LOGIC-LEVEL POWER ESTIMATION IN CMOS VLSI CIRCUITS

by

Mustafa Aktan

BS. in EE, Boğaziçi University, 1999

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Electrical and Electronics Engineering

Boğaziçi University

2001

ACKNOWLEDGMENTS

I would like to express my thanks to Assoc. Prof. Günhan Dündar, my thesis advisor, for his support and help during my thesis. I would also like to thank to the instructors of the Electrical and Electronics Engineering Department of Boğaziçi University for their contributions to my education in Electronics.

I would like to dedicate this study to my mother for her patience and ever-lasting love.

ABSTRACT

LOGIC-LEVEL POWER ESTIMATION IN CMOS VLSI CIRCUITS

In this thesis, a power estimator for both combinational and sequential CMOS logic circuits is presented. The estimation method is probabilistic in nature and is based on tagged probabilistic simulation (TPS). TPS can handle large circuits efficiently with the implementation of the local BDD approach.

TPS was originally developed for power estimation in combinational circuits. In this work it is extended to handle sequential circuits as well. Recently developed methods use symbolic simulation equations to estimate power in sequential circuits. The symbolic simulation method is not applicable to the local BDD approach. To overcome this bottleneck a simple method based on the calculation of the transition probabilities is proposed. The method is efficient as far as speed is concerned but has accuracy problems when compared to the symbolic simulation method.

ÖZET

SAYISAL CMOS DEVRELERDE KAPI SEVİYESİNDE GÜÇ HESABI

Bu tezde, hem kombinasyonel hem de ardışıl CMOS sayısal devreleri için kullanılabilen bir güç hesaplayıcısı tasarlanmıştır. Hesaplayıcı olasılık temellidir ve parçalı olasılık simulasyonu (TPS) metodunu kullanmaktadır. TPS büyük devreleri de lokal BDD metodunu kullanarak rahatça simule edebilmektedir. Daha önce geliştirilen metodlar ardışıl devrelerde güç hesabı yapmak için sembolik simulasyon denklemleri metodunu kullanmışlardır.

TPS kombinasyonal devrelerde güç hesabı yapmak için tasarlanmıştır. Bu çalışmada, ardışıl devreleri de simule edebilme yeteneği eklenmiştir. Bu metod lokal BDD metoduyla birlikte kullanılamamaktadır. Bu problemi çözmek için, geçiş olasılığına hesabına dayalı basit bir yöntem kullanılmıştır. Bu metod hız dikkate alındığında verimlidir fakat sembolik simulasyon metoduna nazaran doğruluk problemleri vardır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

C_L	Load capacitance
E_n	Switching activity factor for node n
$K^{xy,wz}$	Correlation coefficient corresponding to joint tagged waveform $w_c^{xy,wz}$
$P(0 \rightarrow 1)$	Transition probability from 0 to 1
$P_x(1)$	Probability of x evaluating to 1
P_{avg}	Average power dissipation
$sp_n(t)$	Signal probability at time t
$tu_n(t)$	Upward (rising) transition probability
$tu_d(t)$	Downward (falling) transition probability
V_{dd}	Power supply voltage
V_{gs}	Gate to source voltage
V_T	Threshold voltage
wp_n^{xy}	Occurrence probability of tagged waveform w_n^{xy}
w_n^{xy}	Tagged waveform for node n with initial state x and final state y
$w_c^{xy,wz}$	Joint tagged waveform
BDD	Binary Decision Diagram
STG	State Transition Graph
TPS	Tagged Probabilistic Simulation

1. INTRODUCTION

For many consumer electronic applications low power dissipation is desirable and for certain special applications low power dissipation is of critical importance. For applications such as personal communication systems and hand-held mobile telephones, low-power dissipation may be the tightest constraint in the design. More generally, with the increasing scale of integration, power dissipation will assume greater importance, especially in multi-chip modules where heat dissipation is one of the biggest problems.

Today's electronic systems are designed from specification given at a very high level of abstraction with the availability of electronic design automation (EDA) tools. Different power modeling and estimation techniques have been developed at different levels of design hierarchy. If accuracy is the primary concern, circuit and switch level simulation techniques are the best choice. They are accurate and capable of handling various device models and different circuit design styles. The main shortcoming of simulators at this level is that they suffer from memory and execution time constraints. Moreover, working at this level is not suitable for large, cell-based designs. It is not suitable for power optimization platforms where it might be necessary to make many iterations to reach a plausible solution.

Our main concern is the logic level. Therefore, we will concentrate on what has been done at this level. At this level, efficient and accurate techniques have been proposed. They rely on the fact that most of the power consumed in a CMOS gate is due to the charging and discharging of the load capacitance. Therefore, one has to calculate the switching activity at each node of a network constructed with CMOS gates to estimate the power consumption. Initially, methods were developed to meet the needs for combinational circuits. They introduced significant errors when applied to sequential circuits. Therefore, research was focused on estimation methods for sequential circuits.To estimate the power consumption, one has to calculate the switching activity of the internal nodes of the circuit. Power estimation tools can be grouped in two categories according to how they calculate this switching activity: statistical and probabilistic.

Statistical approaches are proposed in [1], [2]. They consist of applying randomly generated input patterns to the circuit and monitoring, with a simulator, the resulting power value. This is continued until a value of power is obtained with a desired accuracy, at a specified confidence level.

For probabilistic methods, estimating the switching activity E_n at a circuit node n involves estimation of the transition probabilities $P_n(0 \rightarrow 1)$ and $P_n(1 \rightarrow 0)$. For combinational circuits, transition probabilities can directly be calculated from state probability $P_n(1)$, the probability that the signal value at node n is one. This probability can be exactly calculated using binary decision diagrams (BDD) [3].

In [4], the exact power estimation of a given combinational circuit is carried out by creating a set of symbolic functions such that summing the signal probabilities of the functions correspond to the average switching activity at a circuit line n in the original combinational circuit. Each function n is the exclusive-OR of the characteristic functions describing the logic values of n at two consecutive clock cycles. The major drawback of this method is its exponential complexity. Another problem is that it does not offer a glitch filtering mechanism to avoid over estimations.

An efficient way for representing the switching activity is using probability waveforms that were introduced in [5]. These waveforms consist of a sequence of transition events over time from the initial state to the final steady state where each event is annotated with an occurrence probability. Given these waveforms, it is straightforward to calculate the switching activity of n that includes the contribution of hazards and glitches. A simulator (DENSIM) developed based on this concept is proposed in [6]. DENSIM is fast but it cannot deal with spatial correlations that introduce a significant amount of error.

A tagged probabilistic (TPS) approach is described in [7] that correctly accounts for reconvergent fanout and glitches. The key idea is to break the set of possible logical waveforms at a node n into four groups. Each group is characterized by its steady state values. Next, each group is combined into a probability waveform with the appropriate steady-state tag. The correlation between probability waveforms at the inputs is

approximated by the correlation between the steady state values of these lines. This approach runs faster than symbolic simulation and offers a simple but worthwhile glitch filtering mechanism.

The main drawback of BDD based approaches is that their huge memory requirement with increasing number of primary inputs. With this aside, the time required to traverse the BDD increases exponentially with increasing number of primary inputs. These problems are overcome with the local BDD approach which is introduced in [8]. The cost is a negligible amount of inaccuracy in switching activity factor.

The estimates produced by purely combinational methods can greatly differ from those produced by the exact probability method. Accurate average switching activity estimation is considerably more difficult than for combinational circuits, because the probability of the circuit being in each of its possible states has to be calculated.

Exact and approximate methods of calculating state probabilities are introduced in [9]. The exact method is based on solving the Chapman-Kolmogorov equations for discrete-time discrete state Markov process. This method is limited to sequential circuits that have a maximum of 15 flip-flops. The approximation scheme is based on calculating the present state line probabilities. The improvement in accuracy is significant compared to the method used in [4], where the sequential circuit was assumed to be equally likely be in either of the possible states.

In summary, probabilistic simulation techniques, using the local BDD approach, constitute the best choice for switching activity estimation at gate level. In this work, a power simulator (EPOSIM) based on tagged probabilistic simulation (TPS) is developed and it is extended to handle sequential circuits. EPOSIM takes as input structural level VHDL code compiled with EPOCOM. Hierarchical VHDL code gives a powerful capability to handle large circuits in a hierarchical manner. Although the local BDD approach outperforms the need for a hierarchical partitioning of the circuit, with this capability, EPOSIM can easily be extended to run simulations at higher levels of the design hierarchy using different models such as the information theoretic model described in [10].

The estimation results obtained with the methods discussed above are in general compared to results again with logic-level simulators. Their main focus is on the estimation of switching activity factor so the results are not exact measures of power consumption. This comes more apparent when compared with spice like simulators. Accurate modeling of the gates is important and this is also a part of this work.

The outline of the thesis is as follows: In Section 2, causes of power consumption are discussed and the power modeling is done for gate-level abstraction. Section 3 deals with probability calculation techniques for Boolean functions. In Section 4, basics of Tagged Probabilistic Simulation (TPS) are given. Power estimation techniques for sequential circuits are the topic of Section 5. The simulation results are given in Section 6.

2. POWER CONSUMPTION IN STATIC CMOS CIRCUITS

2.1. Power Consumption

Causes of power consumption in static CMOS circuits can be grouped into two categories: static and dynamic.

2.1.1. Static Power Consumption

In ideal case static power consumption of static CMOS circuits is zero. This is because PMOS and NMOS devices are never on simultaneously. But there is always a small leakage current flowing through the reverse biased diode junctions of the transistors located between the source and drain and the substrate. This current is in general very small and can be ignored.

A more important source of leakage current is potentially the subthreshold current of the transistors. The subthreshold current for long channel devices increases linearly with the ratio of the channel width over channel length and decreases nearly exponentially with decreasing $V_{GS} - V_T$ where V_{GS} is the gate bias and V_T is the threshold voltage. The reduction of power supply and device threshold voltages, cause subthreshold current to become more pronounced. In addition, at short channel lengths, the subthreshold current also becomes exponentially dependent on drain voltage.

2.1.2. Dynamic Power Consumption

There are mainly two sources for dynamic power dissipation in CMOS circuits: short circuit power consumption and the power consumed by charging and discharging the load capacitance of the circuit. Both depend on the switching activity of the circuit.

For an inverter the maximum short circuit current flows when there is no load. This current decreases with the load. If gate sizes are selected so that the input and output

rise/fall times are approximately equal, the short circuit power consumption will be less than 15% of the dynamic power consumption [11].

The dominant source of dynamic power consumption is the charging of the load capacitance that can be formulated as follows

$$P_L = f \times V_{dd}^2 \times C_L \times P(0 \rightarrow 1)$$

where f is the clock frequency, V_{dd} is the supply voltage, and $P(0 \rightarrow 1)$ is the occurrence probability of a capacitor charging (rising) event.

2.2. Power Dissipation Model

It is widely accepted that the short-circuit and subthreshold currents in CMOS circuits can be made small with proper circuit and device design techniques. Thus the amount of energy dissipated by a CMOS logic gate is roughly equal to the change in energy stored in the gate's output capacitance. Assume gate n is part of a synchronous digital system controlled by a global clock, and then the average power dissipated by n is given by:

$$P_n = 0.5 \times C_L \times f \times V_{dd}^2 \times E_n \tag{2.1}$$

where P_n denotes the average power, C_L is the load capacitance, V_{dd} is the supply voltage, f is the global clock frequency, and E_n is the average number of gate output transitions per global clock cycle.

However this formulation is inadequate in the sense that it does not account for the power consumed inside the gate and the consumption when there is no load, i.e. it models the power consumption due to the load capacitance. A gate has also self loading capacitances which for a gate can be defined as the load driven by the gate when the output is left open. Experimental results show that self-loading capacitances contribute up to 20%

of the total power consumption of CMOS circuits [12]. There is also internal power consumption due to the charge of the internal capacitances of the gate. Therefore C_L should better be replaced by an effective capacitance C_e that is the sum of the effective internal capacitance, self-loading capacitance and loading capacitances.

