance of accumulated dead time (time interval between the end of operation and the beginning of next time step) will be increase. Dead time on some paths means the chance is reduced for using the lower supply voltage for some operations in these paths, which shows we did not achieve the optimal solution. If chaining is done as often as possible with a given long c-step length, the resulting situation is similar (although not exactly the same) as using a smaller c-step but without chaining. We

can also perform the chaining by slightly extending the timing model we use in Chapter 3.1. Let the starting time of operation i be the maximum among the output arrival time of all its predecessors if the output arrival time of operation i is still within the same current c-step of its max-arrival time predecessor. All the other methodology need not be modified. In many cases, if you choose the initial c-step length to be at least the maximum among operation delay when they are operated in $5V$, most of the operations become multi-cycle operations when the supply voltage is reduced. Therefore, the chance of performing operation chaining is small. Let t_i^s be the starting time of operation i, a_i the *output arrival time* of operation i, d_i the execution time (delay) of operation i, t_c the length of a c-step, then we have the following:

$$\begin{aligned} a_i &= t_i^s + d_i \\ t_i^s &= max_{(j,i)}\lceil a_j/t_c \rceil \cdot t_c \end{aligned}$$

where operation j is a predecessor of operation i in the DFG.

3.2 The energy dissipation model

We present in this Chapter two computational models for energy dissipation at behavioral level. Our optimization algorithm is however independent of the specifics of these energy models. More precisely, any energy macro-model whose parameters depend on the input and/or output activity factors can be used here. This includes for example, the power macro-model reported in [LaRa94].

We assume that the dynamic energy dissipation in a functional unit is given by this equation:

$$E_{FU_i} = F_i(\alpha_{i,1}, \alpha_{i,2}) \cdot V_i^2 \tag{3.1}$$

where V_i is the supply voltage of functional unit FU_i, $\alpha_1^{FU_i}$ and $\alpha_2^{FU_i}$ are the average switching activities on the first and second input operands of FU_i, respectively; F_i is a function of $\alpha_1^{FU_i}$

and $\alpha_2^{FU_i}$ and in general may be nonlinear. We propose two methods to calculate E_{FU_i}/V_i^2 given the pairs ($\alpha_{i,1}$,$\alpha_{i,2}$).

The first method is based on look-up table, that is, we store energy dissipation values for various (α_1,α_2) combinations and interpolate to calculate the energy value for a given (α_1^*,α_2^*) combination which is not found in the table. To be more specific, we conduct a set of extensive real-delay gate-level simulations for a combination of ($\alpha_{i,1}$,$\alpha_{i,2}$) by generating biased input sequences that exhibit these average activities. α_1 and α_2 are taken from the interval [0,1] with increments of 0.1 for the total number of 10 × 10 combinations. We then use interpolation (by table look up) for every region (a small square with side length equal to 0.1) to estimate the energy value for the points that fall inside this region. The accuracy can be increased if higher resolution is used (e.g. increment size is set to 0.05, etc.). The 3-dimensional mesh plot of $F(\alpha_1, \alpha_2)$ for adder and multiplier are shown in Fig. 3.1 and 3.2, respectively. This method can achieve very high accuracy based on the number of entries in the look-up table.

The second method is based on energy macro-modeling using a linear equation with α_1 and α_2 as random variables. More precisely, we use the least square fit to find a plane in the 3-dimensional space that best fits the set of points ($\alpha_{i,1}$, $\alpha_{i,2}$, E_{FU_i}/V_i^2) for each module FU_i. From the least square fit, we obtain:

$$F_i(\alpha_{i,1}, \alpha_{i,2}) = C_1 \times \alpha_{i,1} + C_2 \times \alpha_{i,2} + C_3 \tag{3.2}$$

From Fig. 3.1 and 3.2, we can see that the plane (linear) approximation for $F(\alpha_{i,1}, \alpha_{i,2})$ is reasonable for adders and multipliers and obviously uses less storage space for each module compared to table look-up approach. For the least square fit of the 16-bit adder in our library, we obtain:

$$F_i(\alpha_{i,1}, \alpha_{i,2}) = 3.0232 \times \alpha_{i,1} + 3.14912 \times \alpha_{i,2} + 2.1396$$

For the least square fit of the 16-bit multiplier in our library, we obtain:

$$F_i(\alpha_{i,1}, \alpha_{i,2}) = 220.5524 \times \alpha_{i,1} + 419.52 \times \alpha_{i,2} + 353.14$$

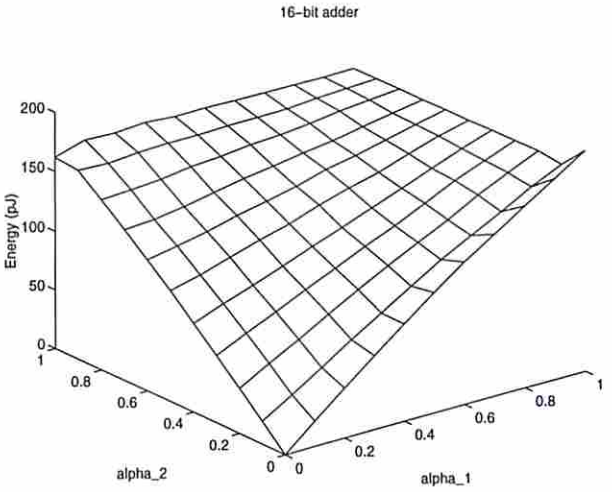


Figure 3.1: Energy vs. 2 switching activities for add16 (shown for $\Delta\alpha$=0.1, at 5V)

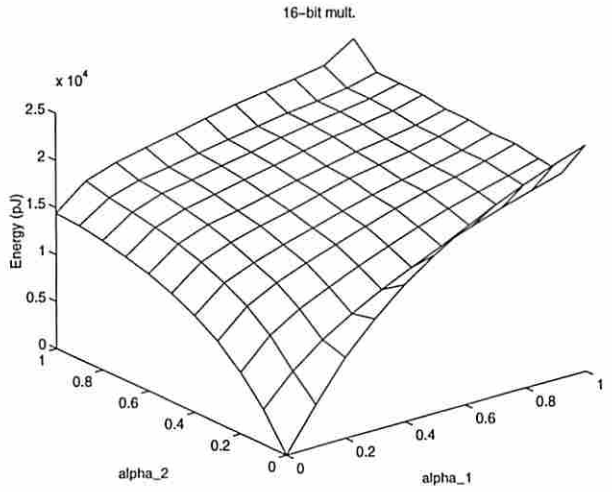


Figure 3.2: Energy vs. 2 switching activities for mult16 (shown for $\Delta\alpha$=0.1, at 5V)

Obviously C_i, $i = 1 \ldots 3$ depends on the module type, the input data width, the technology and logic style used, and the internal module structure. We obtain the C_i, $i = 1 \ldots 3$ for every module in our library using gate-level simulation and the least square fit. Obviously, the accuracy of the model can be improved by using more variables. For example using a dual bit type model, we can write:

$$F(\alpha_{LSB,1}, \alpha_{MSB,1}, \alpha_{LSB,2}, \alpha_{MSB,2}) = C_1 \cdot \alpha_{i,1}^{LSB} + C_2 \cdot \alpha_{i,1}^{MSB} + C_3 \cdot \alpha_{i,2}^{LSB} + C_4 \cdot \alpha_{i,2}^{MSB} + C_5$$

where $\alpha_{i,x}^{LSB}$ and $\alpha_{i,x}^{MSB}$ denote the average switching activity of the LSB and MSB of operand x of module i. The purpose of this Chapter is however not to present the most accurate energy macro-model, but to show that any data-sensitive energy model can be used to provide our dynamic programming approach with the energy values for the modules.

To validate our energy model, we present some results for the set of data-path modules used in our library which are implemented in a 1 μ technology (cf. Table 3.1, 3.2 and 3.3) using the two methods presented above. The first column in each table gives the functional unit name; The second column gives the estimated energy dissipation obtained by using table look-up method (interpolation) with switching activities α_1, α_2 (which are in turn obtained from analysis of the input vector sequence); The third column gives the actual energy dissipation obtained by gate-level simulation of the module using the same vector sequence (observation set); The fourth column shows the average error obtained by using table look-up method. The fifth column gives the estimated energy dissipation obtained by using macro-model equation (3.2); and finally the last column shows the average error obtained by using macro-model equation (3.2). Table 3.1, 3.2 and 3.3 present results when the input sequence has average activities of α_1=α_2=0.5 (random data for both operands), α_1=0.5, α_2=0.1 (random data for one operand only) and α_1=α_2=0.1 (biased data for both operands). The voltage used in these tables is 5 volts. It is clear from these results that the table look-up method (with 100 entries) remains accurate over the range of α value whereas the curve fitting method becomes inaccurate for small α. Note that the error is rather high for small operand activities when using Equation

Circuit	E_{est}^{TLU} (pJ)	E_{actual} (pJ)	$error^{TLU}$	$E_{est}^{Eq.(2)}$ (pJ)	$error^{Eq.(2)}$ %
add16	131.65	131.18	0.35	130.71	0.36
mult16	17582.86	17607.02	0.13	16831.82	4.40
Mux16: 2 to 1	24.05	24.38	1.35	25.04	2.71
Mux16: 4 to 1	68.03	69.59	2.24	72.95	4.83

Table 3.1: Data-Path Circuits and their gate-level simulation results under random sequence with $\alpha_1 = \alpha_2 = 0.5$. (V=5volts)

Circuit	E_{est}^{TLU} (pJ)	E_{actual} (pJ)	$error^{TLU}$	$E_{est}^{Eq.(2)}$ (pJ)	$error^{Eq.(2)}$ %
add16	98.86	100.94	2.06	98.97	1.95
mult16	14421.43	14097.43	3.26	12633.34	10.39
Mux16: 2 to 1	18.27	19.10	4.34	20.38	6.70
Mux16: 4 to 1	49.66	47.37	4.83	50.73	7.09

Table 3.2: Data-Path Circuits and their gate-level simulation results under random sequence with $\alpha_1 = 0.5$, $\alpha_2 = 0.1$. (V=5volts)

(3.2). This is to be expected for two reasons: 1) The plane fit becomes less accurate for input activities range far from 0.5 as can be seen in Fig. 3.1, 3.2. At low activities (say 0.1 for the two operands), the energy dissipation is small, hence the same absolute difference between our estimates and the true energy dessipation gives rise to a large percentage error. This is the factor which explains the large percentage error for the 16-bit adder in Table 3.3.

We have also assumed that the range of V_i is such that the major source of energy consumption is the capacitive charging/discharging; that is, E_i/V_i^2 remains constant as V_i is scaled down. This may not be true if static standby current becomes important at very low voltages.

