# Static power analysis for power-driven synthesis

S.-Y.Yuan
K.-H.Chen
J.-Y.Jou
S.-Y.Kuo

*Indexing terms: Probabilistic method, Simulation-based method, Symbolic simulator*

**Abstract:** A new static power analysis method for CMOS combinational circuits is presented. This approach integrates the simulation-based method and the probabilistic method, and can establish the relationships between the primary inputs and the internal nodes in the circuit. Based on the relationships, our approach can also indicate which internal node or input sequence consumes the most power. It is thus suitable for performing power estimation in the synthesis environment for power optimisation. To the best of our knowledge, this is the first attempt to develop a systematic way to symbolically represent the relationships between the primary inputs and the power consumption at every internal node of a circuit. Furthermore, by using the existing piecewise linear delay model, as well as the proposed algorithm, this novel method is also very accurate and efficient. For a set of benchmark circuits, the experimental results show that the power estimated by our technique is within 5% error as compared with that by the exact SPICE simulation, while the execution speed is more than four orders of magnitude faster.

## 1 Introduction

Power dissipation has emerged as an important design parameter in the design of microelectronic circuits, especially in portable computing and personal communication applications. More generally, as the density and the size of chips and systems continue to increase, the problem of power consumption becomes a critical concern in VLSI design [1]. Low power design techniques are becoming increasingly important in today's integrated circuit designs [2]. Therefore, a fast and accurate power estimator is necessary for a low power circuit/system designer.

During the synthesis for low power at higher level of abstractions such as the register transfer level, it is almost impossible to have a very accurate estimation about the power consumption of each module. However, if the estimation model is able to correctly sort the alternative designs according to the estimated power, we can still have the right designs for low power because there are relatively few alternative designs and each design has very different trade-off among power, area and performance. However, performing synthesis for low power at gate level or below, the power estimation with high accuracy becomes a must, because there could be many alternative designs and it is difficult to have estimation with relatively low accuracy but with good fidelity [3]. Therefore, with a less accurate estimation of power, we may optimise the circuits in the wrong spots such that we cannot lower the power consumption with minimum overheads in terms of area and performance.

There are quite a few approaches proposed [4–6] to reduce the inefficiency of the SPICE while maintaining acceptable estimation errors. The PowerMill approach [4] is a transistor-level power simulator, which uses an event-driven simulation algorithm to increase the speed by two to three orders of magnitude over SPICE. Although the PowerMill is relatively accurate, it is still not suitable for power-driven synthesis. This is because, when power optimisation is performed, the circuits will be modified frequently. Power estimation should be done incrementally to speed up the process. The simulation-based approach cannot be used in this situation. Switch-level simulation techniques are, in general, much faster than circuit-level simulation techniques, but are not as accurate or versatile. Standard simulators, such as IRSIM [6], can be easily modified to report the switched capacitance (thus the dynamic power dissipation) during a simulation run.

However, the approaches mentioned above suffer three major problems as power simulation tools. First, they must simulate the 'chosen patterns' with many iterations to determine the average power consumption of each node, which slows down the simulation speed. Also, the average results strongly depend on the 'chosen patterns', and we may get biased simulation results. Secondly, if the *PI*s are not fully independent, the choices of patterns need much more attention and the number of patterns needed is large. Thirdly, due to the nature of simulation-based simulators, they cannot provide enough power information such as the percentage of glitch power of each node's power consumption, the difference between generation of glitches and passing of glitches, the source node of a glitch, or the cause of the glitch. Thus, these approaches can leave the user or synthesiser in a vague situation to improve or resynthesise a circuit.