For a specific library, f and V_{dd} are known. However current libraries are not documented to meet the needs of power optimisation/estimation tools. They need to be rearranged so that the self-loading capacitance and the effect of the internal capacitances to the input/output capacitances are taken into account.

Under the assumption that we have the proper library, the problem of estimating the average power can be reduced to one estimating the average switching activity, E_n, at the gate output. Factors that affect E_n are as follows:

- The logic function performed
- Statistical properties of the inputs
- Timing relationship of the inputs

Ignoring the timing relationship at the inputs, the resulting switching activity is the minimum average activity that can be observed at the gate output. The power consumed due to this switching activity may be referred as the zero-delay power. This is the minimum average power consumed by the gate.

Unequal signal arrival times at the inputs may cause hazards and glitches which means an extra switching activity. The power consumed by this extra activity is referred to as glitch power.

3. SIGNAL PROBABILITY CALCULATIONS

3.1. Static Probabilities

In this context, the following notation will be used: given x as a Boolean variable, $P_x(1)$ is defined to be the probability that x assumes logic one. The probability that x is logic zero, denoted $P_x(0)$ is

$$P_x(0) = 1 - P_x(1)$$

Here, $P_x(1)$ and $P_x(0)$ are referred to as the static probabilities.

In general, given a disjoint cover for a Boolean function of uncorrelated inputs, the probability of the function evaluating to a one is equal to the sum of the probabilities of each of the cubes in the cover evaluating to a one. Assume a two-input combinational circuit of which the Boolean function f is

$$f = a + b$$

where a and b are the primary inputs. The inputs are assumed to be uncorrelated. To evaluate $P_f(1)$, the primary input probabilities are given as $P_a(1)$ and $P_b(1)$. Note that

$$P_f(1) = P_{a+b}(1) \neq P_a(1) + P_b(1)$$

because the first and second product terms are not independent. Rather

$$P_{a+b}(1) = P_{a+\bar{a}\bullet b}(1) = P_a(1) + P_a(0)P_b(1)$$

where the second equality holds because a is disjoint from $\bar{a} \bullet b$. That is, there is not any input vector that evaluates both of them to logic one at the same time. Here, $a + \bar{a} \bullet b$

represents a disjoint cover for the logic function f and the terms a and $\overline{a} \bullet b$ are referred to as the cubes in the cover.

3.2. Transition Probabilities

For static combinational CMOS logic, a gate output can only change when its inputs change, and then only if the Boolean function describing the gate evaluates differently. Consider the function f given in the previous section. f will change between clock cycle t and $t+1$ if

$$f(t) \oplus f(t+1) = (a(t)+b(t)) \oplus (a(t+1)+b(t+1)) \tag{3.1}$$

evaluates to one, where $a(t)$, $b(t)$ and $a(t+1)$, $b(t+1)$ are the inputs to the gate at clock cycles t and $t+1$ respectively. The disjoint cover for (3.1) is

$$\begin{aligned} &a(t) \bullet \overline{a}(t+1) \bullet \overline{b}(t+1) + \overline{a}(t) \bullet b(t) \bullet \overline{a}(t+1) \bullet \overline{b}(t+1) \\ &+ \overline{a}(t) \bullet \overline{b}(t) \bullet a(t+1) + \overline{a}(t) \bullet \overline{b}(t) \bullet \overline{a}(t+1) \bullet b(t+1) \end{aligned} \tag{3.2}$$

To evaluate the probability of (3.2), transition probabilities for the transitions $0 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 0$, and $1 \rightarrow 1$ must be used, because of the fact that an input at time $t+1$ may be correlated to its behaviour at time t (as in a sequential circuit). The transition probabilities are denoted as $P_x(0 \rightarrow 0)$, $P_x(0 \rightarrow 1)$, $P_x(1 \rightarrow 0)$, and $P_x(1 \rightarrow 1)$, respectively. Static probabilities can be calculated from transition probabilities as follows:

$$\begin{aligned} P_x(0) &= P_x(0 \rightarrow 0) + P_x(0 \rightarrow 1) \\ P_x(1) &= P_x(1 \rightarrow 0) + P_x(1 \rightarrow 1) \end{aligned}$$

Both static and transition probabilities are used to compute (3.2). The expression for the probability that (3.2) evaluates to one now becomes

$$SW_f = P_a(1 \rightarrow 0)P_b(1) + P_a(1 \rightarrow 1)P_b(1 \rightarrow 0) + P_a(0 \rightarrow 1)P_b(1) + P_a(1 \rightarrow 1)P_b(0 \rightarrow 1)$$

where SW_f is the switching activity.

3.3. Probability Calculation Using BDD's

By using BDD' s we can exactly calculate the signal probability of a circuit node. It is calculated by constructing a BDD representation of the global function corresponding to the node. Once the BDD is constructed, the problem reduces to a satisfy-all problem [3]. The BDD is traversed and the input vectors that cause the node to be a one are summed according to their probability value.

For example, we have a circuit with four primary inputs i_1, i_2, i_3 and i_4. Consider a node n in this circuit for which we want to calculate the signal probability. Say $i_1\overline{i_2}i_3\overline{i_4}$, $\overline{i_1}\overline{i_2}i_3\overline{i_4}$, and $i_1i_2i_3\overline{i_4}$ are the input vectors satisfying a one. The signal probability of n evaluating to a one is

$$P_n(1) = \sum_{i\ such\ that\ n=1} P_{v_i}(1) = P_{i_1\overline{i_2}i_3\overline{i_4}}(1) + P_{\overline{i_1}\overline{i_2}i_3\overline{i_4}}(1) + P_{i_1i_2i_3\overline{i_4}}(1)$$

Under the assumption that the primary inputs are uncorrelated the above equation becomes

$$P_n(1) = P_{i_1}(1)P_{i_2}(0)P_{i_3}(1)P_{i_4}(0) + P_{i_1}(0)P_{i_2}(0)P_{i_3}(1)P_{i_4}(0) + P_{i_1}(1)P_{i_2}(1)P_{i_3}(1)P_{i_4}(0)$$

This formulation is implemented by slightly modifying the satisfy-all algorithm given in [3]. The pseudo code of the modified algorithm is shown in Figure 3.1.

3.3.1. State Probability Calculation

Consider two nodes m and n in a circuit for which we want to calculate the probability of m being a 0 and n being a 1. This can be done In two ways. The first approach is constructing the OBDD representation of $\overline{m} \bullet n$ and then using the algorithm mentioned in Figure 3.1. This approach is inefficient in terms of memory and time. The

second approach is traversing the OBDD representations of m and n and finding the input vectors that satisfy the constraints m=0 and n=1. This is again a satisfy-all problem, but now we have to deal with two vertices. The algorithm is shown in Figure 3.2.

```
char  X[]: primary input state vector
float P[]: array of input probabilities
float p : probability

satisfyAll ( v: vertex, i: integer ) {
 if( v = V0 ) return;
 if( v = V1 ){
       p = p + probability(X[], i);
       return;
 }

 X[i] = '0';
 satisfyAll(LOW(v), i + 1);

 X[i] = '1';
 satisfyAll(HIGH(v), i + 1);
}
```

Figure 3.1. Signal probability calculation algorithm

3.3.2. Transition Probability Calculation

Let f be a node in a Boolean network represented in terms of n primary inputs $x_1, x_2, \ldots x_n$. f is expressed through the two sets of BDD paths:

- $\prod_1$: The set of all paths in the ON-set of f
- $\prod_0$: The set of all paths in the OFF-set of f

Based on this representation, the probability of the event f switching from value i to value j ($i, j \in \{0,1\}$) is written as

$$f(i \to j) = \sum_{\pi \in \Pi_i} \sum_{\pi' \in \Pi_j} \prod_{k=1}^{n} x_k(i_k \to j_k)$$

where i_k, j_k are the values of x_k on paths π and π' respectively. Thus the probability that f switches from i to j is expressed as

$$P_f(i \to j) = \sum_{\pi \in \Pi_i} \sum_{\pi' \in \Pi_j} P(\prod_{k=1}^{n} x_k(i_k \to j_k)) \tag{3.3}$$

```
char  X[]: primary input state vector
float P[]: array of input probabilities
float p : probability of the node

satisfy_01 ( v1, v2: vertex, i: integer ) {
 if( v1 = V1 || v2 = V0 ) return;
 if( v1 = V0 && v2 = V1 ){
       p = p + probability(X[], i);
       return;
 }

 X[i] = '0';
 satisfy_01(LOW(v1), LOW(v2), i + 1);

 X[i] = '1';
 satisfy_01(HIGH(v1), HIGH(v2), i + 1);
}
```

Figure 3.2. State probability calculation algorithm

This will be clearer on the following example. f is a Boolean function expressed in terms of the primary inputs a and b as

$$f = a + b$$

The OFF-set and ON-set of f can be expressed in terms of the inputs a and b as

$$\prod_0 = \{(0,0)\}$$

$$\prod_1 = \{(0,1),(1,0),(1,1)\}$$

Thus a $0 \to 1$ transition occurs when a=0, b=0 at time t, a=0, b=1 or a=1, b=0 or a=1, b=1 at time t+1. Thus the probability is

$$P_f(0 \to 1) = a_{0\to0}b_{0\to1} + a_{0\to1}b_{0\to0} + a_{0\to1}b_{0\to1} \tag{3.4}$$

Knowing the transition probabilities of the inputs, it is straightforward to calculate (3.4).

3.4. Zero-Delay Power Dissipation

Consider a Boolean function f with inputs temporally uncorrelated. The switching activity of f can be directly computed from the static probabilities $P_f(1)$ and $P_f(0)$ as follows:

$$E_f = P_{f(t)\oplus f(t+1)}(1) = P_f(1)P_f(0) + P_f(0)P_f(1) = 2\times P_f(1)\times P_f(0)$$
$$E_f = 2\times P_f(1)\times(1-P_f(1)) \tag{3.5}$$

where $P_f(1)$ is directly computed from the static probabilities of the primary inputs.

For a general combinational network power dissipation can be formulated, using (3.5), as follows:

$$P_{avg} = 0.5\times f\times V_{dd}^2\times\sum C_i\times E_i \tag{3.6}$$

where C_i denotes the load capacitance and E_i denotes the expected value of transitions of the i'th gate, and the sum is over all gates in the network. If the inputs are temporally and spatially uncorrelated, and their static probabilities are given, then the static probability of each node in the network can easily be computed as described in Section 3.1. If we assume zero delay for each gate in the network, then plugging Equation (3.5) in (3.6) we get

$$P_{avg} = f\times V_{dd}^2\times\sum C_i\times P_i(1)\times(1-P_i(1)) \tag{3.7}$$

which is referred to as the zero-delay average power dissipation of the network.

4. TAGGED PROBABILISTIC SIMULATION

In this section a review of Tagged Probabilistic Simulation (TPS) is given. Some of the concepts are reviewed and some of them are extended in this work.

4.1. Tagged Probability Waveforms

The probability waveform is a compact representation of the set of all logic waveforms at a node under all possible input vector pairs between two consecutive clock cycles. A probability waveform is a sequence of transition events over time where each event is annotated with an occurrence probability (see Figure 4.1.b).

A tagged probability waveform is obtained by partitioning a probability waveform according to the initial and final steady-state logic values of the logical waveforms that have contributed to this waveform. That is, four tagged waveforms can be defined at each node n: w_n^{00}, w_n^{01}, w_n^{10}, and w_n^{11}. All logic waveforms at the output of node n with initial state x and final state y are compactly represented by the tagged waveform w_n^{xy}.