With this macro-modeling, we can calculate the energy consumption of each module alternative under different supply voltages and switching activities. Note that $\alpha_1^{FU_i}$ and $\alpha_2^{FU_i}$ are calculated by using behavioral simulation of the given DFG using the set of user-specified (application-dependent) input vectors.

Circuit	E_{est}^{TLU} (pJ)	E_{actual} (pJ)	$error^{TLU}$	$E_{est}^{Eq.(2)}$ (pJ)	$error^{Eq.(2)}$ %
add16	34.77	36.63	5.07	68.84	87.93
mult16	6880.95	7083.91	2.87	10418.44	47.07
Mux16: 2 to 1	5.77	6.07	4.94	7.71	27.02
Mux16: 4 to 1	17.28	18.19	4.53	22.94	26.11

Table 3.3: Data-Path Circuits and their gate-level simulation results under random sequence with $\alpha_1 = \alpha_2 = 0.1$. (V=5volts)

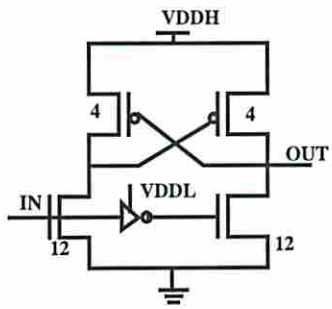


Figure 3.3: A Level Shifter Circuit

x \ y	1.5	2.4	3.3	5
1.5	0	38.4	58.4	88.0
2.4	28.0	0	64.0	128.0
3.3	36.0	49.6	0	142.4
5	73.6	88.0	104.0	0

Table 3.4: Average energy consumption (in units of pJ) of 16-bit level shifter per logic transition (all 16-bits are switching) produced by Spice simulation. Note that, entry $(\mathbf{x}, \mathbf{y})$ in this table is the energy used for converting the output of a module which uses supply voltage x to the input of a module which uses supply voltage y.

Let $\mathbf{E_{LS\_i}}$ be the energy used by level shifter i in the circuit when its input changes once. $\mathbf{E_{LS\_i}} = \mathbf{E_{LS\_static\_i}} + \mathbf{E_{LS\_dynamic\_i}}$. For a well designed level shifter, such as the one shown in Fig. 3.3 (taken from [UsHo95]), $\mathbf{E_{LS\_static\_i}} = \mathbf{0}$. The energy consumed in a 16-bit level shifter per voltage level transition is given in Table 3.4. Note, the bits of level shifter are clearly independent; our Spice simulation was done on a single bit level shifter. Since our tables are energy consumption of 16-bit wide functional units, we decided to report the energy dissipation for a 16-bit level shifter to underline the relative magnitude of the level shifter energy consumption in the 16-bit data-path. Using this table and the switching activity of the level shifters obtained from behavioral simulation, the dynamic energy consumption of the level shifters used in the design can be easily calculated. The propagation delay through a level shifter for typical load value is less than $1ns$ (which makes it negligible compared to the propagation delay through the modules) (cf. Table 6.3). The delay cost of the level shifter shown in Fig. 3.3 is 1 (ns) by Spice simulation, which is much smaller than the minimal delay of modules such as adder ($\geq$ 20 ns) or multiplier in our Table 6.1 or 6.3. Note that at most one (sometimes zero) level shifter will be used to follow any module if the next module in the same path using a different (the same) voltage in any path in the circuit. We can therefore absorb the delay costs (1 ns) for level shifters into the delay of the functional units they follow, because in the module library, the minimum module delay is at least 20 times larger than the level shifter delay. As for the accumulation of the level shifter delays, since at most one level shifter follows any module on any path, the relative magnitude of shifter delays is considered very small compared to the accumulation of delays from modules in the corresponding path. A level shifter is only used following a module whose output has to be scaled up or down. So if we have n modules on the critical path, there will be at most n level shifters; the ratio of level shifter delay to adder delay is 0.05. This ratio is 0.01 for the multipliers. Hence the level shifters can not increase any path delay by more than 5 %.

Multiplexors will be used to route data in for non-overlapping operations that share the same module sequentially. From Table 3.1, we can also see that the energy consumed in multiplexors is relatively small compared to energy dissipation in adders and multipliers. In any case, Mux'es are needed with or without multiple supply voltages.

The *average power* is given by:

$$\textbf{average power} = \frac{\mathbf{E_{FU}} + \mathbf{E_{LS}}}{\mathbf{T_{comp}}} \tag{3.3}$$

where E_{FU} and E_{LS} are the total energy consumption of all modules and all level shifters and $\mathbf{T_{comp}}$ is the total computation time for one data sample. $\mathbf{T_{comp}}$ is a user-specified constraint and is known. Therefore, if we minimize ($\mathbf{E_{FU}} + \mathbf{E_{LS}}$), we are minimizing the average power dissipation in the circuit.

We assume (and *enforce*) that each module is **active** only when it is performing an operation, and is in the **sleep mode** at all other times. The sleep mode can be achieved by clock gating or use of flip-flops with enable/disable.

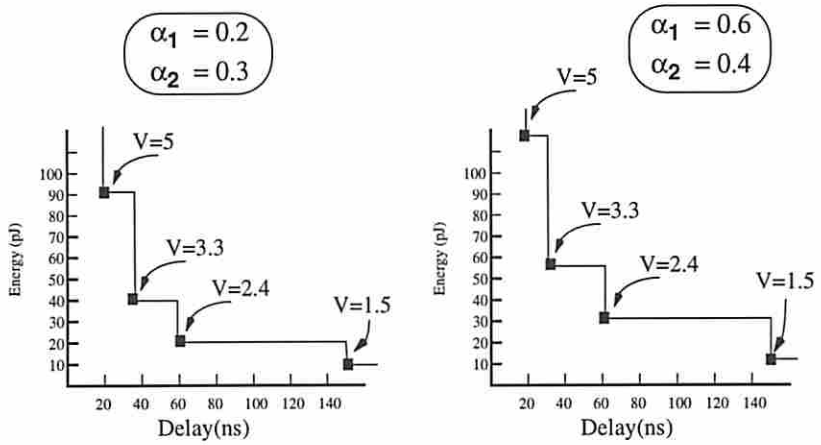
3.3 Trade-off curves

We assume that for each module in the library, the information according to our second method in Chapter 3.2 is stored as in Fig. 3.4(a). During dynamic programming, we have to calculate the energy-delay trade-off points for each instance of the module. At that time, the input operand activities (α_1, α_2) are known from a behavioral simulation of the DFG; the information shown in Fig. 3.4(a) can be thus used to generate the energy delay curve shown in Fig. 3.4(b) and (c) for any given pair of (α_1, α_2). Points on the curve represent various voltage assignment solutions with different trade-offs between the speed and energy.

Note, the energy-delay curves are the actual starting point for the energy optimization. The points in energy-delay curves can be generated by all power estimation methods, not limited to our simpler model. Some power estimation can take the switching activity α^{FU} inside the

16-bit Adder		Regression Coefficients (pF)		
Voltage	Delay (ns)	C_1	C_2	C_3
5.0	20.4	3.02	3.14	2.14
3.3	36.1	3.78	3.46	2.02
3.3	48.3	3.02	3.14	2.14
2.4	60.2	3.02	3.14	2.14
1.5	149.75	3.02	3.14	2.14

(a)



(b) (c)

Figure 3.4: Our module library using the second method in Chapter 3.2 for a 16 bit adder and the energy vs. Delay curves under different α's

modules into consideration and produces the more accurate points in these curves.

We only keep *non-inferior* points on each curve. Point p^* is a non-inferior point if and only if there does not exist a point $P = (t, e)$ such that either $t \leq t^*$, $e < e^*$ or $t < t^*$, $e \leq e^*$.

Chapter 4

The Scheduling Algorithm

We first describe scheduling of DFGs which are **trees**. The goal here is to obtain a minimum energy solution that binds the operations in DFG to modules in the library while satisfying a computation time constraint. We first show that the decision version of this problem is *NP-complete.*

Theorem 4.1 *Multiple-voltage scheduling problem for minimum energy is* NP-complete.

Proof: The multiple-voltage scheduling problem is defined as follows: given a behavioral description of an algorithm in the form of a data-flow graph (DFG), a module library, and a fixed number of supply voltage levels, find a solution where energy dissipation in the DFG $\leq \alpha$ and total computation time $\leq \beta$. By restricting our DFG into a chain and allowing only two implementations for each operation in the chain, our problem is identical to the circuit implementation problem which is known to be NP-complete [LiLi92]. Hence, our problem is proven to be NP-complete by restriction [GaJo79]. □

It is a simple exercise to formulate this problem as an integer linear programming problem (ILP). However, the ILP formulation does *not* take advantage of the *problem structure* and is in general very difficult and inefficient to solve. Instead, we use a *dynamic programming* approach as described next.

First, a post-order traversal is used to determine a set of possible output arrival times at the root (primary output) of the tree. Then a pre-order traversal is performed starting from the root to recursively determine the specific solution on each node in the tree based on the given computation time constraint.

We calculate on each node a delay function (or delay curve) where each point on that curve relates the accumulated energy consumed on the subtree rooted at that node (or operation) and the output arrival time of the node when a certain module (with certain supply voltage level and hence delay) is used to perform that operation. Different module alternatives for the same operation give rise to different points on the delay curve. The accumulated energy is the sum of energy consumed in all modules in that subtree (including the root of that subtree) plus all energy consumed in the necessary level shifters.

The delay function is therefore represented by a set of ordered pairs of real positive number (t, e), where a piecewise linear function $e = f(t)$ can be constructed which describes the set of all possible energy-delay trade-off solutions.

4.1 Post-order traversal

A post-order traversal of the tree is performed, where for each node n and for each module alternative at n, a new delay function is produced by appropriately *adding* the delay functions at the children of node n. Adding must occur in the common region among all delay functions in order to ensure that the resulting merged function reflects feasible matches at the children of n (cf. Fig. 4.2). Note that the energy consumed in *level shifters* is computed during the post-order traversal by keeping track of the voltages used in the current node and its children (using Table 3.4 and switching activity information). The delay function for successive module alternatives at the same node n are then merged by applying a *lower-bound merge* operation on the corresponding delay functions. The procedure is repeated until all combinations of points on curves A and B are exhausted. To illustrate the lower-bound merge operation, see Fig. 4.1.

The delay function addition and merging are performed recursively until the root of the tree is reached. The resulting function is saved in the tree at its corresponding node. Thus each node of the tree will have an associated delay function. The set of (t, e) pairs corresponding to the composite delay function at the root node defines a set of arrival time-energy trade-offs for the user to choose from.