*IEE Proceedings* online no. 19981909

Paper received 5th January 1998

S.-Y.Yuan, K.-H.Chen and S.-Y.Kuo are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

J.-Y.Jou is with the Department of Electronic Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China

In this paper, we will develop a new method which retains the advantages of both the simulation-based methods and the probabilistic methods. This power analyser can estimate the total power consumption due to the circuit itself as well as the power consumption due to the transition, and the spike at every internal node in the circuit efficiently and accurately. Furthermore, this new approach can not only establish the relationships between the primary inputs and the internal nodes in the circuit but also increase the efficiency of the simulation-based method. Ghosh proposed a symbolic simulator based on the binary decision diagram (BDD) [7]. The major difference between our method and the method in [7] is that we use the idea of cube representation to simplify the process of simulation. By using the idea of a cube, we can get much more information from the simulation than the method in [7]. To the best of our knowledge, our approach is the first attempt to develop a systematic way to symbolically represent the relationships between the primary inputs and the power consumption at every internal node of a circuit. The simulation results show that our new method provides much more information on power consumption than other methods. The approach can thus be integrated into a synthesis environment to determine where it can be improved or resynthesised for low power. Thus, this method is very suitable for performing power estimation in the synthesis environment for power optimisation.

## 2 Power dissipation model and definitions

### 2.1 Power dissipation model

It is well known that the dynamic power estimation formula is $P = 1/2\alpha CV^2f$ where $P$ is the average power, $\alpha$ is the switching activity, $V$ is the supply voltage, $f$ is the frequency, and $C$ is the load capacitance of the gate [8, 9]. We use an ideal gate (e.g. AND, OR, etc.) and equivalent input and output capacitances to model a real gate. Using [1], we can estimate $C$.

$$P = \frac{1}{2}\alpha CV^2 f$$
$$= \frac{1}{2} \times \frac{actual\_node\_transition\_number}{max\_node\_transition\_number} \times CV^2 \times \frac{1}{T_{cycle}}$$
$$\Rightarrow C = \frac{2 \times P \times simulation\_time}{V^2 \times actual\_node\_transition\_number} \quad (1)$$

We also established the database of mutiple-SPICE-simulation-based thresholds [10] to approach the piecewise linear delay model, which will further improve the precision and efficiency of the simulation. The simulation result was very accurate and efficient in [10]. The error percentage is less than 5% as compared with the HSPICE simulation, while the execution speed is more than three orders of magnitude faster. For some circuits, the speedups are even more than four orders of magnitude larger. However, a large number of different input sequences is required for [10], and this simulation-based method was still very time consuming. Furthermore, the information on the average power consumption is not enough for performing power optimisation in the synthesis environment. In other words, with only these average power values derived from the simulation-based method, we cannot efficiently figure out where the most power is consumed and why. Therefore, it is necessary to combine a simulation-based method with a probabilistic method. In the following discussions, we will develop a new method based on the static analysis approach. This method cannot only construct the relationships between the primary inputs and the internal nodes in the circuit but also increase the efficiency of the simulation-based method.

### 2.2 Node probabilities

The signal probability $p(X)$ of a node $X$ is defined as the probability that node $X$ has a value of logic 1. Let us now define three special probabilities $P_1$, $P_0$, and $P_S$. Assume that node $X$ is the output of a gate $g$. Thus, the switching probability of $X$, $P_S(X)$, is equal to $2 \times p(X) \times (1 - p(X))$ and is defined as the probability that node $X$ will switch from low to high or high to low if any input(s) of gate $g$ changes. The holding-one probability of $X$, $P_1(X)$, is equal to $p(X)^2$ and is defined as the probability that node $X$ will hold in high (one) if any input(s) of gate $g$ changes. The holding-zero probability of $X$, $P_0(X)$, is equal to $(1 - p(X))^2$ and is defined as the probability that node $X$ will hold in low (zero) if any input(s) of gate $g$ changes.

We define a cube with $n$ elements as: $<PK(PI_1)\ (PI_1), PK(PI_2)\ (PI_2), \ldots, PK(PI_n)\ (PI_n)>$, given a circuit with $n$ primary inputs ($PI_j$, $j = 1, \ldots, n$), $m$ primary outputs ($PO_k$, $k = 1, \ldots, m$), and many internal nodes. The probability of this cube is: $\Pi_{i=1}^{n}\ PK(PI_i)\ (PI_i)$, where $K(PI_i)$ can be 1, 0, or $S$. Here we assume that the primary inputs are uncorrelated for the sake of easy explanation.