A probability waveform w_n^{xy} has three aspects: *signal probability, transition probability* and *waveform probability* that are defined as follows. *Signal probability* $sp_n(t)$ is defined as the probability that a node n assumes logic *one* at time t during a transition from state x to state y. t takes values between 0 and T where T is the clock period. The transition probability ($tu_n(t)/tu_d(t)$) is the occurrence probability of an event at time t. An event that changes from zero (one) to one (zero) is defined as the *upward transition probability* $tu_n(t)$ (*downward transition probability* $tu_d(t)$). Waveform probability wp_n^{xy} is the occurrence probability of the tag xy and it is constant. It is the sum of the probability of node n assuming a one (*signal probability*) and zero at any t.

Figure 4.1.b shows a probability waveform that is a compact representation of the three logic waveforms in Figure 4.1.a.

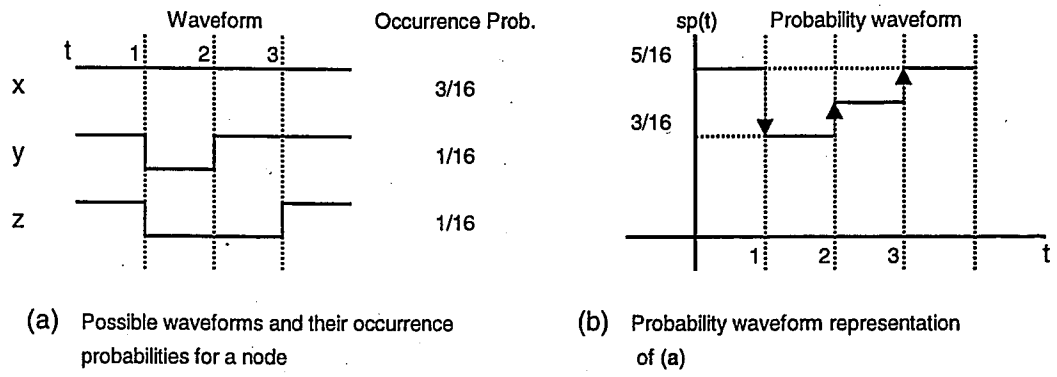

Figure 4.1. Probability Waveforms

Figure 4.1.a shows three possible $1 \rightarrow 1$ transition logic waveforms with their occurrence probabilities at some node n, i.e. tag w_n^{11}. Thus wp_n^{11} is equal to the sum of the probabilities corresponding to waves x, y, and z which is $5/16$. This is also the signal probability for $0 < t < 1$. At time t=1 there are two falling events belonging to waves y and z. The probabilities of these events are the occurrence probabilities of the waves to which they belong. Thus the probability of a falling event ($td_n(t=1)$) at time t=1 is $2/16$. Since only x assumes logic one for $1 < t < 2$ the signal probability for that interval is the occurrence probability of x, $3/16$. Calculation of signal probabilities after transition events is formulised as follows

$$sp_n(t_+) = sp_n(t_-) + tu_n(t) - td_n(t) \tag{4.1}$$

where $sp_n(t_+)$ ($sp_n(t_-)$) is the signal probability after (before) any events ($tu_n(t)/tu_d(t)$) occurring at time t.

Having constructed the probability waveforms at some node n the average power consumption due to the switching activity at n can be computed as follows

$$P_{avg,n} = 0.5 \times f \times V_{dd}^2 \times C_n \times \sum_{x=0}^{1} \sum_{y=0}^{1} \sum_{k=0}^{W} td_n^{xy}(t_k) + tu_n^{xy}(t_k) \tag{4.2}$$

where f, V_{dd}, and C_n are defined as in Equation (2.1).

4.2. Waveform Propagation

The waveform propagation mechanism used for TPS deals with four tagged waveforms. In [7], propagation through a two-input AND gate and an inverter is given. The behaviour of other two-input gates are emulated by using AND gates and inverters. Multi-input primitive gates are again decomposed into ANDs and inverters. To avoid the overhead caused by this method in our work, the propagation equations for all primitive gates are derived as follows:

4.2.1. Single-input Gates

4.2.1.1. Inverter. A rising event at the input of an inverter will cause a falling event at the output with a time shift equal to delay of the inverter. The resulting event will belong to output tag which has an opposite phase to the input. That is, for an event on input tag 00, the output tag will be 11, for 01 it will be 10, for 10 it will be 01, and for 11 it will be 00. We can write the equations as follows:

Let c be an inverter with input node a, delay d. Then

$$tu_c^{00}(t+d) = td_a^{11}(t),\ td_c^{00}(t+d) = tu_a^{11}(t)$$
$$tu_c^{01}(t+d) = td_a^{10}(t),\ td_c^{01}(t+d) = tu_a^{10}(t)$$
$$tu_c^{10}(t+d) = td_a^{01}(t),\ td_c^{10}(t+d) = tu_a^{01}(t)$$
$$tu_c^{11}(t+d) = td_a^{00}(t),\ td_c^{11}(t+d) = tu_a^{00}(t)$$

4.2.1.2. Buffer. The events at the output of a buffer will be delayed versions of the inputs. They will preserve their tags and types(upward/downward). For a buffer c with input node a and delay d

$$tu_c^{00}(t+d) = tu_a^{00}(t),\ td_c^{00}(t+d) = td_a^{00}(t)$$
$$tu_c^{01}(t+d) = tu_a^{01}(t),\ td_c^{01}(t+d) = td_a^{01}(t)$$
$$tu_c^{10}(t+d) = tu_a^{10}(t),\ td_c^{10}(t+d) = td_a^{10}(t)$$

$$tu_c^{11}(t+d) = tu_a^{11}(t),\ td_c^{11}(t+d) = td_a^{11}(t)$$

are the propagation equations.

4.2.2. Two-input Gates

Consider a two-input gate with inputs a and b and gate output c. During the waveform propagation, there are four tagged waveforms w_a^{xy} (w_b^{xy}) at input a (b), where $x, y, w, z \in \{0,1\}$. That makes 16 tagged waveform combinations at the gate inputs. Each of these combinations is referred to as the input *joint tagged waveform* and denoted by $w_c^{xy,wz}$. For instance, $w_c^{01,10}$ represents the logic waveforms produced at gate output c when logic waveforms at input a assume initial state 0 and final state 1 and logic waveforms at input b assume initial 1 and final state 0. All the joint tagged waveforms that produce the same initial and final state values at the gate output are added together to form the new four tagged waveforms at c.

Let c be a two input gate, which has a gate delay d and inputs named a and b. Consider tagged waveform $w_c^{xy,wz}$. For input a, there are four possible combinations of values that a can assume at time t_- and t_+. Let P_a^{00}, P_a^{01}, P_a^{10}, and P_a^{11} represent the occurrence probabilities of these four combinations. From the waveform probability wp_a^{xy}, the transition probabilities $tu_a^{xy}(t)$, $td_a^{xy}(t)$, and signal probability $sp_a^{xy}(t_-)$, these four occurrence probabilities can be computed as

$$\begin{aligned} P_a^{00}(t) &= wp_a^{xy} - sp_a^{xy}(t_-) - tu_a^{xy}(t) \\ P_a^{01}(t) &= tu_a^{xy}(t),\ P_a^{10}(t) = td_a^{xy}(t) \\ P_a^{11}(t) &= sp_a^{xy}(t_-) - td_a^{xy}(t) \end{aligned} \tag{4.3}$$

where wp_a^{xy} is mistakenly written as 1 in [7]. The same equations are valid for input b. Derivation of the propagation equations for the AND, OR, NAND, and EXOR gates are given in the following sections.

4.2.2.1. AND Gate. First consider the propagation of a rising event (transition) at the inputs. For a rising event occurring at one of the inputs to propagate the other input should either experience a $0 \rightarrow 1$ transition or $1 \rightarrow 1$ transition. This can be expressed in terms of occurrence probabilities and (4.3) as follows:

$$tu_c^{xy,wz} = P_a^{01}(t)P_b^{11}(t) + P_a^{01}(t)P_b^{01}(t) + P_a^{11}(t)P_b^{01}(t)$$

$$tu_c^{xy,wz} = P_a^{01}(t)(P_b^{01}(t) + P_b^{11}(t)) + P_b^{01}(t)(P_a^{01}(t) + P_a^{11}(t)) - P_a^{01}(t)P_b^{01}(t)$$

$$tu_c^{xy,wz}(t+d) = tu_a^{xy}(t)sp_b^{wz}(t_+) + tu_b^{wz}(t)sp_a^{xy}(t_+) - tu_a^{xy}(t)tu_b^{wz}(t)$$

Similarly for the falling events.

$$td_c^{xy,wz} = P_a^{10}(t)P_b^{11}(t) + P_a^{10}(t)P_b^{10}(t) + P_a^{11}(t)P_b^{10}(t)$$

$$td_c^{xy,wz} = P_a^{10}(t)(P_b^{10}(t) + P_b^{11}(t)) + P_b^{10}(t)(P_a^{10}(t) + P_a^{11}(t)) - P_a^{10}(t)P_b^{10}(t)$$

$$td_c^{xy,wz}(t+d) = td_a^{xy}(t)sp_b^{wz}(t_-) + td_b^{wz}(t)sp_a^{xy}(t_-) - td_a^{xy}(t)td_b^{wz}(t)$$

The waveform tag (w_c^{ab}) of the resulting output event ($tu_c^{xy,wz}, td_c^{xy,wz}$) can be found from the forcing set table of an AND gate (Table 4.1). This is a list of the tags of the joint tagged waveforms for a two-input AND gate that should be combined into each output tagged waveform. For example, if we consider the input waveforms w_a^{01} and w_b^{11} at inputs a and b respectively, events propagating to the output will belong to output tag 01 (w_c^{01}).

4.2.2.2. OR Gate. For a rising event occurring at one of the inputs to propagate the other input should either experience a $0 \rightarrow 1$ transition or $0 \rightarrow 0$ transition. That is

$$tu_c^{xy,wz} = P_a^{01}(t)P_b^{00}(t) + P_a^{01}(t)P_b^{01}(t) + P_a^{00}(t)P_b^{01}(t)$$

$$tu_c^{xy,wz} = P_a^{01}(t)(P_b^{01}(t) + P_b^{00}(t)) + P_b^{01}(t)(P_a^{01}(t) + P_a^{00}(t)) - P_a^{01}(t)P_b^{01}(t)$$

$$tu_c^{xy,wz}(t+d) = tu_a^{xy}(t)(wp_b^{wz} - sp_b^{wz}(t_-)) + tu_b^{wz}(t)(wp_a^{xy} - sp_a^{xy}(t_-)) - tu_a^{xy}(t)tu_b^{wz}(t)$$

$$td_c^{xy,wz} = P_a^{10}(t)P_b^{00}(t) + P_a^{10}(t)P_b^{10}(t) + P_a^{00}(t)P_b^{10}(t)$$

$$td_c^{xy,wz} = P_a^{10}(t)(P_b^{10}(t) + P_b^{00}(t)) + P_b^{10}(t)(P_a^{10}(t) + P_a^{00}(t)) - P_a^{10}(t)P_b^{10}(t)$$

$$td_c^{xy,wz} = td_a^{xy}(wp_b^{wz} - sp_b^{wz}(t\_) - tu_b^{wz}(t)) + td_b^{wz}(wp_a^{xy} - sp_a^{xy}(t\_) - tu_a^{xy}(t))$$
$$- td_a^{xy}(t)td_b^{wz}(t)$$

Using Equation (4.1)

$$td_c^{xy,wz}(t+d) = td_a^{xy}(t)(wp_b^{wz} - sp_b^{wz}(t_+)) + td_b^{wz}(t)(wp_a^{xy} - sp_a^{xy}(t_+)) - td_a^{xy}(t)td_b^{wz}(t)$$

The forcing set table of an OR gate is given in Table 4.2.