To illustrate the delay function addition, consider the example in Fig. 4.2. It shows the addition of the children's curve to its parent for a module alternative m match at node C. The children of this match are nodes A and B. The delay functions for A and B are known at this time. To compute a point on the delay function for node C, we select a point from delay function of the children, i.e. point a on delay curve of node A. The delay of point a is 3 units. So, we look for a point on the delay function of node B with delay less than 3 which has the minimum energy. In this example, d is the desired point. We therefore combine points a and d to generate point a' on delay-curve(C), with

$$\begin{aligned} arrival(a') &= t^s_{a'} + delay(m) \\ t^s_{a'} &= \lceil arrival(a)/t_c \rceil \cdot t_c \\ energy(a') &= energy(a) + energy(d) + energy(m) \end{aligned}$$

4.2 Pre-order traversal

The user can now use the total computation time constraint $\mathbf{T_{comp}}$ on the root of the tree and perform a pre-order traversal to determine the specific point on each curve associated with each node of the tree. The timing constraints of children at the root is computed as $\mathbf{T_{comp}}$ - $\mathbf{t_{delay}}$, where $\mathbf{t_{delay}}$ is the delay of the module alternative of the root that makes the root satisfy arrival time $\leq$ $\mathbf{T_{comp}}$ and has the minimum energy. This module selection and timing constraint propagation technique is applied recursively at all internal nodes during the

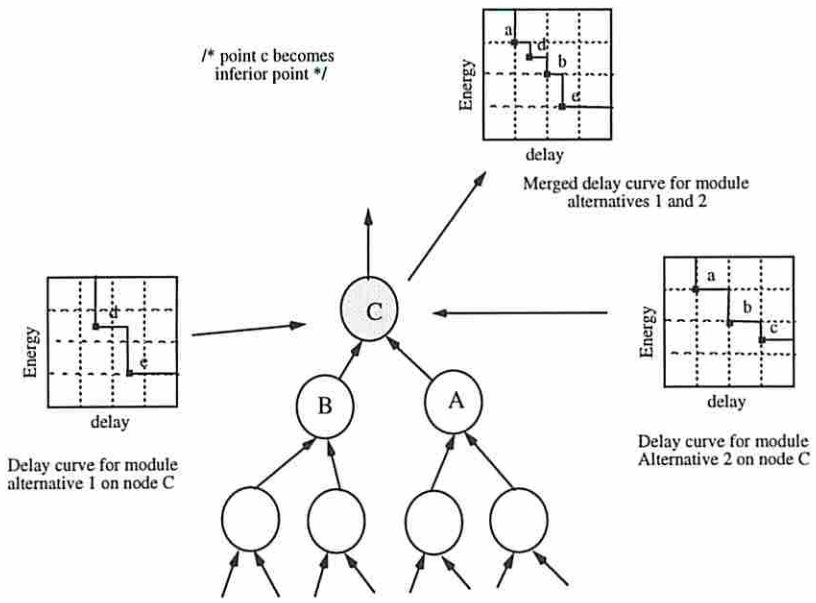


Figure 4.1: Lower bound merging of delay curves

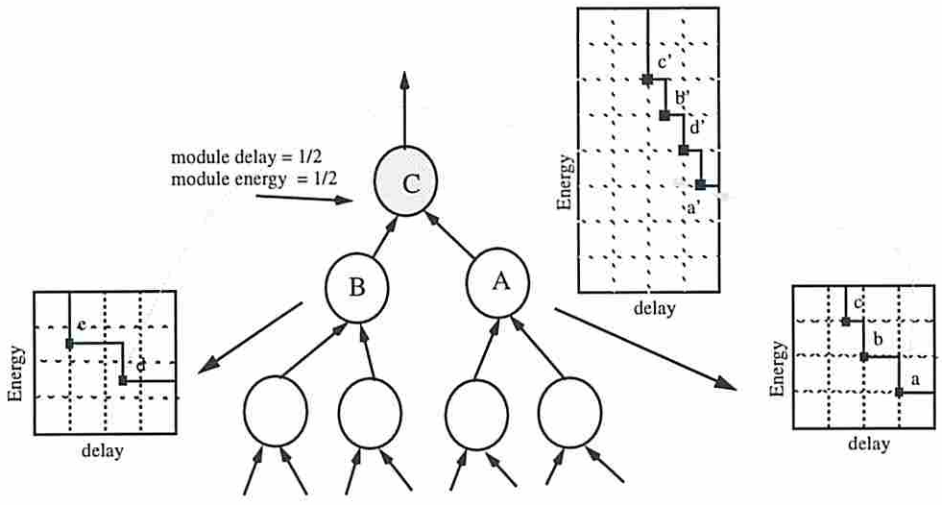


Figure 4.2: An example of adding two curves to obtain the parent curve

pre-order traversal.

4.3 Extension to general DFG's

The delay functions for nodes of a *general DFG* are computed by a post-order traversal as was the case for a tree-like DFG. The key question is how to add up the energy cost of children of a node during the post-order step. Consider Fig. 4.3, where node n fans out to nodes p, q and r and node u, v and w are re-convergent fanout nodes. When we try to calculate the energy-delay curve at a node like q, we face the problem of deciding what the energy-delay curve along the input line coming from n should be. If we use the energy-delay curve of n, then we will overestimate the energy contribution of node n (and its transitive fan-in cone) when we reach the re-convergent fanout node u. On the other hand, if we scale down the energy value of node n by its fanout count (of 3), then we will underestimate the energy contribution of n (and its transitive fan-in cone) at u (although we will correctly calculate the energy contribution of n at w); Finally, if we scale down the energy value of n by 2, then we will overestimate the energy contribution of n at w (although we will correctly calculate the energy contribution of n at v). This simple example shows that in fact it is not possible to propagate the energy value of n (or any scaling of this energy) up the multiple fanout point without causing a miscalculation at some node in its transitive fanout cone.

We have adopted a heuristic whereby the energy value of a multiple fanout point is divided by its fanout count when propagate upward in the DFG. This heuristic is also adopted in technology mapping programs such as MIS [DeGa97] or ad-mapper [ChPe92] and tends to produce better results. Then a pre-order traversal is used to select the specific points on each delay curve associated with each node in the DFG. However, due to the DAG structure of the DFG, a node in the DFG may have more that one parent. Therefore a node is visited more than once during the pre-order traversal of the DFG. Different points on the delay curve of the same node may be selected during different visits to that node. If during the pre-order traversal we

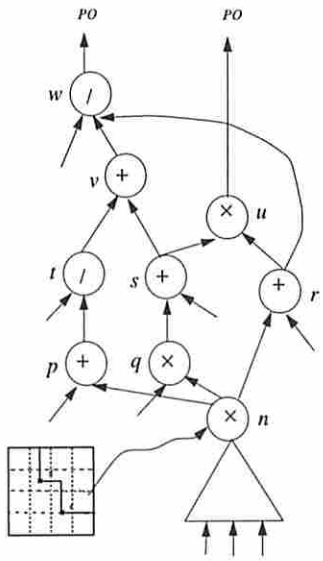


Figure 4.3: Post-order energy-delay curve propagation in a DAG (*PO* denotes a primary output node)

come to a node which has been mapped previously, we will check to see whether the previous solution at that node satisfies the current timing requirement. If so, we keep the mapping; otherwise, we replace it with another solution which satisfies the current timing requirement and has minimum energy. The new solution may have higher energy compared to the previous solution, however it will satisfy the timing constraint. Note that satisfying the current timing can only decrease the delay (output arrival time) for the previously mapped subtrees. The above regeneration of solutions at the same node ensures that all timing constraints are met at all nodes in the DFG.

If a solution generated at some multiple fanout node in the DFG is overwritten during the pre-order step (when coming back along a different path), then all energy-delay values in the *transitive fanout cone* of the node in question must be updated by doing a second post-order traversal of the DFG. (The second post-order traversal is only done after module selection for all DFG nodes is completed. It purpose is to calculate the correct signal arrival times at the

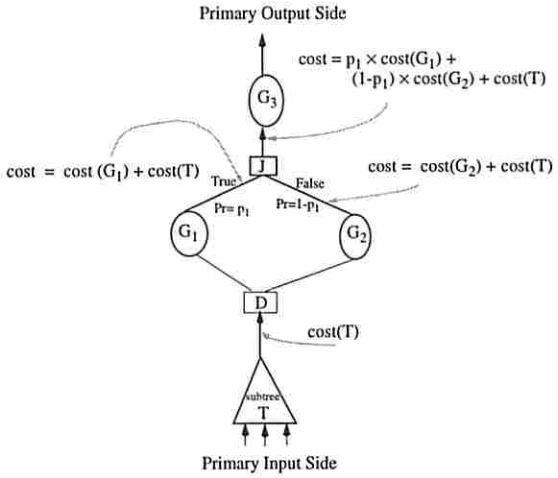


Figure 4.4: Cost calculation in a DFG with a conditional branch

circuit outputs, but not to alter the module selection.)

General DFG's contain *conditional branches.* We use nodes D and J to indicate the distribute and join nodes in order to express the conditional branches. For each D and J pairs (which really serve as *synchronization points*), there are two subgraphs which represent the 'true' and 'false' conditions, respectively. We treat the two subgraphs as if they are two simultaneous (parallel) subgraphs and apply the dynamic programming technique except for the following. During the post-order traversal, when we come to a D node, we need not divide the cost of the subgraph rooted at D by two (in case of a single branch). Furthermore, when we come to a J node, we must weight the cost of each branch by the probability that the branch is taken and only then add the weighted branch costs to obtained the cost of a J node. Note that J and D are dummy nodes and hence they, by themselves, do not contribute any additional cost (cf. Fig. 4.4).

4.4 Complexity Analysis

We introduce the definition of *pseudo-polynomial* complexity, which is taken from [GaJo79].

Definition 4.1 *Let $\mathcal{I}$ be an instance of a computational problem–typically $\mathcal{I}$ will be a sequence of combinatorial objects such as graphs, sets, or integers. Then $\mid \mathcal{I} \mid$ is the problem size and Max($\mathcal{I}$) is the largest integer appearing in $\mathcal{I}$.*

Definition 4.2 *An algorithm $\mathcal{B}$ for a problem Π is pseudo-polynomial if it solves any instance $\mathcal{I}$ of Π in time bounded by a polynomial in $\mid \mathcal{I} \mid$ and Max($\mathcal{I}$).*

Let's scale delay values for all modules under different voltage assignments to become integers. Furthermore, let's denotes the maximum computation time for a tree-like DFG (using the worst-case integer delay values on any path) by T_{max} and assume that T_{max} is bounded from above by an integer M. Let $\mid \mathcal{I} \mid = n$ where n is the number of nodes in the DFG.

The MVS problem Π is a *number problem* because there exists no polynomial p such that M is less than or equal to $p(n)$. This implies that we can develop an algorithm for solving Π with a pseudo-polynomial time complexity (Π is *not* NP-complete in the strong sense).