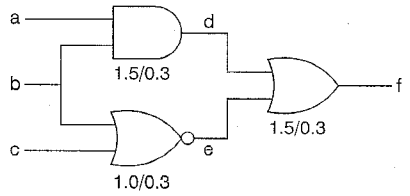


**Fig. 1** *Example circuit*

For example, in Fig. 1, $a$, $b$, $c$ are $PI$s, $f$ is $PO$, and $d$, $e$ are internal nodes.

Assume the AND/OR gate has a delay time of 1.5 units, the NOR gate has a delay time of 1.0 unit, and the inertial of each gate is 0.3 units. Given the signal probabilities of the primary inputs $a$, $b$, and $c$, we can determine $P_1(a)$, $P_0(a)$, $P_S(a)$, ..., $P_S(c)$. By applying the symbolic simulation approach, we can further calculate the $P_1(a)$, $P_0(d)$, $P_S(d)$, ..., $P_S(f)$. For example, let $C_1(d) = <P_1(a), P_1(b), 1>$ for internal node $d$, the cube $C_1(d)$ means that to have internal node $d$ in the holding-one state, the primary input node $a$ and node $b$ must be both in the holding-one state and the node $c$ can be in any of the three states. The probability $P_1(d)$ is equal to $P_1(a) \times P_1(b) \times 1$.

### 2.3 Definitions of symbols and cubes

Since each element of the cube represents the corresponding input, we do not have to write the $X$ explicitly. Therefore, we introduce simple notations to be used in the cube as follows:

(i) **1**: the probability of any particular $PI$ in the holding-one state.

(ii) **0**: the probability of any particular $PI$ in the holding-zero state.

(iii) **s or b**: the probability of any particular $PI$ in the switching state. All the $PI$s with the same switching direction (i.e. low to high (high to low)), are represented as **s** (**b**), while all other $PI$s with opposite switching direction are represented as **b** (**s**).

(iv) **S(i) or B(i)**: the probability of any particular *PI* in the switching state (*i* is just a sequence index). However, the effect of this switching is blocked by some gates between the *PI*s and the node under consideration, and hence no transition is generated by the switching of *PI*s at this node. Thus, this switching state is less restrictive than **s** or **b**. We call this a don't-care switching state.

(v) –: this input is not related to the cube. We call this a don't-care state. The probability is one.

For the determination of the effect of a spike, we extend the cube by adding two fields, the beginning time and the lasting time, to describe the timing information. The beginning time represents the starting time of the action and the lasting time represents how long such an action will last. We will simply call the holding-one cube as the 1-cube (represented as $C_1(X)$), the holding-zero cube as the 0-cube (represented as $C_0(X)$), the switching cube as the *S*-cube (represented as $C_S(X)$), and the spike cube as the *G*-cube (represented as $C_G(X)$). The 1-cube and the 0-cube specify the conditions and probabilities for any particular node to hold in the 1 state and the 0 state respectively. Thus, their beginning times are always set to 0.0 and their lasting times are always set to $\infty$. For the *S*-cube, the beginning time is set to the time when the switching starts and the lasting time is set to $\infty$. The *G*-cube's beginning time is the beginning time of the first calculated switching, and its lasting time is the spike duration time. For example, assume that there is a node *y* whose $C_S(y)$ = <1.35, $\infty$, *s*, *b*, 1, *S*(1), -, *B*(1), *S*(2)>. The circuit has seven *PI*s ($PI_1$ ... $PI_7$). The cube $C_S(y)$ means that one of the cases to make node *y* switch is to let $PI_1$ in the switching state, $PI_2$ in the switching state with a switching direction opposite to the switching direction of $PI_1$, $PI_3$ in the holding-one state, $PI_4$ in the don't-care switching state, $PI_5$ in the don't-care state, $PI_6$ in the don't-care switching state with opposite switching direction to $PI_4$, and $PI_7$ also in the don't-care switching state not related with any other *PI*s. The switching will start at time = 1.35 units.