Table 4.1. Forcing set table for a two-input AND gate

OUTPUT TAGS	INPUT TAGS
00	[00,00], [00,01], [00,10], [00,11] [01,00], [01,10] [10,00], [10,01] [11,00]
01	[01,01], [01,11] [11,01]
10	[10,10], [10,11] [11,10]
11	[11,11]

4.2.2.3. NAND Gate. Rising (falling) events at the output of a NAND gate are produced by falling (rising) events at the inputs. For a falling event occurring at one of the inputs to propagate the other input should either experience a $1 \rightarrow 0$ transition or $1 \rightarrow 1$ transition. That is

$$tu_c^{xy,wz} = P_a^{10}(t)P_b^{11}(t) + P_a^{10}(t)P_b^{10}(t) + P_a^{11}(t)P_b^{10}(t)$$

$$tu_c^{xy,wz} = P_a^{10}(t)(P_b^{10}(t) + P_b^{11}(t)) + P_b^{10}(t)(P_a^{10}(t) + P_a^{11}(t)) - P_a^{10}(t)P_b^{10}(t)$$

$$tu_c^{xy,wz}(t+d) = td_a^{xy}(t)sp_b^{wz}(t\_) + td_b^{wz}(t)sp_a^{xy}(t\_) - td_a^{xy}(t)td_b^{wz}(t)$$

$$td_c^{xy,wz} = P_a^{01}(t)P_b^{11}(t) + P_a^{01}(t)P_b^{01}(t) + P_a^{11}(t)P_b^{01}(t)$$

$$td_c^{xy,wz} = P_a^{01}(t)(P_b^{01}(t) + P_b^{11}(t)) + P_b^{01}(t)(P_a^{01}(t) + P_a^{11}(t)) - P_a^{01}(t)P_b^{01}(t)$$

$$td_c^{xy,wz} = tu_a^{xy}(t)(tu_b^{wz}(t) + sp_b^{wz}(t\_) - td_b^{wz}(t))$$
$$+ tu_b^{wz}(t)(tu_a^{xy}(t) + sp_a^{xy}(t\_) - td_a^{xy}(t)) - td_a^{xy}(t)td_b^{wz}(t)$$

Using Equation (4.1)

$$td_c^{xy,wz}(t+d) = tu_a^{xy}(t)sp_b^{wz}(t_+) + tu_b^{wz}(t)sp_a^{xy}(t_+) - tu_a^{xy}(t)tu_b^{wz}(t)$$

The forcing set table of a NAND gate is given in Table 4.3.

Table 4.2. Forcing set table for a two-input OR gate

OUTPUT TAGS	INPUT TAGS
00	[00,00]
01	[01,01], [01,00] [00,01]
10	[10,10], [10,00] [00,10]
11	[11,11], [11,10], [11,01], [11,00] [10,11], [10,01] [01,11], [01,10] [00,11]

4.2.2.4. EXOR Gate. The EXOR gate is different then other gates that a rising/falling event can cause either a rising or falling event at the output depending on the transition occurring at the other input. A rising event at the output can be generated either by a falling event at one input and a $1 \rightarrow 1$ transition at the other or by a rising event at one input and a $0 \rightarrow 0$ transition at the other. We can formulise the probability of a rising event at the output of an EXOR gate c with inputs a and b, delay d as follows:

$$tu_c^{xy,wz} = P_a^{01}(t)P_b^{00}(t) + P_a^{00}(t)P_b^{01}(t) + P_a^{10}(t)P_b^{11}(t) + P_a^{11}(t)P_b^{10}(t)$$

$$tu_c^{xy,wz} = tu_a^{xy}(t)(wp_b^{wz} - sp_b^{wz}(t_-) - tu_b^{wz}(t)) + tu_b^{wz}(t)(wp_a^{xy} - sp_a^{xy}(t_-) - tu_a^{xy}(t))$$
$$+ td_a^{xy}(t)(sp_b^{wz}(t_-) - td_b^{wz}(t)) + td_b^{wz}(t)(sp_a^{xy}(t_-) - td_a^{xy}(t))$$

Similarly for a falling event

$$td_c^{xy,wz} = P_a^{10}(t)P_b^{00}(t) + P_a^{00}(t)P_b^{10}(t) + P_a^{01}(t)P_b^{11}(t) + P_a^{11}(t)P_b^{01}(t)$$

$$td_c^{xy,wz} = td_a^{xy}(t)(wp_b^{wz} - sp_b^{wz}(t_-) - tu_b^{wz}(t)) + td_b^{wz}(t)(wp_a^{xy} - sp_a^{xy}(t_-) - tu_a^{xy}(t))$$
$$+ tu_a^{xy}(t)(sp_b^{wz}(t_-) - td_b^{wz}(t)) + tu_b^{wz}(t)(sp_a^{xy}(t_-) - td_a^{xy}(t))$$

Using Equation (4.1)

$$td_c^{xy,wz} = td_a^{xy}(t)(wp_b^{wz} - sp_b^{wz}(t_+) - td_b^{wz}(t)) + td_b^{wz}(t)(wp_a^{xy} - sp_a^{xy}(t_+) - td_a^{xy}(t))$$
$$+ tu_a^{xy}(t)(sp_b^{wz}(t_+) - tu_b^{wz}(t)) + tu_b^{wz}(t)(sp_a^{xy}(t_+) - tu_a^{xy}(t))$$

Table 4.3. Forcing set table for a two-input NAND gate

OUTPUT TAGS	INPUT TAGS
00	[11,11]
01	[10,10], [10,11] [11,10]
10	[01,01], [01,11] [11,01]
11	[00,00], [00,01], [00,10], [00,11] [01,00], [01,10] [10,00], [10,01] [11,00]

Table 4.4. Forcing set table for a two-input EXOR gate

OUTPUT TAGS	INPUT TAGS
00	[00,00], [01,01], [10,10], [11,11]
01	[00,01], [01,00], [11,10], [10,11]
10	[11,01], [01,11], [00,10], [10,00]
11	[01,10], [00,11], [11,00], [10,01]

4.2.3. Multi-input Gates

Although propagation equations for multi-input gates can be derived, it is better to decompose them into single and two input primitive gates and use the derived equations for those gates. Figure 4.2 shows a three-input AND gate decomposed into two two-input AND gates. In this work, structural level VHDL code is used as for circuit description. Therefore decomposition is already done when the code is written.

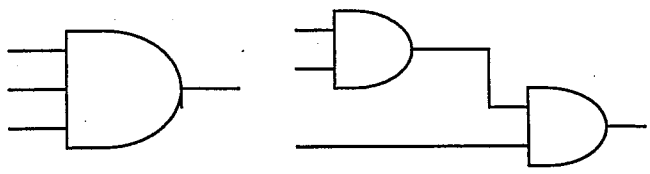

Figure 4.2. Gate decomposition

4.3. Taking Correlation into Account

To exactly calculate the output tagged waveform from a joint input tagged waveform in a two-input gate, we need to know the correlation between any two states at the same time instance of the input tagged waveforms from different gate inputs. To take into account the exact correlation is practically impossible even for small circuits. Therefore, TPS approximates it by allowing only pair-wise correlation. The correlation coefficient at the inputs of a two-input gate, for two input tags w_a^{xy} and w_b^{wz}, ignoring temporal correlations, is

$$\mathrm{K}^{xy,wz} = \frac{P(a=x \wedge b=w)P(a=y \wedge b=z)}{P(a=x)P(b=w)P(a=y)P(b=z)} \tag{4.4}$$

where $x, y, w, z \in \{0,1\}$.

Taking into account the correlation, the propagation equation for a two input AND gate can be rewritten as follows

$$\begin{aligned} tu_c^{xy,wz}(t+d) &= \mathrm{K}^{xy,wz}\{tu_a^{xy}(t)sp_b^{wz}(t_+) + tu_b^{wz}(t)sp_a^{xy}(t_+) - tu_a^{xy}(t)tu_b^{wz}(t)\} \\ td_c^{xy,wz}(t+d) &= \mathrm{K}^{xy,wz}\{td_a^{xy}(t)p_b^{wz}(t_-) + td_b^{wz}(t)p_a^{xy}(t_-) - td_a^{xy}(t)td_b^{wz}(t)\} \end{aligned} \tag{4.5}$$

where $\mathrm{K}^{xy,wz}$ is the correlation coefficient corresponding to the joint tagged waveform $w_c^{xy,wz}$.

4.4. Glitch Filtering

Glitch filtering is necessary because of the fact that short glitches in the logic waveforms do not pass through gates due to inertial delay of these gates. The minimum glitch width that enables it to propagate through a gate can be taken as the gate delay. In [7], only glitch filtering in an AND gate is mentioned. Here, glitch-filtering mechanisms for the rest of the primitive gates are developed.

For single input gates such as inverters, it is difficult to do filtering since the correlation between successive transition events that are to be filtered within a gate delay time is almost impossible to determine. For two-input gates, glitches that come from the same input are ignored for the same reason. Here, glitch filtering mechanisms for two-input gates are developed based on the mechanism for an AND gate in [7]. Glitch filtering is applied on joint tagged waveform $w_c^{xy,wz}$ of two-input gates c with delay d, inputs a and b.

4.4.1. AND Gate

For an upward transition event $tu_a^{xy}(t1)$, all of the downward transition events $td_b^{wz}(t2)$ that come from the other gate input and where $t2$ is within a gate delay time after $t1$ $(t1 < t2 \le t1+d)$ are subject to glitch filtering. That is, $\mathrm{K}^{xy,wz} tu_a^{xy}(t1) td_b^{wz}(t2)$ is subtracted from both $tu_c^{xy}(t1+d)$ and $td_c^{wz}(t2+d)$.

4.4.2. NAND Gate

Same as the AND gate except that $\mathrm{K}^{xy,wz} tu_a^{xy}(t1) td_b^{wz}(t2)$ is subtracted from $td_c^{xy}(t1+d)$ and $tu_c^{wz}(t2+d)$.

4.4.3. OR Gate

For a falling event $td_a^{xy}(t1)$, all of the upward events $tu_b^{wz}(t2)$ that come from the other gate input and where $t2$ is within a gate delay time after $t1$ $(t1 < t2 \le t1+d)$ are subject to glitch filtering. That is, $\mathrm{K}^{xy,wz} td_a^{xy}(t1) tu_b^{wz}(t2)$ is subtracted from both $td_c^{xy}(t1+d)$ and $tu_c^{wz}(t2+d)$.

4.4.4. NOR Gate

Same as the OR gate except that $\mathrm{K}^{xy,wz} tu_a^{xy}(t1) td_b^{wz}(t2)$ is subtracted from $tu_c^{xy}(t1+d)$ and $td_c^{wz}(t2+d)$.

4.4.5. EXOR Gate

For a falling (rising) event $td_a^{xy}(t1)$ ($tu_a^{xy}(t1)$), all of the upward (downward) events $tu_b^{wz}(t2)$ ($td_b^{wz}(t2)$) that come from the other gate input and where $t2$ is within a gate delay time after $t1$ ($t1 < t2 \le t1 + d$) are subject to glitch filtering. That is, $K^{xy,wz} td_a^{xy}(t1) tu_b^{wz}(t2)$ ($K^{xy,wz} tu_a^{xy}(t1) td_b^{wz}(t2)$) is subtracted from $td_c^{xy}(t1+d)$ ($tu_c^{xy}(t1+d)$) and $tu_c^{wz}(t2+d)$ ($td_c^{wz}(t2+d)$).

For a falling (rising) event $td_a^{xy}(t1)$ ($tu_a^{xy}(t1)$), all of the downward (upward) events $td_b^{wz}(t2)$ ($tu_b^{wz}(t2)$) that come from the other gate input and where $t2$ is within a gate delay time after $t1$ ($t1 < t2 \le t1 + d$) are subject to glitch filtering. That is, $K^{xy,wz} td_a^{xy}(t1) td_b^{wz}(t2)$ ($K^{xy,wz} tu_a^{xy}(t1) tu_b^{wz}(t2)$) is subtracted from $td_c^{xy}(t1+d)$ ($tu_c^{xy}(t1+d)$) and $td_c^{wz}(t2+d)$ ($tu_c^{wz}(t2+d)$).