Theorem 4.1 *Our dynamic programming algorithm provides a pseudo-polynomial time algorithm for solving the MVS problem.*

Proof: The number of energy-delay points on each node in the DFG is bounded from above by M. The algorithm thus has a time complexity of $n \cdot M$. Delay function merging and adding can be done in polynomial time in the number of points on the curves involved in the operations. Therefore, our algorithm for solving the MVS problem runs in pseudo-polynomial time because its time complexity is bounded above by a polynomial function of n and M as defined above. □

Theorem 4.2 *If the tree is node-balanced (its height is logarithmic in the number of its leaf nodes), then our dynamic programming algorithm runs in polynomial time.*

Proof: The maximum number of points on any delay curve in the tree is bounded from above by m^l, where l is the number of the levels in the tree, and m is the maximum number of module

alternatives in the library which match some node in the DFG. In this case, $l = log_2 n$, thus the number of points is bounded by $n^{log_2 m}$. □

4.5 Module sharing after scheduling

Dividing a problem into two subproblems and solving each subproblem optimally still produces a suboptimal solution to the original problem. This is however necessary in many domains because of the complexity of the problems encountered in practice. Examples include separating scheduling from module allocation in behavioral synthesis, separating logic restructuring from logic minimization, etc. In this case, we have also divided the MVS problem into an initial voltage assignment phase followed by module sharing optimization phase.

After scheduling is completed, a module allocation and binding algorithm is applied whose goal is to exploit the possibility for sharing modules among compatible operations. This algorithm uses conventional techniques to detect operation compatibility and mutual exclusiveness of operations (as in parallel branches). [1] It is difficult to account for the possibility of module sharing during dynamic programming as at any point during the post-order tree traversal we have only partial information about the operations that have been assigned to modules and cannot modify the dynamic programming cost function to reflect the sharing potential. In another word, an attempt to consider sharing during the module assignment and scheduling phase will violate the principle of optimality that is the basis for using dynamic programming. This is because the dynamic programming cost at the root of a subtree cannot be determined independently of the rest of the tree (which is not yet mapped), so the optimal solution cannot be obtained by merging optimal solutions for the corresponding subproblems. We illustrate this difficulty with an example (cf. Fig. 4.5). When we try to calculate the energy cost of node C, we have to consider not only the cost of node C when it is mapped to a module, but

[1] Two operations are said to be compatible if they can be executed by the same module with the same supply voltage level and hence delay if their lifetimes do not overlap. Two operations are said to be mutually exclusive if they cannot not be alive at the same time in a DFG with conditional branches.

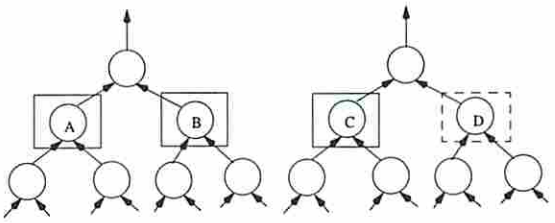


Figure 4.5: Module sharing during post-order traversal in dynamic programming

also the possibility that nodes A and B which have been already assigned and node D which has not been traversed yet, might share the same module with node C. Furthermore, before the scheduling is completed, the exact time-span of some operations (nodes) is not known, This information is however required to determine module sharing solution. Because of these difficulties, we estimate the switching activity at the inputs of a module during the dynamic programming phase assuming that the module is not shared.

The potential for module sharing is clearly lower in the multiple voltage DFG compared to the single-voltage case. However, we still observe a sizeable reduction in total module area as a result of this post-processing step. We use a scheme similar to that of [ChPe95a] for minimum energy module binding using a max-cost network flow algorithm. Details can be found in [ChPe95b].

Chapter 5

Functionally Pipelined Data-path

5.1 Background

In a functionally pipelined design, several instances of the execution of a data flow graph are overlapped in time. The time domain is discretized into *time steps* (for a given length of a time step). Unlike a structural pipelining, there is no physical (but logical) stages in a functional pipeline. Structural pipelining implies the use of pipelined modules, such as 4-stage pipelined multiplier. Both functional and structural pipelining are aimed to increase the throughput of computation. *Latency* L is defined as the number of time steps between two consecutive pipeline initiations. A *control step* or c-step is a group of time steps that overlap in time (cf. Fig. 5.2). For a given latency L, c-step i corresponds to time steps $i + (m \cdot L)$, where m is an integer. We denote the L consecutive c-steps in a pipeline initiation as a *frame*. When the supply voltage level of a module is lowered, its delay increases and the operation assigned to the module may become *multi-cycle*. If the voltage is further lowered, for a small pipeline initiation latency L, an operation may become *multi-frame*.

The *computation time* $\mathbf{T_{comp}}$ of a functionally pipelined data-path is defined as the total time needed to process one data sample. Normally, a functionally pipelined circuit has to meet some throughput and/or computation time constraints. Throughput constraint is often more important than the computation time constraint in a functionally pipelined design.

Suppose we are given N input samples to be processed by a functionally pipelined data-path. Let T_{comp} be the computation time and t_c be the length of a c-step. Then total time used is equal to $(\mathbf{N-1}) \cdot \mathbf{L} \cdot \mathbf{t_c} + T_{comp}$. Let $\mathbf{E_{LS}}$ be the energy used by all of the level shifters in the circuit *per pipeline initiation* (or the energy used to process only one data sample) and $\mathbf{E_{FU}}$ be the average energy used by all of the modules per pipeline initiation. Then total energy used is $N \cdot (E_{FU} + E_{LS})$ and

$$\textbf{average power} = \frac{\mathbf{N} \cdot (\mathbf{E_{FU}} + \mathbf{E_{LS}})}{(\mathbf{N-1}) \cdot \mathbf{L} \cdot \mathbf{t_c} + \mathbf{T_{comp}}} \approx \frac{(\mathbf{E_{FU}} + \mathbf{E_{LS}})}{\mathbf{L} \cdot \mathbf{t_c}} \tag{5.1}$$

In our problem, the latency, L and t_c are assumed to be given. Therefore, when we minimize $(\mathbf{E_{FU}}+\mathbf{E_{LS}})$, which is the average total *energy* used by all modules and level shifters per pipeline initiation, we are indeed minimizing the average *power* dissipation.

An algorithm for performing scheduling and allocation for functionally pipelined DFG's is described in [PaPa88]. This technique known as the *feasible scheduling* deals with single cycle operations and operations that can be chained together in one c-step, but not multi-cycle or multi-frame operations. The idea is to build a *resource allocation table* where columns correspond to c-steps, and rows correspond to module instances and entry (i, j) of the table denotes the assignment of operation(s) to module instance i at c-step j. In functional pipelining, two operations that use the same module are said to be non-overlapping if their life spans (in terms of the c-steps in which they are alive) are not overlapping or that they are mutually exclusive operations in a conditional DFG. The feasible scheduling algorithm [PaPa88] is basically a modified *list scheduling* with two main ingredients: urgency priority and allocation table. Forward/backward urgency of an operation is the longest path delay from an operation to the primary outputs/inputs. The main flow of list scheduling is preserved while the urgency priority is used to sort the list of operations and the resource allocation table is used to check for resource conflicts.

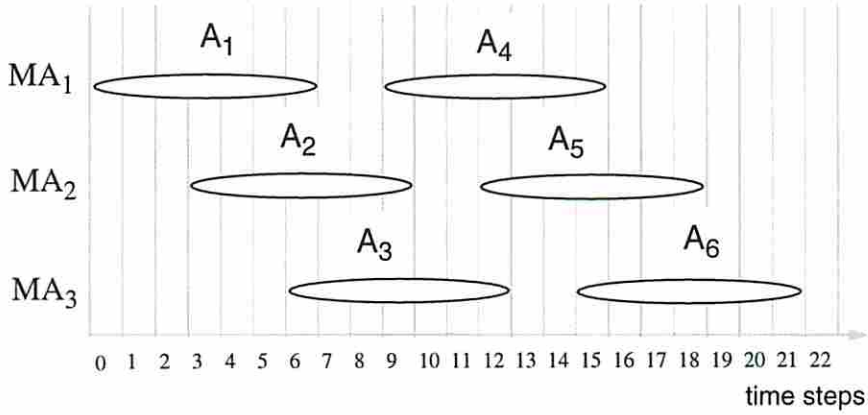


Figure 5.1: Example to Show the a Revolving Schedule on 3 module $MA's$, for the module delay = $7t_c$ and pipeline latency, $L = 3(t_c)$. Note that A_i is the execution of operation A in pipeline initiation i, and c-step 1 = time steps {1,4,7,...}, c-step 2 = time steps {2,5,8, ...}.

5.2 Handling multi-frame operations

Our goal is to obtain a minimum energy functionally pipelined data-path realization while meeting the global throughput constraint (which is described by two parameters t_c and L). Suppose there is a module MA with delay equal to $k \cdot t_c$, where $\frac{k}{L} > 1$, which is capable of performing an operation A in the DFG. To sustain the initiation rate of one data sample per $L \cdot t_c$, we use $\lceil \frac{k}{L} \rceil$ modules for operation A and use **a revolving schedule** as described next.

Suppose that we have modules $MA_1, MA_2 \ldots MA_{\lceil \frac{k}{L} \rceil}$ for operation A in the DFG. Our revolving schedule assigns operation A on the $m \cdot \lceil \frac{k}{L} \rceil + 1st$ data sample (pipeline initiation) to module MA_1 at *time step* $L \cdot (m \cdot \lceil \frac{k}{L} \rceil)$, assigns operation A on the $m \cdot \lceil \frac{k}{L} \rceil + 2nd$ data sample to module MA_2 at *time step* $L \cdot (m \cdot \lceil \frac{k}{L} \rceil + 1)$, etc., where $m = 0, 1, 2, \ldots$. Fig. 5.1 illustrates the scheduling result for $k = 7$ and $L = 3$.

Theorem 5.1 *The* revolving scheduling *algorithm assigns the operation whose corresponding module delay is $k \cdot t_c$, where $\frac{k}{L} > 1$ to $\lceil \frac{k}{L} \rceil$ modules without creating any resource conflict while meeting the throughput constraint of $\frac{1}{L}$ where L is the latency of the functional pipeline.*

Proof: For any positive number x and L, we have $x \leq \lceil x \rceil$. Then we know that $\frac{k}{L} \leq \lceil \frac{k}{L} \rceil$,

and thus $k \leq L \cdot \lceil \frac{k}{L} \rceil$. $\Rightarrow L \cdot m \cdot \lceil \frac{k}{L} \rceil + k + L(r-1) \leq L \cdot (m+1) \cdot \lceil \frac{k}{L} \rceil + L(r-1) \Rightarrow k + L \cdot (m \cdot \lceil \frac{k}{L} \rceil + r - 1) \leq L \cdot ((m+1) \cdot \lceil \frac{k}{L} \rceil + r - 1)$. This shows the m-th operation scheduled on module MA_r finishes before the $(m+1)$-th operation scheduled on that module begin, $\forall r, 1 \leq r \leq \lceil \frac{k}{L} \rceil$. Therefore the revolving schedule schedules the multi-frame operations safely. □

In the following, we prove that the revolving schedule is the best possible schedule in terms of the number of the module instances used.