The symbols 0, 1, and *s* are similar to $P_j^{00}$, $P_j^{01}$, $P_j^{10}$, and $P_j^{11}$ proposed in [7]. However, Ghosh in [7] used the BDD representation to simulate the circuit. Instead of BDD, we use the cube-based operation in our simulator. It cannot only easily simulate the circuit but can also provide us with much more information for synthesising low-power circuits. This information includes which internal node or which input sequence consumes the most power. Therefore, our new proposed method is more suitable to be used in the synthesis environment.

## 2.4 Definition of cube sets

We call the union of the same type of cubes a 'cube set'. Every node in a circuit has four cube sets (i.e. 1, 0, *S*, and *G* cube sets). For a node *X*, the four cube sets are represented as $\{C_0(X)\}$, $\{C_1(X)\}$, $\{C_S(X)\}$, $\{C_G(X)\}$. For example, $\{C_S(d)\}$ = {<1.5, $\infty$, 1, *s*, ->, <1.5, $\infty$, *s*, 1, ->, <1.5, $\infty$, *s*, *s*, ->}, and $\{C_G(f)\}$ = {<2.5, 0.5, *s*, *s*, *s* >, ... }. The $\{C_S(d)\}$ has three *S*-cubes in it. Each represents a different *PI* combination to make node *d* switch. The example *G*-cube in the $\{C_G(f)\}$ means that the node *f* will have a spike after a delay of 2.5 time units and the spike_duration is 0.5 time unit, if all the *PI*s switch in the same direction.

## 3 Operators and algorithm

We will, in this Section, define operations at the cube level, the cube set level, and the logic node level, respectively. All higher level operations are built upon the lower level operations.

## 3.1 Cube level operators

The proposed operators for cubes are intersection ($\cap$, e.g. $A \cap B$), *bar*(-, e.g. $\bar{A}$), and don't-care (*dc*, e.g. *dc*(*A*)). Given an OR gate *g* with a gate delay of 1.3 time units and inertial delay of 0.3 time unit, its fanout is node *z* and its fanins are node *x* and node *y*. The nodes *x*, *y*, and *z* are all internal nodes. Assume a 1-cube in $\{C_1(X)\}$ is <0.0, $\infty$, 1, -, *S*(1), *B*(1), *s*> and an *S*-cube in $\{C_S(Y)\}$ is <1.4, $\infty$, 1, *S*(2), *S*(2), *s*, ->. The main function of the $\cap$ operator is to find the *PI* state which is compatible with the *PI*-part of both cubes. The procedure is explained as follows and is illustrated in Fig. 2.

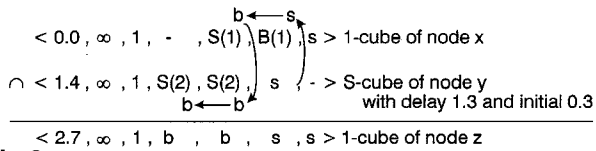


**Fig.2** *Intersection example*

(*i*) $PI_5$: $s \cap - \Rightarrow s$.

(ii) $PI_4$: $B(1) \cap s \Rightarrow s$. Since *s* is more restricted than *B*(1), the result is *s*.

(iii) $PI_3$: $S(1) \cap S(2) = b \cap S(2) \Rightarrow b$. Since *B*(1) in $PI_4$ position of the first cube is changed to *s*, the *S*(1) in $PI_3$ position of the first cube has to be changed to *b*, which is opposite to *s*, and *S*(2) in $PI_3$ position of the second cube is also changed to *b*.

(iv) $PI_2$: $- \cap S(2) = - \cap b \Rightarrow b$, since the *S*(2) in $PI_3$ position of the second cube has been changed to *b*.

(v) $PI_1$: $1 \cap 1 \Rightarrow 1$.