5. SEQUENTIAL CIRCUITS

VLSI circuits are sequential, i.e., they contain memory elements or flip-flops. This sequential nature of VLSI circuits makes estimation more complicated than for combinational circuits. Besides spatial correlations, now temporal correlations come into play.

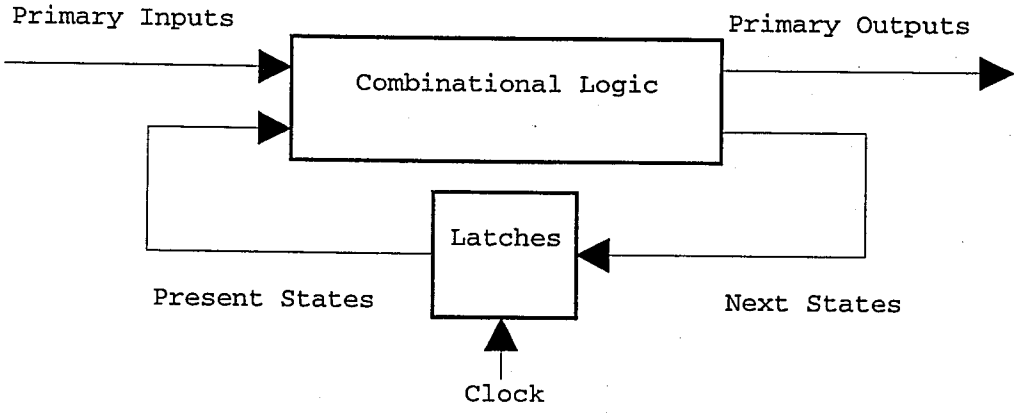

Figure 5.1. A synchronous sequential circuit

In a sequential circuit (Figure 5.1) there is an extra set of inputs, the present-state lines. The static probabilities of the present state lines are spatially correlated. Therefore knowledge of present-state probabilities is required as opposed to present-state line probabilities to exactly calculate the switching activity in a sequential machine. Exact and approximate methods for estimating the switching activity in sequential circuits are proposed in [9], [4]. The following section is a brief discussion on what has been proposed, and next the approach in this work is mentioned.

5.1. The Exact Method

Once state probabilities are computed it is easy to compute the switching activity with the help of the symbolic equations. The state probabilities are dependent on the state transition graph (STG) of the circuit. For a STG with K states the probability of being at a

state s_i is the sum of the product of the probabilities of the states s_j whose next state is s_i with the probability of the corresponding primary inputs to make the transition $s_j \rightarrow s_i$, where $i, j \in [1, K]$. Given K states we obtain K equations of which any one can be derived from the remaining $K-1$ equations.

$$P(s_i) = \sum_{S_j .such.that .next .state=s_i} P(s_j) P(primary\_inputs)$$

The sum of the probabilities of the states is one.

$$\sum_{i=1}^{K} P(s_i) = 1$$

Now we can obtain the probability of each state by solving these equations.

As an example, consider the STG in Figure 5.2 for which the following equations are obtained

$$P(R) = P_x(1) \times P(A)$$
$$P(A) = P_x(1) \times P(R) + P_x(1) \times P(B) + P_x(0) \times P(C)$$
$$P(B) = P_x(0) \times P(R) + P_x(0) \times P(A)$$
$$P(R) + P(A) + P(B) + P(C) = 1$$

Solving these equations the state probabilities are

$$P(R) = 1/6, P(A) = 1/3, P(B) = 1/4, P(C) = 1/4$$

Knowing these probabilities we can compute switching activities for a circuit realized based on this STG. However computing the state probabilities is an expensive task in terms of time and memory. For a finite state machine having n flip-flops there are 2^n states to be determined.

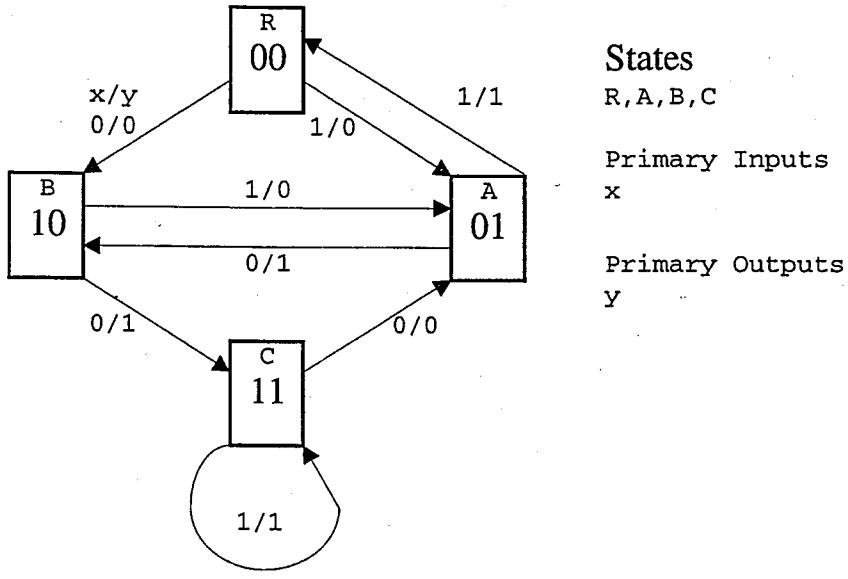

Figure 5.2. Example STG given in [9]

If we want to realize the STG in Figure 5.3 with D flip-flops, we come up with the following equations for the next state lines and output

$$D_1 = x \bullet Q_1 \bullet Q_2 + \bar{x} \bullet Q_1 \bullet \overline{Q_2} + \bar{x} \bullet \overline{Q_1}$$

$$D_2 = x \bullet \overline{Q_1} \bullet \overline{Q_2} + Q_1$$

$$y = \bar{x} \bullet \overline{Q_1} \bullet Q_2 + \bar{x} \bullet Q_1 \bullet \overline{Q_2} + x \bullet Q_2$$

where D_1 and D_2 denote the next state lines, Q_1 and Q_2 denote the present state lines, x is the primary input, and y is the primary output.

From the state probabilities, the present state lines probabilities are computed as follows:

$$P(Q_1 = 0) = P_{Q_1}(0) = P(R) + P(A) = P(00) + P(01)$$

$$P(Q_1 = 0) = 1/6 + 1/3 = 1/2$$

$$P(Q_1 = 1) = P_{Q_1}(1) = P(B) + P(C) = P(10) + P(11)$$

$$P(Q_1 = 1) = 1/4 + 1/4 = 1/2$$

$$P(Q_2 = 0) = P_{Q_2}(0) = P(R) + P(B) = P(00) + P(10)$$

$$P(Q_2 = 0) = 1/6 + 1/4 = 5/12$$

$$P(Q_2 = 1) = P_{Q_2}(1) = P(A) + P(C) = P(01) + P(11)$$

$$P(Q_2 = 1) = 1/3 + 1/4 = 7/12$$

Computing the probability of Q_1Q_2 (state 11) using the line probabilities

$$P_{Q_1Q_2}(1) = P_{Q_1}(1)P_{Q_2}(1) = 1/2 \times 7/12 = 7/24$$

It is not equal to the state probability $P(11) = 6/24$. This is because present state lines Q_1 and Q_2 are correlated. In [9], it is shown that power estimation using state line probabilities instead of state probabilities does not introduce a significant error (3% on average). Computing the state line probabilities rather than state probabilities is a good alternative to computing the state probabilities.

5.2. Computing Present-state Line Probabilities

The approximation framework is based on determining the line probabilities without considering the STG. The method is based on solving a nonlinear system of equations that is given by the combinational logic implementing the next state function of the sequential circuit.

As an example consider the present state line equation for the STG in Figure 5.2.

$$D_1 = x \bullet Q_1 \bullet Q_2 + \bar{x} \bullet Q_1 \bullet \overline{Q_2} + \bar{x} \bullet \overline{Q_1}$$

$$D_2 = x \bullet \overline{Q_1} \bullet \overline{Q_2} + Q_1$$

where D_1 and D_2 are the next state lines, Q_1 and Q_2 are the present state lines. Assuming the probability of input x being a 1 is 0.5 we obtain the equations

$$d_1 = 0.5 \times q_1 \times q_2 + (1-0.5) \times q_1 \times (1-q_2) + (1-0.5) \times (1-q_1) = 0.5$$

$$d_2 = 0.5\times(1-q_1)\times(1-q_2)+q_1 = 0.5+0.5\times q_1 - 0.5\times q_2 + 0.5\times q_1 \times q_2$$

Setting $d_1 = q_1$ and $d_2 = q_2$ and solving the equations gives us

$$q_1 = 0.5$$

$$q_2 = 0.6$$

The results of the approximation are shown in Table 5.1. If we had assumed a probability of 0.5 for each present state line, the error for q_1 is 0% and for q_2 is 14%.

Table 5.1. State line probabilities

line	Exact	approx.	err.(%)
q1	0,5	0,5	0,0
q2	0,583	0,6	2,9

5.3. Computing Transition Probabilities

The approach mentioned in [4], [9] uses symbolic simulation equations to compute the switching activity in a circuit. But this approach has high memory requirements. Even in [2], for large circuits Monte-Carlo simulation is used. Moreover it is not applicable to the local BDD approach that is used to overcome the high memory requirements of large circuits. Therefore a method based on the computation and propagation of transition probabilities is used. First present-state line probabilities are computed with the method mentioned above. Then using state line probabilities transition probabilities of the present-state lines are computed. The last step is propagating these probabilities through the primary outputs. This method accounts for temporal correlations, but ignores the spatial correlation between the present-state lines.

Dealing with sequential circuits is dealing with temporal correlations. Therefore we have to consider transition probabilities instead of static probabilities. We want to know the switching activity at the output of the gate in Figure 5.3. Assume that the gate is part of

the realization of the sample STG given in Figure 5.2. x is the temporally and spatially uncorrelated primary input that has a probability of 0.5 being a one. Since Q_2 is temporally dependent we need to know its transition probabilities.

Figure 5.3. A sample gate

The $0 \rightarrow 1$ transition probability at present-state line Q_2 is the probability of the function $f = \overline{Q_2}(t) \bullet Q_2(t+1)$ evaluating to a 1. Since $Q_2(t+1) = D_2(t)$

$$f = \overline{Q_2}(t) \bullet (x \bullet \overline{Q_1}(t) \bullet \overline{Q_2}(t) + Q_1(t))$$

$$f = x \bullet \overline{Q_1}(t) \bullet \overline{Q_2}(t) + Q_1(t) \bullet \overline{Q_2}(t)$$

$$P_f(1) = x(1-q_1)(1-q_2) + q_1(1-q_2)$$

$$P_f(1) = 0.5(1-0.5)(1-0.6) + 0.5(1-0.6) = 0.3$$

Using this procedure transition probabilities for Q_1 and Q_2 are

$$P_{Q_1}(0 \rightarrow 0) = 0.25, P_{Q_2}(0 \rightarrow 0) = 0.1$$

$$P_{Q_1}(0 \rightarrow 1) = 0.25, P_{Q_2}(0 \rightarrow 1) = 0.3$$

$$P_{Q_1}(1 \rightarrow 0) = 0.25, P_{Q_2}(1 \rightarrow 0) = 0.3$$

$$P_{Q_1}(1 \rightarrow 1) = 0.25, P_{Q_2}(1 \rightarrow 1) = 0.3$$

The probability of a $0 \rightarrow 1$ transition occurring at f can be computed from the input transitions with the method proposed in Section 3.2 as follows

$$f(0 \rightarrow 1) = \overline{(x(t)Q_2(t))}(x(t+1)Q_2(t+1))$$

$$f(0 \rightarrow 1) = \bar{x}(t)x(t+1)Q_2(t+1) + x(t)x(t+1)\overline{Q_2}(t)Q_2(t+1)$$

$$P_f(0 \rightarrow 1) = P_x(0 \rightarrow 1)(P_{Q_2}(1 \rightarrow 1) + P_{Q_2}(0 \rightarrow 1)) + P_x(1 \rightarrow 1)P_{Q_2}(0 \rightarrow 1)$$

$$P_f(0 \rightarrow 1) = 0.25 \times (0.3 + 0.3) + 0.25 \times 0.3 = 0.225$$

Calculation of the transition probabilities at the present-state lines can be done with the algorithm presented in Section 3.3. But it is not easy to implement this with the local BDD approach. In this work, instead, we just calculate the line probabilities. Using the line probabilities we directly calculate the transition probabilities as follows:

$$PS_i(x \rightarrow y) = PS_i(x)PS_i(y)\ ,\ x, y \in \{0,1\}$$

where $PS_i(x \rightarrow y)$ is the probability of the transition of line i from x to y, $PS_i(x)$ is the line probability of line i being x. This method does not account for temporal/spatial correlations properly but it is better then assuming uniform probabilities for all present state lines as it is done in [4].