Theorem 5.2 *For any module with delay $k \cdot t_c$, where $\frac{k}{L} > 1$, $\lceil \frac{k}{L} \rceil$ is the theoretical lower bound on the number of modules that have to be utilized in order to perform the corresponding operation with the pipeline latency of L without creating any resource conflict.*

Proof: Consider an operation that has a time-span of $[0, k]$. Multiple instances of this operation in different pipeline initiations with delay $k \cdot t_c$ can be described as a family of intervals $[-sL, k - sL]$, where s is an integer. For an observation window $[0, L]$, the minimum number of modules needed is the number of elements of this family of intervals $[-sL, k - sL]$ that intersect the observation window $[0, L]$. Obviously, the last interval that intersects this window is the interval when $s = 0$, that is $[0, k]$. The first interval that intersects this window is the interval $[-uL, k - uL]$ where u is the largest integer satisfying $k - uL > 0$. The total number of the intervals that overlap with the window $[0, L]$ is $N_{min} = u - 0 + 1 = u + 1$. If $\frac{k}{L}$ is an integer t, then $u = t - 1$, otherwise, if $\frac{k}{L}$ is not an integer, $u = \lfloor \frac{k}{L} \rfloor$. In both case, N_{min} can be represented as $\lceil \frac{k}{L} \rceil$. This gives the minimal number of modules required to avoid resource conflict. □

We next discuss how the dynamic programming approach has to be modified for the functionally pipelined designs. We consider three cases.

1) Operation delay $k \cdot t_c$ is larger than $L \cdot t_c$. As shown before, here we have no choice but to use $\lceil \frac{k}{L} \rceil$ modules to perform the operation without creating any resource conflict while meeting

the global throughput constraint. Recall that each module is *active* only when it is performing an operation, otherwise, it is in the *sleep mode*. In any time interval, given t_c and L, the total number of operations is the same regardless of the number of modules used to execute those operations. Consequently, the total energy consumption for processing N data samples can be calculated as follows. It is true that the temporal characteristics of input data changes when we replicate one module into $\lceil \frac{k}{L} \rceil$ modules so as to satisfy the throughput constraint. For example, let the input vectors feeding to a module MA be denoted by V_1, V_2, V_3, V_4, V_5, V_6, etc.. Suppose the corresponding operation becomes multi-frame and thus we need to duplicate the module to MA_1 and MA_2. The input sequence feeding to MA_1 is V_1, V_3, V_5, V_7, ... whereas that feeding to MA_2 is now V_2, V_4, V_6, V_8, ... Obviously, the input activities for MA_1 and MA_2 are different from that of MA. However, the activities for MA_1 and MA_2 can still be calculated based on behavioral simulation results (as long as we know how the data is multiplexed to either MA_1 or MA_2. This is known before dynamic programming step based on the delay of MA and L. The sequence of operands fed to module MA_1 and MA_2 can be obtained by sampling the original sequence of operands feeding to operation A at the sampling period of 2.). Next, the energy dissipation of module MA averaged over one time frame is calculated as the *arithmetic mean* of the energy dissipations of MA_1 and MA_2 under their respective input sequences. This is obviously valid only if we guarantee to shut off MA_1 or MA_2 when they are not in use. Note, that although the energy dissipation of module MA after functional pipelining may change, we are still able to precisely calculate this change. Hence our energy model remains data-sensitive. The area for module MA however increases by a factor of $\lceil \frac{k}{L} \rceil$.

In Fig. 5.2 we show a very simple DFG with three operations A, B and C. Suppose operations A, B and C were assigned voltages during the scheduling step and the resulting delays for tentative modules MA, MB, and MC are , $7{\cdot}t_c$, $5{\cdot}t_c$, and $3{\cdot}t_c$, respectively. Fig. 5.2 shows the result after using the revolving scheduling. A_1 represents operation A performed on

the pipeline initiation 1, etc.

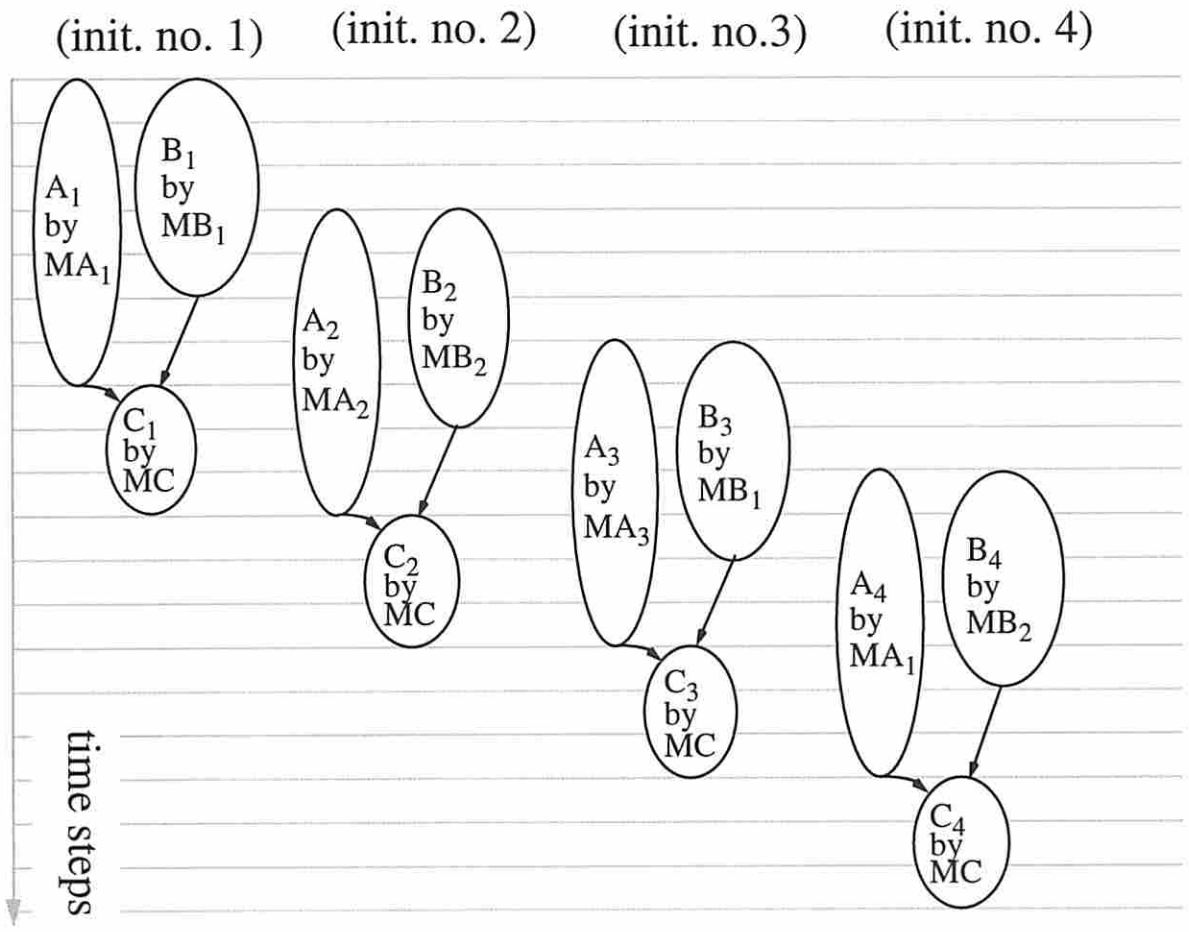
2) Operation delay $k \cdot t_c = L \cdot t_c$. We need to use exactly one module to perform the operation. No other operation can share the module. The energy cost of the operation is that of the corresponding module per data value.

3) Operation delay $k \cdot t_c < L \cdot t_c$. We use one module per operation, however, the module may be shared with other operations. We again relegate the sharing issue to a post-processing phase where the scheduling solution obtained by dynamic programming approach is further modified to increase module sharing (thus reducing area cost of the design).

5.3 Module sharing after scheduling

Our goal is to minimize the resources after the scheduling has been done. The problem can be formulated as a minimal coloring of a circular arc graph [Golu80]. (For a functionally pipelined data-path, a row in the resource allocation table is a track which is circular in nature, i.e. the Lth c-step in the current frame comes before the first c-step of next frame). The exact solution is obtained by the algorithm proposed in [Stok91] which solves the register allocation problem in cyclic data flow graphs by using a multi-commodity flow formulation. Instead, we have adopted a less expensive heuristic for doing module sharing as described next.

Recall the resource allocation table defined in Chapter 5.1. To check the resource conflicts in a functionally pipelined data-path, the interval of time steps of an operation is written as the interval of corresponding c-steps (that is, a time step i will be in c-step i modulo L, where L is the latency of the functional pipeline). The resource allocation table is similar to the one used in [PaPa88], with one important difference. The table used in [PaPa88] only contains modules corresponding to single cycle operations; our table has extended this table to deal with multi-cycle operations. The following describe the outline of our heuristic for module sharing after scheduling. Our resource allocation table has no modules of any types initially. After we scan through the initial schedule produced by the dynamic programming, we know



latency L = 3(t_c) , delay(MA) = 7 t_c , delay(MB)=5t_c

delay(MC) = 3 t_c, Use 3 MA's , 2 MB's , 1 MC

Figure 5.2: 4 pipeline initiations and the corresponding revolving schedule on multiple modules instances of corresponding operations

exactly the time-step spans of all operations. We assign one operation (say x) at a time to the table by scanning through the existing rows of the resource allocation table. If there is a row corresponding to module that has been assigned to some other operation y which is compatible with x, then we assign x to the module and modify the allocation table to reflect the new assignment. Mutually exclusive operations can share the same module even if their time-spans (in terms of c-steps) overlaps. A special coloring algorithm in [PaPa88] can be used to detect the mutual exclusiveness of operations in a given DFG with conditional branches. If there is no row that can be used for operation x, we then create a new row (thus a new module) for the current operation. The process ends after we have assigned all of the operations. This is similar to applying the left edge algorithm [KuAl87] to a set of circular tracks.

5.4 Controllable parameters

Our framework is flexible and the user may modify a number of parameters to achieve different objectives.