We can see that the $\cap$ operation is not a straightforward element-wise AND operation on cubes. Because the result of intersection is in the 1-cube set of node *z*, the cube cannot contain *s* (or *b*) and the beginning time must be reset to 0.0. So we apply the *dc* operator on the intersection result, such that *dc*(<2.7, $\infty$, 1, *b*, *b*, *s*, *s*<) $\Rightarrow$ <0.0, 00, 1, *B*(1), *B*(1), *S*(1), *S*(1)>. Another operator is the bar ($\overline{cube}$). It inverts all switching/don't-care switching elements in a cube only. For example, given the delay time *d* of an inverter, <0.5, $\infty$, 1, *s*, –, *B*(2), *b*, *S*(1)> $\Rightarrow$ <0.5 + *d*, $\infty$, 1, *b*, –*S*(2), *s*, *B*(1)>.

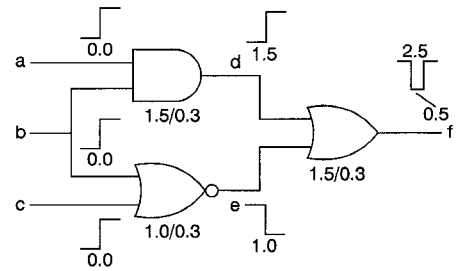


**Fig.3** *Example spike (glitch generated)*

## 3.2 Spike determination

A spike is generated by two different signals going through the same gate with different arrival times and opposite transition directions. In Fig. 3, we show a

spike generated at node $f$. Here we assume the parameters of Fig. 3 are the same as Fig. 1.

The $G$-cube is determined as:

(i) beginning time = the smaller of the beginning times of the two cubes + the delay of this gate,

(ii) lasting time = absolute value of the difference of the two cubes' beginning times,

(iii) $PI$-part: the intersection of the two cubes' $PI$-part with one cube inverted.

For example, the $C_G(f)$ can be obtained by: $C_S(d) \cap \overline{C_S(e)} \Rightarrow <1.0, \infty, s, s\text{->} \cap \overline{<1.0, \infty, -, b, b>} \Rightarrow <1.5, \infty, s, s, \text{->} \cap <1.0, \infty, -, s, s> \Rightarrow <1.0 + 1.5, |1.5 - 1.0|, s, s, s>$. When all the primary inputs $a$, $b$, $c$ make transitions in the same direction, the spike will be generated at node $f$ at time = 2.5 time units after the $PI$ transitions (add a delay of 1.5 time units since $|1.5 - 1.0| > 0.3$), and this spike will last for a time = 0.5 time unit.

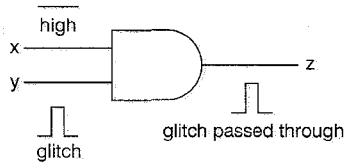


**Fig. 4** *Another spike example (glitch pass through)*

Fig. 4 shows another example which will pass a spike. Our method can determine this spike by intersecting $C_1(x)$ with $C_G(y)$.

## 3.3 Formal definitions of cube level operators

After describing the basic ideas of the key operations at the cube level, let us formally define all the operators needed in our approach. Assume $d$ is a gate's delay and $i$ is a gate's inertial.

*3.3.1 $bar_{(d,i)}$:* Compare the lasting time of a cube with $i$. If the lasting time is less than $i$, the result of this operator is an empty cube. Otherwise, add a delay of $d$ units to the beginning time of the cube, keep the original lasting time in the cube, and change the $PI$-part based on the rules as shown in Table 1.

**Table 1: Operation rules of bar**

| Bar | Change to |
|---|---|
| 1 | 1 |
| 0 | 0 |
| s | b |
| b | s |
| $S(i)$ | $B(i)$ |
| $B(i)$ | $S(i)$ |

*3.3.2 dc:* Reset the beginning time to 0.0, and the lasting time to $\infty$. Scan the $PI$-part to find the smallest $i \in N$ such that neither $S(i)$ nor $B(i)$ is in the $PI$- part. Change the symbol $s$ to $S(i)$ and the symbol $b$ to $B(i)$.

*3.3.3 $\cap(d)$:* Intersect two cubes and obtain a new cube. In the new cube, the beginning time = $d$ + Max(beginning times of the two cubes), the lasting time = $\infty$, and the $PI$-part is the result of applying $\cap$ operator on the $PI$-parts of the two cubes.