6. IMPLEMENTATION AND RESULTS

The concepts mentioned in Sections 4 and 5 are implemented in a program called EPOSIM. The program is about 13.000 lines and is written in C. It accepts a circuit description code compiled by EPOCOM. EPOCOM accepts structural level VHDL code files and compiles them into an intermediate format. EPOSIM works with a library file in text format.

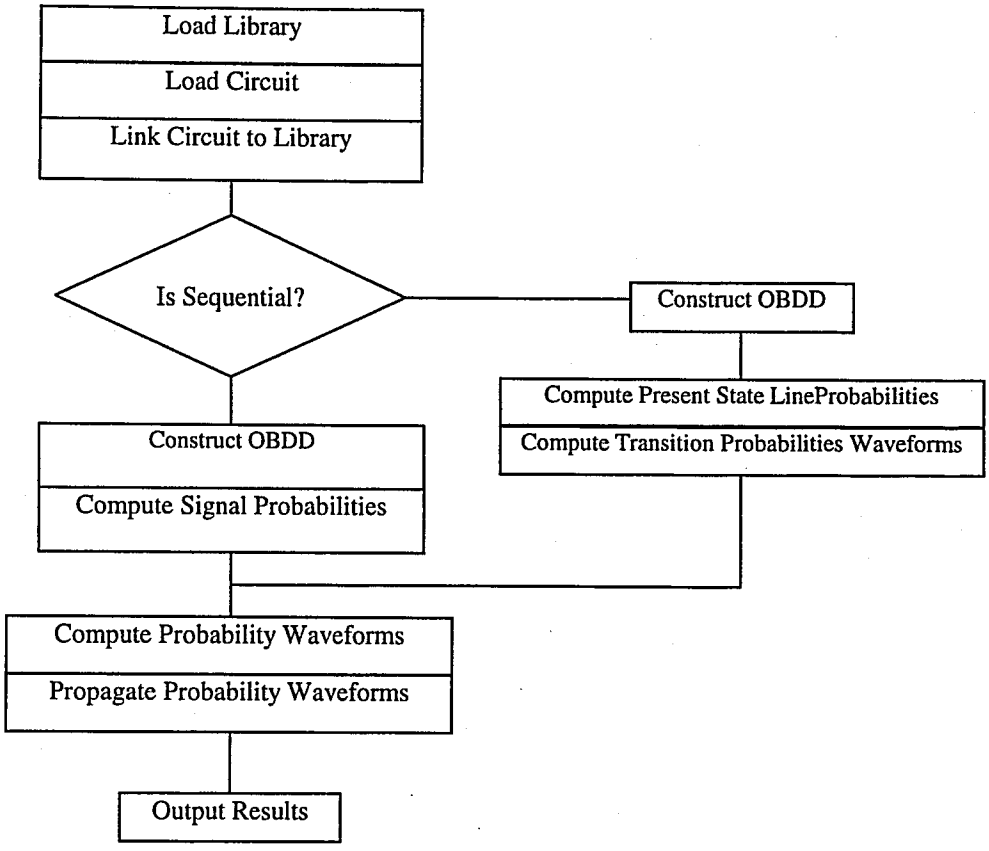

Figure 6.1. Flow-chart of EPOSIM

6.1. Usage

EPOCOM takes as input structural level VHDL code. The invocation of the program is as follows.

\epocom filename

where filename is the file containing VHDL code. EPOCOM will generate a directory with the same name as filename (without the extension) and will place the compiled code under this directory. Now circuits can be simulated with EPOSIM as follows:

\eposim dirname libname

where 'dirname' is the directory where the compiled code lies and 'libname' is the library file to be linked up with the circuit. After the program is invoked a window will prompt with two menus to select: RUN and SETUP. In the SETUP menu you can adjust EPOSIM to treat the circuit as combinational or sequential. You can also select the BDD approach (global / local) to be used. By default, circuits are assumed to be combinational and are simulated using the local BDD approach.

When the RUN menu is selected a list of available circuits will be prompted on the screen. Selecting a circuit will start the simulation. Simulation results will be written to a file named

_circuitname.txt

where 'circuitname' is the name of the simulated circuit.

EPOSIM reports total power consumption in microwatts at 1 MHz. The supply voltage is 5 V. For example if the resultant power is $10\mu W / MHz$ the power consumption at 100 MHz is

$$Power = 100Mhz \times 10\mu W / MHz = 1000\mu W$$

6.2. Simulation Results

Simulations in Table 6.1 are done with a library prepared with the results of [13], [14]. A library for HP 1.2μ technology is prepared. Gate delay values are taken from [14]. H-Spice simulation results are taken from [13].

Table 6.1. Simulation results I

	POWER (uW)			
	EPOSIM	H-SPICE	ERROR(%)	Circuit description
RCA_FA	184 uW	224 uW	18	Full adder
RCA2	426 uW	704 uW	40	2 bit ripple carry adder
RCA4	978 uW	1451 uW	32	4 bit ripple carry adder
CLA4	2476 uW	3000 uW	17	4 bit carry look-ahead adder
CLA8	5370 uW	6280 uW	14	8 bit carry look-ahead adder
CSKA16	4072 uW	4446 uW	8	16 bit carry skip adder
WTM4	4830 uW	6136 uW	21	4 bit wallace tree multiplier
Average			21	

The results in Table 6.2 are simulation results that are run with a library prepared with HP 1.2μ technology, but different transistor sizing. As opposed to the results in Table 6.1, the clock frequency is fixed and it is 100 MHz.

Table 6.2. Simulation results II

	POWER (uW)		
	HSPICE	EPOSIM	ERROR (%)
RCA2	428	305	29
RCA4	1055	700	34
RCA8	2280	1547	32
CLA4	1455	986	32
CSKA8	2440	1680	31
PPA4	1225	816	33
WTM4	3292	2332	29
Average			31

If we look at the results we see that they are in error. The results reported in [7] for TPS have an average error rate of 5%. This is a very good result in the sense that TPS can accurately estimate the switching activity. However the exact error rate should be much higher because of the fact that it estimates the power due to the charging and discharging of the load capacitances. There is also short circuit power consumption as mentioned in Section 2, which cannot be avoided. It can be reduced to 15% of the total power consumption with proper design but without it might constitute up to 50% of the total power consumption.

There is another source of error regarding the correctness of the gate models. Looking at the results in [7] this cannot be understood because the models used are same for the different simulators that are used for comparison. This error will become apparent when compared to spice results. Considering these sources of errors the results are satisfactory.

6.2.1. Speed vs. Accuracy

With the increasing number of primary inputs of a circuit the memory required to construct the BDD of the circuit increases significantly. In addition, traversing the BDD becomes time consuming. To avoid these difficulties EPOSIM uses the local BDD approach.

Table 6.3 shows a comparison of the two methods. The simulations are run at a frequency of 100 MHz using the ES 1.0μ library. From the results we can state that for circuits having less than 20 primary inputs the global BDD approach should be preferred. Although the accuracy does not change too much for those circuits, using the local BDD approach can even increase the run time of the simulation. The four bit (eight primary input) wallace tree multiplier (wtm4) is a good example for this situation.

6.2.2. Sequential Circuits

Three different simulations methods are tested using the ISCAS89 benchmark circuits S27 and S298. Table 6.4 presents the number of inputs, outputs, flip-flops and gates in the circuits. The simulation results are presented in Table 6.5. The first method is the symbolic simulation method used in [9] that makes use of the global BDD approach. The second method is the uniform line probability method used in [4]. The third one is the method implemented in EPOSIM that was mentioned in the previous section. The simulations are run at a frequency of 100 MHz using the MIETEC 0.7μ library.

When the number of inputs and flip-flops is small, as it is the case with S27, there is no speed advantage in using EPOSIM. Moreover it has an estimation result even worse

than the uniform probability approach. But when we look at the results of S298 EPOSIM outperforms the uniform probability approach. EPOSIM has an error rate of 31% where as the uniform probability approach has 96%. The error rate for EPOSIM is still high that shows that more work has to be done to improve its capability in handling sequential circuits.

Table 6.3. Global / Local BDD comparison

	Local OBDD		Global OBDD			
	Power (uW)	Time (s)	Power (uW)	Time (s)	Diff. (%)	Speedup
RCA4	1234	1	1237	0	0,3	
RCA8	2525	1	2542	28	0,6	28,0
RCA12	3817	3	3812	9000	0,1	3000,0
CLA4	2129	3	2131	1	0,1	0,3
CLA8	4322	7	4339	44	0,4	6,3
CSKA4	1548	1	1554	1	0,4	1,0
CSKA8	2857	13	2877	32	0,7	2,5
CSLA4	3222	7	3384	1	4,8	0,1
CSLA8	4727	59	4919	39	3,9	0,7
WTM4	3029	236	3068	1	1,3	0,0

Table 6.4. The ISCAS 89 benchmark circuits

Circuit	#inputs	#outputs	#flip-flops	#gates
S27	4	1	3	10
S298	3	6	14	119

Table 6.5. Power and time results for the ISCAS 89 circuits

	1		2		3	
	Symbolic Simulation		Uniform Probability		EPOSIM	
Circuit	Power (uW)	Time (s)	Power (uW)	Time (s)	Power (uW)	Time (s)
S27	1757	1	1598	1	1439	1
S298	15557	1680	30524	16	20500	87

7. CONCLUSION

Today's electronic design automation tools need efficient power estimation techniques. To address this need several techniques at different abstraction levels have been developed. Among them, logic-level techniques provide a good balance between accuracy and efficiency.

In this thesis, a logic-level probabilistic simulator is developed and tested for various circuits. The simulator is based on tagged probabilistic simulation (TPS) and it is extended to handle sequential circuits. The reliability of the simulator is validated with HSPICE simulations.

TPS correctly accounts for reconvergent fanout and glitches that is a bottleneck of probabilistic techniques. This approach when used with the local BDD approach requires less memory and runs faster than symbolic simulation. The local BDD technique cannot be applied to symbolic equation based techniques used to estimate power in sequential circuits. To overcome this difficulty a method based on the computation and propagation of transition probabilities is used. First present-state line transition probabilities are computed. Given the primary input transition probabilities, transition probabilities are propagated through the primary outputs. This method accounts for temporal correlations, but ignores the spatial correlation between the present-state lines.

Comparison with HSPICE has revealed that accurate estimation of switching activity is not sufficient. One of the future studies might be accurate modeling of gate capacitances to increase accuracy. Another study might be developing methods that can account for the short circuit power consumption. One way of modeling the short circuit power can be the incorporation of the effect of this power to the self loading capacitance of a gate. This makes sense because short circuit power is also a function of the switching activity at the output of the gate. With small changes EPOSIM can also be used to estimate delay in static CMOS circuits. This can be achieved by using real delay models that take into account the loading effect of the fanout capacitances on the delay value of a gate. At the moment

EPOSIM uses the unit delay model and can report the occurrence time of the last event on any of the primary outputs of the circuit. The latest among them will give the delay of the circuit.