First, the number of voltages used in the final design are controllable by the users. The user can limit his module library to contain only certain voltages. For example, if the user limits the available voltages to 5 and 3.3 volts, then the final design produced by our algorithm will use only these two voltages. In addition, only a subset of the available supply voltages can be used for certain types of operations. This is because the voltages corresponding to points on the curves for each operation type determine the possible voltages that may be used for that operation in the final design. Different points on the the energy-curve of an operation may also correspond to the same voltage but with a different architecture (such as carry lookahead adder vs. serial adder).

Second, T_{comp} is user controllable. This is especially important for the case of functionally pipelined data-path. By increasing T_{comp}, our algorithm can find a solution which consumes less energy per pipeline initiation. The throughput of the functional pipeline remains the same

as L^{-1} and is not changed by T_{comp}. In the case of non-pipelined data-path, increasing T_{comp} can reduce the total energy used. By using $T_{comp} > T_{crit}$ (T_{crit} is the length of the critical path when all operations use the highest voltage), all operations become "non-critical" and hence can be realized at lower supply voltage levels. This is however achieved at the cost of degrading the performance of the non-pipelined design.

Chapter 6

Experimental Results

We first present the result obtained by our algorithm on a small seven node DAG (not a tree) and the result obtained by exhaustive search. We assume four voltages are available and that all primary inputs carry 5-V signals. The module library is shown in Table 6.1. The energy consumed by the level shifters is shown in Table 3.4. In this example, the length of a c-step is 30 (ns) and a total computation time constraint $T_{comp} = 700$ (ns). The results of dynamic programming algorithm and exhaustive search are shown in Fig. 6.1. Note our new method can handle a very large graph (more than thousands of nodes) in seconds, but the exhaustive search (and the ILP formulation) which can be used to obtain the true optimal solution can only handle a small example ($\leq$ 20 nodes) in a reasonable amount of time. The two solutions obtained are different, but the results show that our solution which is only 1% away from the optimal solution.

	5.0V		3.3V		2.4V		1.5V	
	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)
Module	delay	Energy	delay	Energy	delay	Energy	delay	Energy
mult16	100	2504	175.20	1090.7	286.80	576.9	717.03	225.3
add16	20	118	35.05	51.4	57.36	27.2	143.40	10.6
sub16	20	118	35.05	51.4	57.36	27.2	143.40	10.6

Table 6.1: Library of Module to be used in a Small Example. at $\alpha_1^{\mathbf{FU}} = \alpha_2^{\mathbf{FU}} = \mathbf{0.5}$

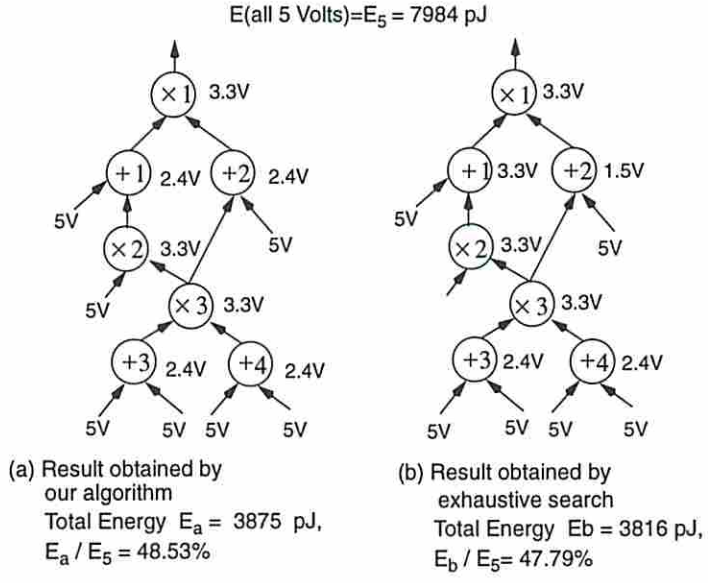


Figure 6.1: A Small Example

We next present another small example to compare our algorithm with the one in [RaSa95]. We can do this by using an unrealistic example where all adders and multipliers have the same energy vs. delay curve. It is not very meaningful to do so, since multiplier takes more energy and delay to do the multiplication than the adder does to do the addition. However, we will still do the comparison.

Their algorithm produces design that uses variable voltages (might be very irregular), instead of using voltages from a fixed set of user specified voltages. Furthermore, their algorithm can produce a result that uses up to 5 voltages for a small tree like DFG (which contains only 11 nodes) as shown in Fig. 6.2(a) (their result for a total computation time constraint of 125 ns is shown in Fig. 6.2(b)). Note that in Fig. 6.2(b), V_i is the voltage that their algorithm used when the delay of the adder equal to $i \cdot t_c$ (cf. Fig. 6.2(c))from the voltage vs. delay curve.

Our result for the same DFG and same T_{comp} using the library shown in Table 6.2, is depicted in Fig. 6.2(d). Although we are restricted to use voltages from the set {5, 3.3,

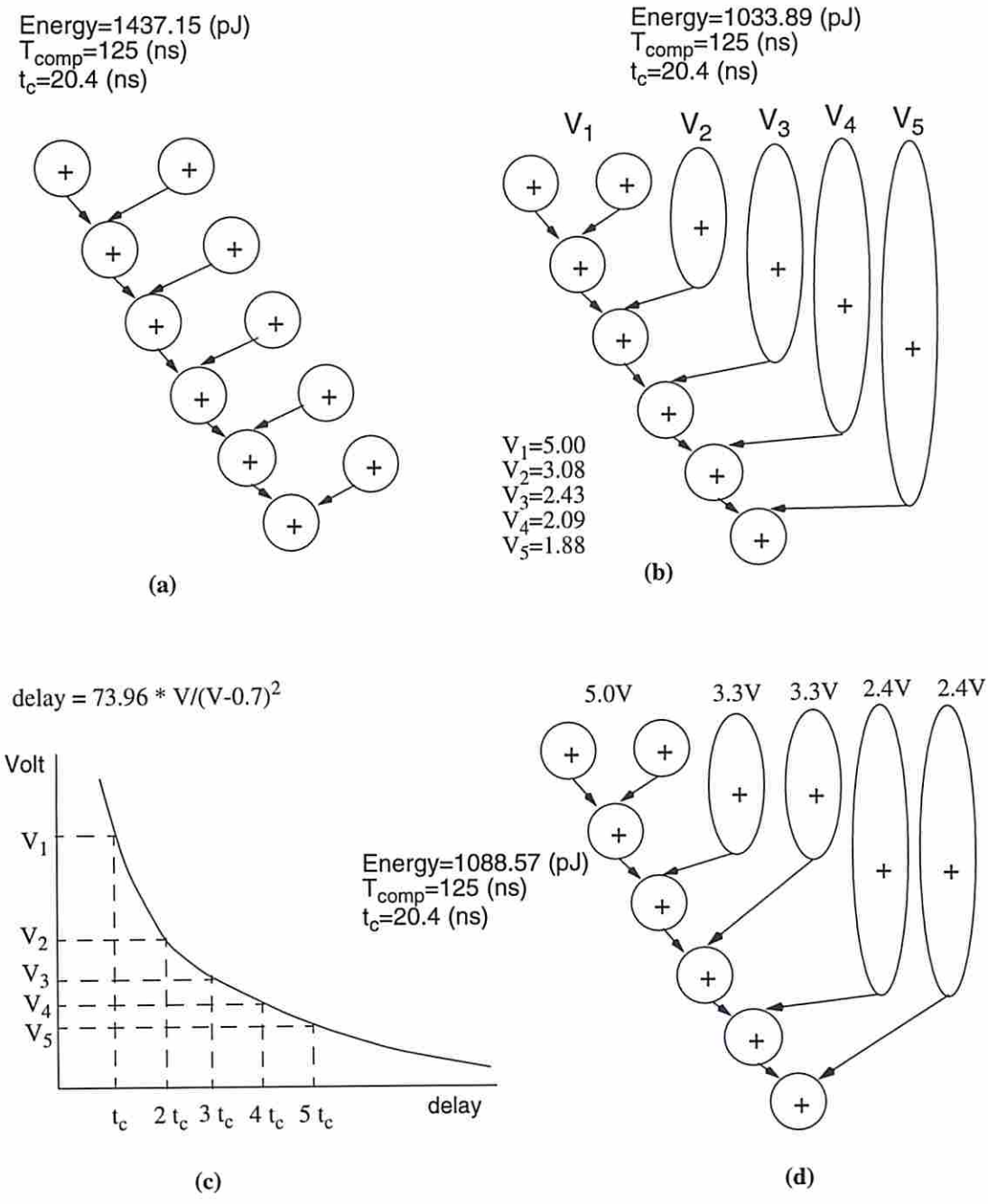


Figure 6.2: Another small example to compare our algorithm with the one found in [RaSa95]

	5.0V				3.3V				2.4V		1.5V	
	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)
Module	delay	Energy	delay	Energy	delay	Energy	delay	Energy	delay	Energy	delay	Energy
add16	20.4	130.65	-	-	36.14	56.91	48.31	61.40	60.27	30.10	149.75	11.76

Table 6.2: Module Energy (in pJ) under a pseudo-random white noise data model at $\alpha_1^{\mathbf{FU}} = \alpha_2^{\mathbf{FU}} = \mathbf{0.5}$

2.4, 1.5}, our solution has about the same energy dissipation as the solution given by Raje's algorithm.

In the remainder of this section, we present detailed results of our algorithm on a number of standard benchmarks including a Test DFG, AR Filter, Elliptical Wave Filter[GeEl92], Discrete Cosine Transform, Robotic Arm Controller, 2nd-order Adaptive Transversal Filter [Hayk91] and Differential Equation Solver [CaWo91].

We use the table look-up method presented in Section 3.2 for energy calculation. Our module library is shown in Table 6.3. For the sake of giving energy behavior of our library modules, the energy values in this table are reported for α_1=α_2=0.5, but as shown in Section 3.2, we calculate the energy values for any other α_1, α_2 pairs as they become necessary.

Note that this table, for example, shows that we have 5 $mult16$ implementations, two at 5.0 volts and three at lower supply voltages. Difference between the two which operate operated at $5V$ is their architectures (parallel multiplier vs. Wallace tree multiplier). This is obviously an example library; Any other library with different module architectures, and different voltage assignments can be used instead.

Our experimental results are shown in Table 6.4. In this table, E^1 is energy dissipation corresponding to the supply voltage of 5 volt. E^2, E^3 and E^4 are the average energy obtained when the libraries contain modules of $\{5V, 3.3V\}$, $\{5V, 3.3V, 2.4V\}$ and $\{5V, 3.3V, 2.4V, 1.5V\}$, respectively. The columns corresponding to $\frac{E_{LS}^i}{E^i}$ are the percentage of energy consumed in level shifters over the total energy. The results show that although the power consumed

in level shifters is not negligible, it is not large either. Note that we can delete level shifters for step-down voltage conversions as described in [UsHo95]. In our experiments, however we inserted the level shifters for both step-up and step-down conversions.