*3.3.4 $\cap(d, i, 1)$:* If both lasting times are less than $i$, the result is an empty cube; otherwise, the way to derive the new cube is:

timing fields: (beginning time = d + the larger beginning time of the two cubes; lasting time = the smaller lasting time of the two cubes;)

$PI$-part: Intersection of the $PI$-parts of the two cubes.

*3.3.5 $\cap(d, i, 2)$:* If the difference between the two cubes' beginning times is less than $i$, the result is an empty cube; otherwise, the way to derive the new cube is:

timing fields: (beginning time = d + the smaller beginning time of the two cubes; lasting time = difference of the two cubes' beginning times;)

$PI$-part: Intersection of the $PI$-parts of the two cubes.

## 3.4 Formal definitions of cube-set level operators

Let $Cu$, $Cv$ and $Cw$ be three cube sets. Union is the ordinary union operator on sets. For the cube set (1, 0, $S$, $G$), there are six operators.

(i) – $(d, i)$: $Cu = \overline{Cv}^{(d,i)} \equiv$ union of $(\forall v \in Cv, bar_{(d,i)}(v))$

(ii) $DC$: $Cu = DC(Cw) \equiv$ union of $(\forall w \in Cw, dc(w))$

(iii) $\times_{(d)}$: $Cu = Cv \times (d)\ Cw \equiv$ union of $\{\forall v \in Cv, \forall w \in Cw, v \cap_{(d)} w\}$

(iv) $\times_{(d,i,1)}$: $Cu = Cv \times_{(d,i,1)} Cw \equiv$ union of $\{\forall v \in Cv, \forall w \in Cw, v \cap_{(d,i,1)} w\}$

(v) $\times_{(d,i,2)}$: $Cu = Cv \times_{(d,i,2)} Cw \equiv$ union of $\{\forall v \in Cv, \forall w \in Cw, v \cap_{(d,i,2)} w\}$

(vi) +: $Cu = Cw + Cv \equiv$ union of $\{Cv$ and $Cw\}$

## 3.5 Operators at the node level

We will develop the operators only for the two input AND, OR, and NOT gates. The operators for all other complex gates can be built based on this foundation.

*3.5.1 operator NOT:* Given an inverter with input $a$ and output $c$ and with delay $d$ and inertial $i$, the corresponding cube set of output $c$ is calculated as follows:

(i) $\{C_1(c)\} = \{C_0(a)\}$, $\{C_0(c)\} = \{C_1(a)\}$

(ii) $\{C_S(c)\} = \overline{\{C_S(a)\}}^{(d,i)}$ $\{C_G(c)\} = \overline{\{C_G(a)\}}^{(d,i)}$

*3.5.2 operator AND:* Given an AND gate with inputs $a$ and $b$ and output $c$ and with delay $d$ and inertial $i$, the corresponding cube sets of output $c$ is calculated as follows:

(i) $\{C_1(c)\} = DC(\{C_1(a)\} \times_{(d)} \{C_1(b)\})$

(ii) $\{C_0(c)\} = DC(\{C_0(a)\} + (\{C_1(a)\} + \{C_S(a)\}) \times_{(d)} \{C_0(b)\} + \{C_S(a)\} \times_{(d)} \overline{\{C_S(b)\}}^{(d,0)})$

(iii) $\{C_S(c)\} = (\{C_1(a)\} \times_{(d)} \{C_S(b)\}) + (\{C_S(a)\} \times_{(d)} \{C_1(b)\}) + (\{C_S(a)\} \times_{(d)} \{C_S(b)\})$

(iv) $\{C_G(c)\} = (\{C_1(a)\} \times_{(d,i,1)} \{C_G(b)\}) + (\{C_G(a)\} \times_{(d,i,1)} \{C_1(b)\}) + (\{C_S(a)\} \times_{(d,i,2)} \overline{\{C_S(b)\}}^{(d,0)})$

*3.5.3 operator OR:* Given an OR gate with inputs $a$ and $b$ and output $c$ and with delay $d$ and inertial $i$, the corresponding cube sets of output $c$ is calculated as follows:

(i) $\{C_0(c)\} = DC(\{C_0(a)\} \times_{(d)} \{C_0(b)\}$

(ii) $\{C_1(c)\} = DC(\{C_1(a)\} + (\{C_0(a)\} + \{C_S(a)\}) \times_{(d)} \{C_1(b)\} + \{C_S(a)\} \times_{(d)} \{C_S(b)\})$

(iii) $\{C_S(c)\} = (\{C_0(a)\} \times_{(d)} \{C_S(b)\}) + (\{C_S(a)\} \times_{(d)} \{C_0(b)\}) + (\{C_S(a)\} \times_{(d)} \{C_S(b)\})$

(iv) $\{C_G(c)\} = (\{C_0(a)\} \times_{(d,i,1)} \{C_G(b)\}) + (\{C_G(a)\} \times_{(d,i,1)} \{C_0(b)\}) + (\{C_S(a)\} \times_{(d,i,2)} \{C_S(b)\})$

### 3.6 Partitioning and cube reduction algorithm

To store the cube and cubeset completely, the memory may not be enough. If we can reduce the number of *PI*s, we can reduce the size of individual cube and cubeset. Also, if we can reduce the number of cubes, we can surely reduce the memory requirement. We solve the problem with two steps: partitioning the circuit into groups, then reducing the number of cubes dynamically.

*3.6.1 Partitioning into groups:* We first determine the distances between different *PO*s, then divide *PO*s into several groups by using the distance information. Then, the grouped *PO*s, the *PI*s and the internal nodes that have fanouts to these *PO*s are put into the same partition. Note that we have not lost any information at this step. We just separate the unrelated *PI*s and internal nodes into different groups to reduce the unnecessary calculations.

The distance is defined as:

$$d(x,y) = 1, \quad \text{if } \exists \text{ gate } g, \text{ such that } x \text{ (or } y\text{) is } g\text{'s fanin, and the other is } g\text{'s fanout.}$$

$$d(x,y) = \min_{z \in \text{internal mode}} \{d(x,z) + d(z,y)\}$$

*3.6.2 Cube reduction algorithm:* When a predefined memory usage limitation is reached, the dynamic cube reduction algorithm is executed to average the $C_1$ and $C_0$ cubeset of some frontier nodes. Make them as new *PI*s and have a new simulation starting from these nodes. Since the most important cubesets are $C_S$ and $C_G$, we focus on how to reduce these cube sets.

### 3.7 Algorithm

Given a netlist, we first sort the gates in the netlist topologically. We then apply the corresponding operators defined previously to each gate based on the topological order. The complexity of traverse is proportional to the number of gates. The overall algorithm is shown below.

```
Static-power-analysis
{
  Read netlist;
  Read technology file;
  Sort nodes topologically;
  Partition the netlist by predefined distance;
  for (each partition) do
  {
    for (each node in this partition) do
    {
    if (memory limit is reached) do
    cube-reduction;
    Apply node level operations to this node;
    }
  }
  Report results;
}
```

## 4 Experimental results

The proposed analyser based on the above power estimation model and delay model has been implemented in C on a SUN SPARCstation 10 with 7000 lines of C codes. The transistor models used are the models of a TSMC 0.8μm SPDM CMOS technology. Table 2 shows the full timing data of the basic gates derived from the SPICE simulation by applying the method proposed in [10]. We run the SPICE and the proposed analyser on the circuit shown in Fig. 5 to evaluate the quality and efficiency of the method. Table 3 shows the detail simulation results the power consumption.