APPENDIX A: SWITCHING ACTIVITY ESTIMATION USING TPS

A solution with tagged probabilistic simulation of the multiplexer circuit in Figure A.1. (The sample circuit in [7]) is given in this section. First, the signal probability at each node is computed from which each tagged waveform probability is calculated. Then probability waveforms are propagated to the output.

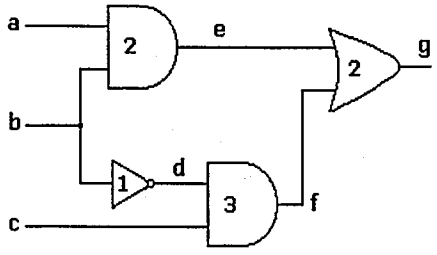

Figure A.1. An example circuit

A.1. Computation of Signal Probabilities

We assume that the inputs to the circuit are temporally and spatially uncorrelated and each has a signal probability equal to 0.5 . The delay of each gate is as it is written on it in Figure A.1.

$$P_a(1) = P_b(1) = P_c(1) = 0.5$$

$$sp_a^{11} = sp_b^{11} = sp_c^{11} = 0.5 \times 0.5 = 0.25$$

$$sp_a^{10} = sp_b^{10} = sp_c^{10} = 0.5 \times (1-0.5) = 0.25$$

$$sp_a^{01} = sp_b^{01} = sp_c^{01} = 0.5 \times (1-0.5) = 0.25$$

$$sp_a^{00} = sp_b^{00} = sp_c^{00} = (1-0.5) \times (1-0.5) = 0.25$$

$$d = \overline{b}$$

$$P_d(1) = P_b(0) = 1 - P_b(1) = 1 - 0.5 = 0.5$$

$$sp_d^{11} = 0.5 \times 0.5 = 0.25$$

$$sp_d^{10} = 0.5 \times (1-0.5) = 0.25$$

$$sp_d^{01} = (1-0.5) \times 0.5 = 0.25$$

$$sp_d^{00} = (1-0.5) \times (1-0.5) = 0.25$$

$$e = a \bullet b$$

$$P_e(1) = P_a(1) P_b(1) = 0.5 \times 0.5 = 0.25$$

$$sp_e^{11} = 0.25 \times 0.25 = 1/16$$

$$sp_e^{10} = 0.25 \times (1-0.25) = 3/16$$

$$sp_e^{01} = (1-0.25) \times 0.25 = 3/16$$

$$sp_e^{00} = (1-0.25) \times (1-0.25) = 9/16$$

$$f = d \bullet c = \overline{b} \bullet c$$

$$P_f(1) = P_d(1) P_c(1) = 0.5 \times 0.5 = 0.25$$

$$sp_f^{11} = 0.25 \times 0.25 = 1/16$$

$$sp_f^{10} = 0.25 \times (1-0.25) = 3/16$$

$$sp_f^{01} = (1-0.25) \times 0.25 = 3/16$$

$$sp_f^{00} = (1-0.25) \times (1-0.25) = 9/16$$

$$g = e + f = a \bullet b + \overline{b} \bullet c$$

$$P_g(1) = P_a(1) P_b(1) + P_b(0) P_c(1) = 0.5 \times 0.5 + (1-0.5) \times 0.5 = 0.5$$

$$sp_g^{11} = 0.5 \times 0.5 = 0.25$$

$$sp_g^{10} = 0.5 \times (1-0.5) = 0.25$$

$$sp_g^{01} = (1-0.5) \times 0.5 = 0.25$$

$$sp_g^{00} = (1-0.5) \times (1-0.5) = 0.25$$

A.2. Computation of Correlation Coefficients

Table A.1 shows the correlation coefficients of nodes *e*, *f*, and *g*. Since *e* and *f* have uncorrelated inputs their coefficients are 1. Node *g*'s inputs are *e* and *f*, and they both have *b*

as an input in common. So g's inputs are correlated. Correlation coefficients for node g are computed with the formula below as follows:

$$K_g^{xy,wz} = \frac{P(e = x \wedge f = w)P(e = y \wedge f = z)}{P_e(x)P_f(w)P_e(y)P_f(z)}$$

where $x, y, w, z \in \{0,1\}$.

$$P(e = 1 \wedge f = 1) = P_{e \bullet f}(1) = P_{a \bullet b \bullet \bar{b} \bullet c}(1) = P_0(1) = 0$$

$$P(e = 1 \wedge f = 0) = P_{e \bullet \bar{f}}(1) = P_{a \bullet b \bullet \overline{\bar{b} \bullet c}}(1) = P_{a \bullet b}(1) = P_a(1)P_b(1) = 0.5 \times 0.5 = 0.25$$

$$P(e = 0 \wedge f = 1) = P_{\bar{e} \bullet f}(1) = P_{\overline{a \bullet b \bullet \bar{b} \bullet c}}(1) = P_{\overline{\bar{b} \bullet c}}(1) = P_b(0)P_c(1) = (1 - 0.5) \times 0.5 = 0.25$$

$$P(e = 0 \wedge f = 0) = P_{\bar{e} \bullet \bar{f}}(1) = P_{\overline{e+f}}(1) = P_{e+f}(0) = P_g(0) = 1 - 0.5 = 0.5$$

$$K_g^{00,00} = (0.50 \times 0.50)/(0.75 \times 0.75 \times 0.75 \times 0.75) = 64/81 = 0.79$$

$$K_g^{00,01} = (0.50 \times 0.25)/(0.75 \times 0.75 \times 0.75 \times 0.25) = 32/27 = 1.19$$

$$K_g^{00,10} = (0.25 \times 0.50)/(0.75 \times 0.25 \times 0.75 \times 0.75) = 32/27 = 1.19$$

$$K_g^{00,11} = (0.25 \times 0.25)/(0.75 \times 0.25 \times 0.75 \times 0.25) = 16/9 = 1.78$$

$$K_g^{01,00} = (0.50 \times 0.25)/(0.75 \times 0.75 \times 0.25 \times 0.75) = 32/27 = 1.19$$

$$K_g^{01,01} = (0.50 \times 0.00)/(0.75 \times 0.75 \times 0.25 \times 0.25) = 0$$

$$K_g^{01,10} = (0.25 \times 0.25)/(0.75 \times 0.25 \times 0.25 \times 0.75) = 16/9 = 1.78$$

$$K_g^{01,11} = (0.25 \times 0.00)/(0.75 \times 0.25 \times 0.25 \times 0.25) = 0$$

$$K_g^{10,00} = (0.25 \times 0.50)/(0.25 \times 0.75 \times 0.75 \times 0.75) = 32/27 = 1.19$$

$$K_g^{10,01} = (0.25 \times 0.25)/(0.25 \times 0.75 \times 0.75 \times 0.25) = 16/9 = 1.78$$

$$K_g^{10,10} = (0.00 \times 0.50)/(0.25 \times 0.25 \times 0.75 \times 0.75) = 0$$

$$K_g^{10,11} = (0.00 \times 0.25)/(0.25 \times 0.75 \times 0.25 \times 0.25) = 0$$

$$K_g^{11,00} = (0.25 \times 0.25)/(0.25 \times 0.75 \times 0.25 \times 0.75) = 16/9 = 1.78$$

$$K_g^{11,01} = (0.25 \times 0.00)/(0.25 \times 0.75 \times 0.75 \times 0.25) = 0$$

$$K_g^{11,10} = (0.00 \times 0.25)/(0.25 \times 0.25 \times 0.25 \times 0.75) = 0$$

$$K_g^{11,11} = (0.00 \times 0.00)/(0.25 \times 0.25 \times 0.25 \times 0.25) = 0$$

A.3. Waveform propagation

w_g^{11}: Table A.2 is a brief description of the input joint tagged waveforms that produce the output tagged waveform w_g^{11}.

Table A.1. Correlation coefficients

[xy],[wz]	e	f	g
00,00	1,00	1,00	0,79
00,01	1,00	1,00	1,19
00,10	1,00	1,00	1,19
00,11	1,00	1,00	1,78
01,00	1,00	1,00	1,19
01,01	1,00	1,00	0,00
01,10	1,00	1,00	1,78
01,11	1,00	1,00	0,00
10,00	1,00	1,00	1,19
10,01	1,00	1,00	1,78
10,10	1,00	1,00	0,00
10,11	1,00	1,00	0,00
11,00	1,00	1,00	1,78
11,01	1,00	1,00	0,00
11,10	1,00	1,00	0,00
11,11	1,00	1,00	0,00

#1, 2, 3

d = 2, [xy] = 01, [wz] = 10

t = 2

$tu_g(2+2)$ = 1.78 x { 3/16 x (3/16 - 3/16) + 0 x (3/16 - 0) - 3/16 x 0 }

$tu_g(4) = 0$

t = 3

$td_g(3+2)$ = 1.78 x { 0 x (3/16 - 1/16) + 2/16 x (3/16 - 3/16) - 0 x 2/16 }

$td_g(5) = 0$

t = 4

$$td_g(4+2) = 1.78 \text{ x } \{ 0 \text{ x } (3/16 - 0) + 1/16 \text{ x } (3/16 - 3/16) - 0 \text{ x } 1/16 \}$$

$$td_g(6) = 0$$

#4, 5, 6

d = 2, [xy] = 10, [wz] = 01

t = 2

$$td_g(2+2) = 1.78 \text{ x } \{ 3/16 \text{ x } (3/16 - 0) + 0 \text{ x } (3/16 - 0) - 3/16 \text{ x } 0 \}$$

$$td_g(4) = 1/16$$

Table A.2. Tag w_g^{11}

Inputs	Coef.	Events				
e , f	K	#	t	up/down	node	prob.
00,11	1,78	-	-	-	-	-
01,10	1,78	1	2	u	e	3/16
		2	3	d	f	2/16
		3	4	d	f	1/16
01,11	0	-	-	-	-	-
10,01	1,78	4	2	d	e	3/16
		5	3	u	f	1/16
		6	4	u	f	2/16
10,11	0	-	-	-	-	-
11,00	1,78	7	3	u	f	1/16
		8	4	d	f	1/16
11,01	0	-	-	-	-	-
11,10	0	-	-	-	-	-
11,11	0	-	-	-	-	-

t = 3

$$tu_g(3+2) = 1.78 \text{ x } \{ 0 \text{ x } (3/16 - 0) + 1/16 \text{ x } (3/16 - 0) - 0 \text{ x } 1/16 \}$$

$$tu_g(5) = 1/48$$

t = 4

$$tu_g(4+2) = 1.78 \text{ x } \{ 0 \text{ x } (3/16 - 1/16) + 2/16 \text{ x } (3/16 - 0) - 0 \text{ x } 1/16 \}$$

$$tu_g(6) = 2/48$$

#7, 8

d = 2, [xy] = 11, [wz] = 00

$$2t = 3$$

$$tu_g(3+2) = 1.78 \text{ x } \{ \text{ 0 x (9/16 - 0) + 1/16 x (1/16 - 1/16) - 0 x 1/16 } \}$$

$$tu_g(5) = 0$$

$$t = 4$$

$$td_g(4+2) = 1.78 \text{ x } \{ \text{ 0 x (9/16 - 1/16) + 1/16 x (1/16 - 1/16) - 0 x 1/16 } \}$$

$$td_g(6) = 0$$

The resulting probability waveform is shown in Figure A.2.

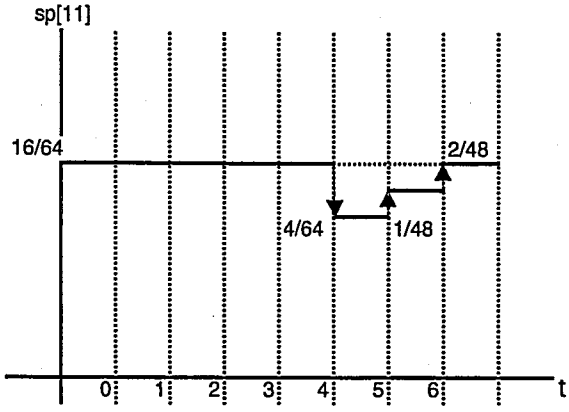

Figure A.2. Tag w_g^{11} of node g

w_g^{01}: Table A.3. is a brief description of the input joint tagged waveforms that produce the output tagged waveform w_g^{01}.