Table 6.4 shows that an average energy saving of 2.76%, 38.01% and 44.58% is achieved when using 2 supply voltage levels with total computation time (T_{comp}) set to T_{crit} (the longest path delay in the DFG), $1.5T_{crit}$ and $2T_{crit}$. Similarly, an average energy saving of 3.88%, 40.19% and 64.8% (or 3.89%, 40.48% and 65.68%) is achieved when using 3 (or 4) supply voltage levels, respectively with total computation time set to T_{crit}, $1.5T_{crit}$ and $2T_{crit}$. These results are depicted graphically in Fig. 6.3.

Energy saving for the case of $T_{comp} = T_{crit}$ is very much circuit-dependent. That is, the energy saving is higher in circuits where the number of non-critical nodes is large. For the AR filter circuit, $\frac{E^4}{E^1}$ ratio is as low as 0.85 while for the $FDCT$ circuit, this ratio is 1. Energy saving potential increases substantially when $T_{comp} > T_{crit}$. For example, $\frac{E^4}{E^1}$ ratio for $T_{comp} = 1.5T_{crit}$ goes down to 0.61 and 0.57 for the AR filter and $FDCT$ circuits, respectively. Also, note that the energy dissipation continues to drop significantly when we go from 2 to 3 voltage levels, but not for 4 voltage levels.

In all benchmarks that we attempted, the single lowest supply voltage that met T_{crit}, $1.5T_{crit}$ was 5 volts. However, when we set the T_{comp} as $2.0T_{crit}$, then this voltage dropped to 3.3 volts. As a result, when we examine $\frac{E^i}{E^1}$, for $i = 2$, 3 and 4, we see that the ratio decreases from T_{crit} to $1.5T_{crit}$, but then rises for $2.0T_{crit}$. This is due to a shift in the E^1 voltage from $5V$ to $3.3V$. (In fact, E^i always decreases when T_{comp} increases).

In the functionally pipelined case, we can achieve lower average energy for a given throughput constraint (which is described by two parameters t_c and L) by using a longer computation time because larger T_{comp} will result in a solution that uses lower voltages and thereby lower average energy. However, this causes more operations to become multi-cycle or multi-frame operations which will increase the number of modules used to achieve the same throughput

	5.0V				3.3V				2.4V		1.5V	
	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)	(ns)	(pJ)
Module	delay	Energy	delay	Energy	delay	Energy	delay	Energy	delay	Energy	delay	Energy
mult16	103.7	16829.52	132.0	13265	181.20	7330.93	-	-	295.43	3877.52	721.15	1514.65
add16	20.4	130.65	-	-	36.14	56.91	48.31	61.40	60.27	30.10	149.75	11.76
sub16	20.4	130.65	-	-	36.14	56.91	-	-	60.27	30.10	149.75	11.76

Table 6.3: Module Energy (in pJ) under a pseudo-random white noise data model at $\alpha_1^{\mathbf{FU}} = \alpha_2^{\mathbf{FU}} = \mathbf{0.5}$

constraint. Thus the computation time constraint indirectly controls the chip area.

Another set of experimental results are shown in Table 6.5. In this table, E^1 is energy dissipation corresponding to the supply voltage of 3.3 volt. From Table 6.5 , we can see that, using smaller t_c (i.e. 20 ns) in general results in lower energy dissipation. The reason is that multi-cycle operations start at multiples of a of time step. Large t_c increases the *"dead time"* (time interval between the end of operation and the beginning of next time step) of a multi-cycle operation. Larger t_c thus tends to require higher voltages to meet the total computation time constraint. These results are depicted graphically in Fig. 6.4.

Bench-mark	T_{comp} (ns)	(pJ) E^1	volt. used	% $\frac{E^1_{LS}}{E^1}$	(pJ) E^2	% $\frac{E^2_{LS}}{E^2}$	(pJ) E^3	% $\frac{E^3_{LS}}{E^3}$	(pJ) E^4	% $\frac{E^4_{LS}}{E^4}$	% $\frac{E^2}{E^1}$	% $\frac{E^3}{E^1}$	% $\frac{E^4}{E^1}$
Test DFG	321†	33985	5.0	0	33985	0	33985	0	33985	0	100	100	100
	481	26856	5.0	0	20937	0.5	20942	0.6	20937	0.5	77.96	77.98	77.96
	642	26856	5.0	0	14791	0.2	11532	1.4	11532	1.4	55.07	42.94	42.94
AR Filter	510†	256578	5.0	0	233029	0.1	219131	0.1	219131	0.1	90.82	85.40	85.40
	765	213804	5.0	0	143112	0.6	129214	0.6	129214	0.6	66.94	60.43	60.43
	1020	213804	5.0	0	118410	0.5	78073	1.5	68517	1.6	55.38	36.52	32.06
EWF	690†	130747	5.0	0	130920	0.3	130933	0.3	130933	0.3	100.1	100.1	100.1
	1035	109360	5.0	0	62434	2.5	60728	1.9	60592	1.7	57.09	55.53	55.40
	1380	109360	5.0	0	60007	0.6	34031	4.1	34517	4.6	54.87	31.12	31.56
FDCT	240†	102738	5.0	0	102738	0	102738	0	102738	0	100	100	100
	360	81351	5.0	0	46018	1.3	46018	1.3	46018	1.3	56.56	56.56	56.56
	480	81351	5.0	0	44982	0.9	25150	3.2	25150	3.2	55.29	30.92	30.92
Robot Ctrl.	650†	297627	5.0	0	286000	0.1	278995	0.1	278982	0.1	96.09	93.74	93.73
	975	240594	5.0	0	133909	0.3	127061	0.5	122831	0.6	55.66	52.81	51.05
	1300	240594	5.0	0	133909	0.3	85650	0.7	80929	0.8	55.66	35.60	33.64
2nd ATF	180†	74106	5.0	0	74140	0.2	74113	0.2	74068	0.1	100.0	100	99.95
	270	66977	5.0	0	37708	1.8	37681	1.7	37635	1.7	56.30	56.26	56.19
	360	66977	5.0	0	37459	1.6	20455	3.4	20410	3.3	55.93	30.54	30.47
Diff Eq.	300†	94500	5.0	0	88560	0.3	88507	0.3	88443	0.2	93.71	93.65	93.59
	450	80242	5.0	0	50985	1.5	47451	1.5	47363	1.4	63.54	59.13	59.02
	600	80242	5.0	0	44711	1.1	31086	2.1	30998	1.9	55.72	38.74	38.63
Avg.	T_{cr}	-	-	0	-	0.1	-	0.1	-	0.1	97.24	96.12	96.11
	1.5 T_{cr}	-	-	0	-	1.2	-	1.2	-	1.1	61.99	59.81	59.52
	$2T_{cr}$	-	-	0	-	0.7	-	2.3	-	2.4	55.42	35.20	34.32

Table 6.4: Experimental Results on Various Benchmarks. Note, that E^1 is energy dissipation corresponding to the supply voltage of 5 volts. E^2, E^3 and E^4 are the average energy obtained when the libraries contain modules of $\{5V, 3.3V\}$, $\{5V, 3.3V, 2.4V\}$ and $\{5V, 3.3V, 2.4V, 1.5V\}$, respectively. †: Corresponds to the critical path delay of the DFG. In this table, t_c= 30 ns and $L = 3$.

Bench-mark	T_{comp} (ns)	(ns) t_c	(pJ) E^1	volt. used	% $\frac{E^1_{LS}}{E^1}$	(pJ) E^2	% $\frac{E^2_{LS}}{E^2}$	(pJ) E^3	% $\frac{E^3_{LS}}{E^3}$	% $\frac{E^2}{E^1}$	% $\frac{E^3}{E^1}$
test DFG	642.0	20	14791	3.3	0.18	14791	0.18	11399	0.66	100	77.06
	642.0	30	14791	3.3	0.18	14791	0.18	11532	1.39	100	77.96
	642.0	40	14791	3.3	0.18	14791	0.18	11433	0.72	100	77.29
AR Filter	1020.0	20	118410	3.3	0.47	118410	0.47	77362	1.18	100	65.33
	1020.0	30	118410	3.3	0.47	118410	0.47	78073	1.51	100	65.93
	1020.0	40	118410	3.3	0.47	118410	0.47	90779	0.65	100	76.66
EWF	1380.0	20	60007	3.3	0.55	60007	0.55	33241	3.14	100	55.39
	1380.0	30	60007	3.3	0.55	60007	0.55	34031	4.05	100	56.71
	1380.0	40	60042	3.3	0.55	60042	0.55	39933	2.00	100	66.50
FDCT	480.0	20	44982	3.3	0.85	44982	0.85	24784	3.17	100	55.09
	480.0	30	44982	3.3	0.85	44982	0.85	25150	3.2	100	55.91
	480.0	40	44997	3.3	0.84	44997	0.84	24814	3.16	100	55.14
RobotC	1300.0	20	134127	3.3	1.07	133909	0.34	85650	0.72	99.83	63.85
	1300.0	30	134127	3.3	1.07	133909	0.34	99464	0.62	99.83	74.15
	1300.0	40	134127	3.3	1.07	134157	1.07	113625	1.45	100.0	84.71
2nd ATF	360.0	20	37459	3.3	1.59	37459	1.59	27144	2.41	100	72.46
	360.0	30	37459	3.3	1.59	37459	1.59	20455	3.39	100	54.60
	360.0	40	37459	3.3	1.59	37459	1.59	27174	2.41	100	72.54
Diff-Eq	600.0	20	44711	3.3	1.06	44711	1.06	30889	1.67	100	69.08
	600.0	30	44711	3.3	1.06	44711	1.06	31086	2.05	100	69.52
	600.0	40	44711	3.3	1.06	44711	1.06	30889	1.67	100	69.08

Table 6.5: This table shows the energy consumption vs. t_c under $T_{comp} = 2T_{crit}$ on various benchmarks. In this table, E^1 is energy dissipation corresponding to the supply voltage of 3.3 volts. E^4 column is not shown since results are similar to that of E^3

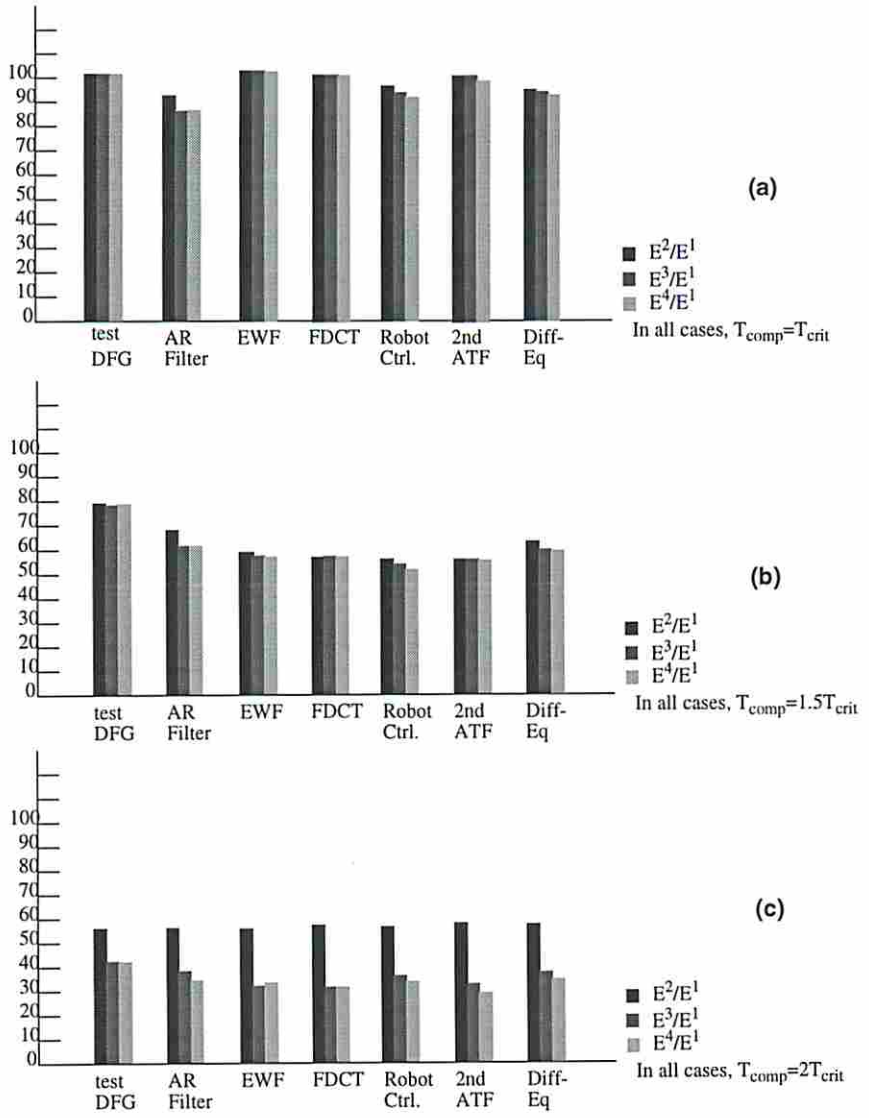


Figure 6.3: Experimental results

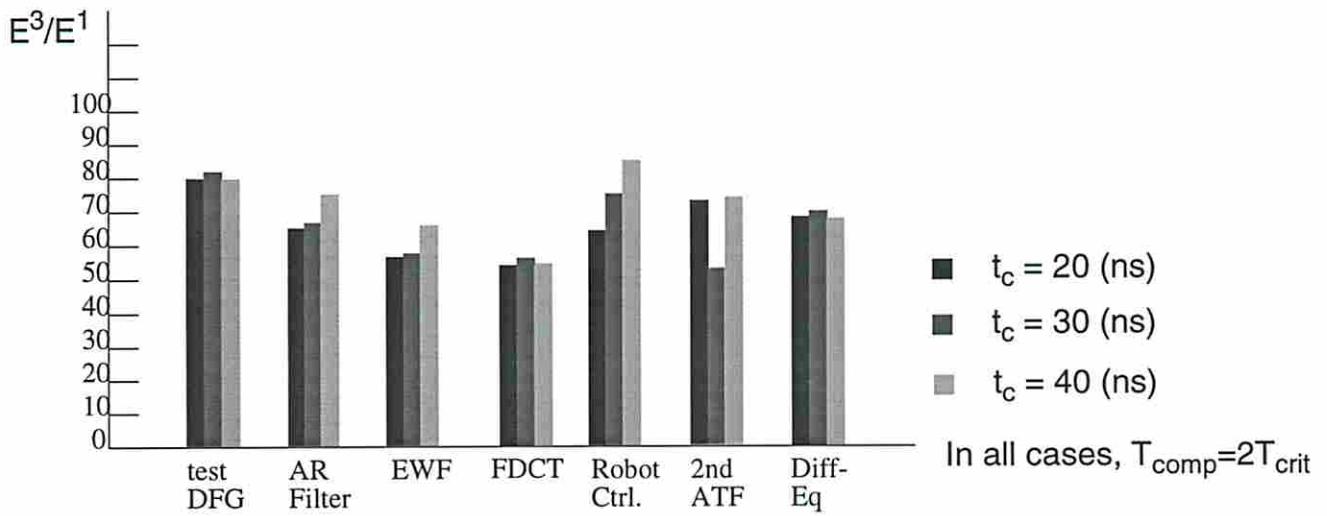


Figure 6.4: Experimental results

Chapter 7

Conclusion

We presented a dynamic programming approach for assigning voltage levels to the modules in non-pipelining and functionally pipelined data-paths. The average power consumption can be reduced by using a single lowered supply voltage. If the computation time constraint is violated with only a single lower supply voltage, then pipelining or parallelism on whole or part of the circuit to recover performance has to be used. Although this is one way of trading the chip area for power, the area penalty is generally much higher. With a given computation time constraint, when multiple voltages are used, our algorithm will lower the supply voltages of operations which are not on the critical path while keeping the supply voltages of operations on the critical path at a maximum. The computation time constraint is thus achieved at lower area overhead.

The use of $\lceil \frac{k}{L} \rceil$ modules for a multi-frame operation was necessary to maintain the throughput while reducing the average energy consumption in the data-path. This, however, increases the controller and multiplexor cost. The multiplexor cost can be easily obtained by *"modulating"* the energy results reported in Table 3.1. An energy cost model for controller can be developed as a function of the total number of functional units used in the circuit. For example, by assuming that the energy cost of the controller scales with the log_2 of the number of modules of a given type used in the circuit. Therefore, during the post-order traversal, we can

add a term reflecting the extra energy consumed in the controller when using $\lceil \frac{k}{L} \rceil$ modules to implement an operation. Thus the accumulated energy consumed in each node in the DFG will include the energy consumed in the controller. The dynamic programming will also have taken the energy consumed in the controller into consideration. Other useful extensions include the introduction of resource constraints (for long pipeline initiation latency L) and support for re-timing.

Our unique contributions can be summarized as follows:

- To our knowledge, this is the first time that the *time-constrained* component selection is solved by dynamic programming in behavioral synthesis.

- In behavioral level synthesis, multi-cycle operations are used to refer to operations which have a delay of several c-steps long. Long delay times however give rise to *multi-frame* operations, i.e., operations which span over multiple frames. To our knowledge, no existing scheduling algorithm for functional pipelining can schedule the multi-frame operations. In this paper, we presented for the first time, a technique (revolving schedule) to handle not only multi-cycle, but also multi-frame operations.

- We used an input data-sensitive energy model during behavioral level oprimization using multiple supply voltage levels.

Bibliography

[CaWo91] R. Camposano and W. Wolf, "High-level VLSI synthesis", page 256, Kluwer Academic Publishers, 1991.

[ChPo92] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R.W. Brodersen,"HYPER-LP: A System for Power Minimization Using Architectural Transformations," in Proceedings of the ICCAD, November 1992.

[ChPe95a] J.-M. Chang and M. Pedram, "Low Power Register Allocation and Binding," in Proceedings of the 32nd DAC, June 1995.

[ChPe95b] J.-M. Chang and M. Pedram, "Power Efficient Register Assignment", CENG Technical Report 95-03, Computer Engineering Division, Dept. of EE-Systems, Univ. of Southern California, 1995.

[ChPe92] K. Chaudhary and M. Pedram, "Computing the area versus delay trade-off curves in technology mapping," in Proceedings of the IEEE Transactions on CAD, v14, n12, Dec 1995.

[Deng94] C. Deng, "Power Analysis for CMOS/BiCMOS circuits," in Proceedingss of the 1994 International Workshop on Low Power Design, pages 3-8, April 1994.

[DeGa97] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. "Technology mapping in MIS," in Proceedings of the IEEE ICCAD, pp. 116-119, Nov, 1987.

[GaJo79] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Company, 1979.

[GeEl92] C. Gebotys and M. Elmasry, "Optimal VLSI Architectural Synthesis", Kluwer Academic Publishers, page 148, 1992.

[Golu80] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.

[Hayk91] S. Haykin, *Adaptive Filter Theory*, 2nd edition, Chapters 5, 6, and 8, Prentice Hall, 1991.

[JoRo96] M. Johnson and K. Roy. "Low-Power data-path Scheduling under resource constraints,", ICCD, 1996.

[KuAl87] F. Kurdahi, A. Parker, "REAL: A Program for Register Allocation," in Proceedings of the 24th DAC, June 1987.

[LaRa94] P. Landman and J. Rabaey, "Black-Box Capacitance Models for Architectural Power Analysis," in Proceedings of the 1994 International Workshop Low Power Design, April 1994.

[LiLi92] W.-N. Li, A. Lim, P. Agrawal and S. Sahni, "On The Circuit Implementation Problem," in Proceedings of the 29th DAC, June 1992.

[MaKn95] R. Martin and J. Knight, "Power Profiler: Optimizing ASICs power consumption at the behavioral level", DAC, June 1995.

[PaPa88] N. Park, A. Parker,"Sehwa: A Software Package for Synthesis of Pipelines from behavioral Specifications," in IEEE Trans. on CAD, Vol. 7, No. 3, March 1988.

[RaSa95] S. Raje, M. Sarrafzadeh, "Variable Voltage Scheduling," in Proceedings of the 1995 International Workshop Low Power Design, 1995

[Stok91] L. Stok, "Architectural Synthesis and Optimization of Digital Systems," Ph.D Dissertation, Eindhoven University of Technology, 1991.

[ToMo90] H. Toutai, W. Moon, R. Brayton, and A. Wang. "Performance-oriented technology mapping," i n Proceedings of the Sixth M.I.T. Conference on Advanced Research in VLSI, pp. 79-97, April 1990.

[TsPe94] C.-Y. Tsui, M. Pedram, and A. Despain, "Power Efficient Technology Decomposition and Mapping Under and Extended Power Consumption Model," in IEEE trans. on CAD, Vol. 13, No. 9, September 1994.

[UsHo95] K. Usami and M. Horowitz, "Clustered Voltage Scaling Technique for Low-Power Design," in Proceedings of the 1995 International Workshop on Low Power Design, pp. 3-8, 1995

[PaSt82] C. Papadimitriou, K. Steiglitz, "Combinatorial Optimization, Algorithms and Complexity". Prentice-Hall, inc., 1982.