Table A.3. Tag w_g^{01}

Inputs	Coef.	Events				
e , f	K	#	t	up/down	node	prob.
00,01	1,19	1	3	u	f	1/16
		2	4	u	f	2/16
01,00	1,19	3	2	u	e	3/16
		4	3	u	f	1/16
		5	4	d	f	1/16
01,01	0	-	-	-	-	-

#1, 2

d = 2, [xy] = 00, [wz] = 01

t = 3

$tu_g(3+2) = 1.19 \times \{ 0 \times (3/16 - 0) + 1/16 \times (9/16 - 0) - 0 \times 1/16 \}$

$tu_g(5) = 1/24$

t = 4

$tu_g(4+2) = 1.19 \times \{ 0 \times (3/16 - 1/16) + 2/16 \times (9/16 - 0) - 0 \times 2/16 \}$

$tu_g(6) = 2/24$

#3, 4, 5

d = 2, [xy] = 01, [wz] = 00

t = 2

$tu_g(2+2) = 1.19 \times \{ 3/16 \times (9/16 - 0) + 0 \times (3/16 - 0) - 3/16 \times 0 \}$

$tu_g(4) = 1/8$

t = 3

$tu_g(3+2) = 1.19 \times \{ 0 \times (9/16 - 0) + 1/16 \times (3/16 - 3/16) - 0 \times 1/16 \}$

$tu_g(5) = 0$

t = 4

$td_g(4+2) = 1.19 \times \{ 0 \times (9/16 - 0) + 1/16 \times (3/16 - 3/16) - 0 \times 1/16 \}$

$td_g(6) = 0$

w_g^{10}: Table A.4 is a brief description of the input joint tagged waveforms that produce the output tagged waveform w_g^{10}.

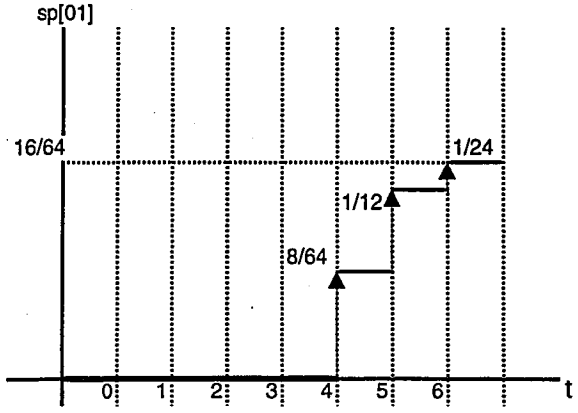

Figure A.3. Tag w_g^{01} of node g

#1, 2

d = 2, [xy] = 00, [wz] = 10

t = 3

$td_g(3+2) = 1.19 \text{ x } \{ 0 \text{ x } (3/16 - 1/16) + 2/16 \text{ x } (9/16 - 0) - 0 \text{ x } 2/16 \}$

$td_g(5) = 2/24$

t = 4

$td_g(4+2) = 1.19 \text{ x } \{ 0 \text{ x } (3/16 - 0) + 1/16 \text{ x } (9/16 - 0) - 0 \text{ x } 1/16 \}$

$td_g(6) = 1/24$

Table A.4. Tag w_g^{10}

Inputs	Coef.	Events				
e , f	K	#	t	up/down	node	prob.
00,10	1,19	1	3	d	F	2/16
		2	4	d	F	1/16
10,00	1,19	3	2	d	E	3/16
		4	3	u	F	1/16
		5	4	d	F	1/16
10,10	0	-	-	-	-	-

#3, 4, 5

d = 2, [xy] = 10, [wz] = 00

t = 2

$td_g(2+2) = 1.19 \text{ x } \{ 3/16 \text{ x } (9/16 - 0) + 0 \text{ x } (3/16 - 3/16) - 3/16 \text{ x } 0 \}$

$td_g(4) = 1/8$

t = 3

$tu_g(3+2) = 1.19 \text{ x } \{ 0 \text{ x } (9/16 - 0) + 1/16 \text{ x } (3/16 - 0) - 0 \text{ x } 1/16 \}$

$tu_g(5) = 1/72$

t = 4

$td_g(4+2) = 1.19 \text{ x } \{ 0 \text{ x } (9/16 - 0) + 1/16 \text{ x } (3/16 - 0) - 0 \text{ x } 1/16 \}$

$td_g(6) = 1/72$

w_g^{00}: Table A.5 is a brief description of the input joint tagged waveforms that produce the output tagged waveform w_g^{00}.

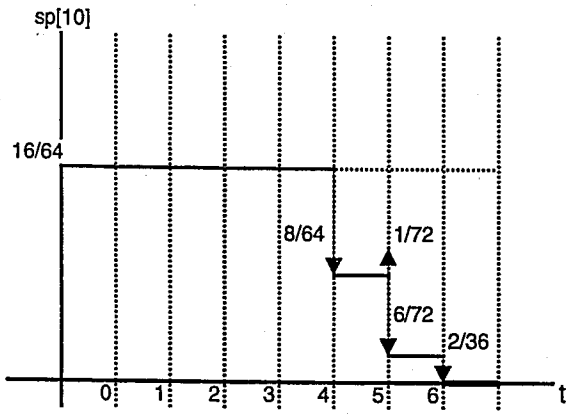

Figure A.4. Tag w_g^{10} of node g

Table A.5. Tag w_g^{00}

Inputs	Coef.	Events				
e , f	K	#	t	up/down	node	prob.
00,00	0,79	1	3	u	f	1/16
		2	4	d	f	1/16

#1, 2

d = 2, [xy] = 00, [wz] = 00

$tu_g(5)$ = 0.79 x { 0 x (9/16 - 0) + 1/16 x (9/16 - 0) - 0 x 1/16 }= 1/36

$td_g(6)$ = 0.79 x { 0 x (9/16 - 0) + 1/16 x (9/16 - 0) - 0 x 1/16 }= 1/36

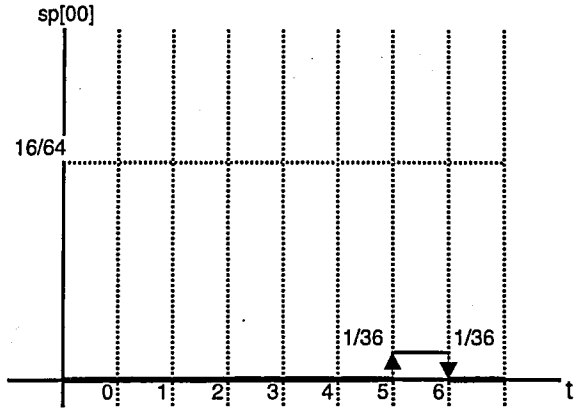

Figure A.5. Tag w_g^{00} of node g

APPENDIX B: A SAMPLE LIBRARY

At the beginning of file the name of the library and the number of components residing in the library should be given. Comments can be written after a double minus sign as it is done in VHDL.

```
LibraryName=es2                -- comment
NumberOfComponents=3
```

For each component in the library the name, delay and port number should be written.

```
<inv>
Delay=15          -- glitch delay
NumberOfPorts=2
```

For each port of a gate the name, delay and capacitance value should be written.

```
[out1]
Delay=15          -- gate delay
Capacitance=0
```

The capacitance values are in fantofarads (fF) and the delay values are in picoseconds (ps). Below is a sample library file with 3 components.

```
LibraryName=es2          -- the name of
                         -- the library
NumberOfComponents=3     -- number gates defined
                         -- in the library

-- 1
```

```
<inv>       -- the gate name should be
            -- between '<' gatename '>'
            -- without any blanks
Delay=15
NumberOfPorts=2

[in1]       -- name of a port should be
            -- between '[' portname ']'
            -- without any blanks
Delay=0
Capacitance=63

[out1]
Delay=15
Capacitance=0

-- 2

<and2>
Delay=50
NumberOfPorts=3

[in1]
Delay=0
Capacitance=32

[in2]
Delay=0
Capacitance=37

[out1]
Delay=50
Capacitance=0

-- 3
```

```
<and3>
Delay=50
NumberOfPorts=4

[in1]
Delay=0
Capacitance=43

[in2]
Delay=0
Capacitance=41

[in3]
Delay=0
Capacitance=41

[out1]
Delay=50
Capacitance=0
```

REFERENCES

1. Najm, F. N., R. Burch, P. Yang, and T. N. Trick, "A Monte Carlo Approach for Power Estimation," *in IEEE Trans. on VLSI Systems*, pp. 63-71, March 1993.

2. Ding, C. S., C. T. Hsieh, Q. Wu, and M. Pedram, "Stratified Random Sampling for Power Estimation," *in IEEE/ACM Int. Conf. on Computer Aided Design*, pp. 576-582, 1996.

3. Bryant, R., "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 677-691, Aug. 1986.

4. Ghosh, A., S. Devadas, K. Keutzer, and J. White, "Estimation of average switching activity in combinational and sequential circuits," *in Proc. Design Automation Conf.*, pp. 253-259, June 1992.

5. Najm, F. N., R. Burch, P. Yang, and I. N. Hajj, "Probabilistic simulation for reliability analysis of CMOS circuits," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 439-450, Apr. 1990.

6. Najm, F. N., "Transition density, a stochastic measure of activity in digital circuits," *in IEEE Trans. on Computer Aided Design*, vol. 12, pp. 310-323, Feb. 1993.

7. Ding, C. S., C. Y. Tsiu, and M. Pedram, "Gate-level power estimation using tagged probabilistic simulation," *IEEE Trans. Computer Aided Design*, vol. 17, pp. 1099-1107, Nov. 1998.

8. Marculescu, R., D. Marculescu, and M. Pedram, "Switching Activity Analysis Considering Spatiotemporal Correlations," *in Proc. IEEE Int. Conf. on Computer Aided Design*, Nov. 1994.

9. Tsiu, C.Y., J. Monteiro, M. Pedram, S. Devadas, and A. M. Despain, "Power Estimation Methods for Sequential Logic Circuits," *IEEE Trans. VLSI Systems*, pp. 404-416, Sep. 1995.

10. Nemani, M., and F. N. Najm, "High-Level Area and Power Estimation for VLSI Cicuits," *in IEEE Trans. on Computer Aided Design*, vol. 18, pp. 697-713, June 1999.

11. Singh, D., J. M. Rabaey, M. Pedram, F. Catthoor, s. Rajgopal, N. Sehgal, and T. J. Mozdzen, "Power Conscious CAD Tools and Methodologies: A Perspective," *in Proc. Of the IEEE*, vol. 83, pp. 570-594, April 1995.

12. Iman, S., and M. Pedram, "POSE: Power Optimization and Synthesis Environment ," *in Proc. IEEE/ACM 33rd Design Automation Conference*, pp. 21-26, June 1996.

13. Dikel, A., "An Optimized Method for the Estimation of Power Dissipation in Adders as CMOS Arithmetic Building Blocks," MS Thesis, Bogazici University, 1999.

14. Karakus, G., "A Fast and Accurate Delay Estimation Method for Adders as CMOS Arithmetic Building Blocks," MS Thesis, Bogazici University, 1999.

REFERENCES NOT CITED

Macii, E., M. Pedram, and F. Somenzi, "High-level power modeling, estimation, and optimization," *IEEE Trans. Computer-Aided Design*, vol 17, pp. 1061-1079, Nov. 1998.

Rabaey, J.M., Digital Integrated Circuits, Prentice Hall, New Jersey, 1996.

Tiwari, V., P. Ashar, and S. Malik, "Technology Mapping for Low Power," *in IEEE/ACM 30th Design Automation Conference*, pp. 74-79, June 1993.