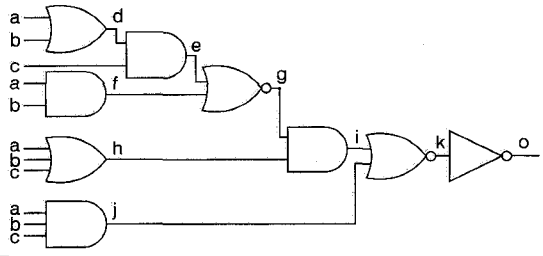


**Fig. 5** *Example circuit*

**Table 2: Full specification of timing data**

| | Fanin | Equ out c[a] | Equ inp c[b] | Delay (ns) | Threshold (ns) |
|---|---|---|---|---|---|
| inv | 1 | 0.5 | 0.4 | 0.065 | 0.0 |
| and | 2 | 1.36 | 0.26 | 0.195 | 0.065 |
| and | 3 | 1.29 | 0.26 | 0.195 | 0.065 |
| nand | 2 | 0.37 | 0.25 | 0.13 | 0.065 |
| nand | 3 | 0.41 | 0.26 | 0.13 | 0.065 |
| or | 2 | 1.45 | 0.37 | 0.195 | 0.065 |
| or | 3 | 1.53 | 0.37 | 0.195 | 0.065 |
| nor | 2 | 0.54 | 0.37 | 0.13 | 0.065 |
| nor | 3 | 0.62 | 0.36 | 0.13 | 0.065 |

[a] stands for equivalent output capacitance, unit: 1e-13f
[b] stands for equivalent input capacitance, unit: 1e-13f

**Table 3: Simulation results on an example circuit**

| Node | SPICE (mW) | Proposed method (mW) | | | Error (%) |
|---|---|---|---|---|---|
| | | real | spike | total | |
| *d* | 0.157 | 0.174 | 0 | 0.174 | 10.83 |
| *e* | 0.238 | 0.218 | 0.044 | 0.262 | 10.08 |
| *f* | 0.156 | 0.174 | 0 | 0.174 | 11.54 |
| *g* | 0.192 | 0.168 | 0.021 | 0.189 | −1.56 |
| *h* | 0.118 | 0.106 | 0 | 0.106 | −10.17 |
| *i* | 0.282 | 0.218 | 0.087 | 0.305 | 8.21 |
| *j* | 0.105 | 0.098 | 0 | 0.098 | −6.67 |
| *k* | 0.176 | 0.118 | 0.059 | 0.177 | 0.57 |
| *o* | 0.070 | 0.045 | 0.023 | 0.068 | −2.86 |
| total | 1.494 | 1.319 | 0.234 | 1.553 | 3.95 |

From the cube sets derived above, we can easily determine the number of real transitions $E_S(sw)$ and the number of spikes $E_G(sw)$. Therefore, $E_{tot}(sw) = E_S(sw) + E_G(sw)$. The simulation results of all the nodes by the SPICE and the proposed method are shown in Table 3. The estimation on the total power consumption by the proposed method has an error of about 4% compared with that by the SPICE simulation.

**Table 4: Simulation results**

| Circuit | Events | Inputs | Outputs | Number of gates | SPICE | | Proposed method | | Error (%) | Speedup |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Power (mW) | Time (s) | Power (mW) | Time (s) | | |
| decoder | 64 | 3 | 4 | 6 | 0.5344 | 106 | 0.520 | 0.055 | -2.62 | 1927 |
| 1b-adder | 64 | 3 | 2 | 10 | 1.569 | 141.5 | 1.609 | 0.109 | 2.54 | 1298 |
| 1b-mult | 256 | 4 | 4 | 14 | 1.543 | 884 | 1.510 | 0.549 | -2.14 | 1610 |
| adder2 | 64 | 3 | 2 | 16 | 1.457 | 161 | 1.483 | 0.136 | 1.78 | 1183 |
| decoder2 | 256 | 4 | 8 | 19 | 1.765 | 1110 | 1.821 | 0.718 | 3.17 | 1546 |
| c17 | 1024 | 5 | 3 | 6 | 0.4731 | 1270 | 0.458 | 0.165 | 3.17 | 7697 |
| cs27 | 1000 | 7 | 4 | 10 | 1.429 | 2197 | 1.71 | 0.275 | 0.96 | 7989 |
| cs208 | 100 | 19 | 10 | 102 | 6.52 | 6320 | 6.60 | 7.30 | 1.23 | 866 |
| cs298 | 100 | 17 | 20 | 164 | 25.77 | 10161 | 24.67 | 8.77 | -4.27 | 1159 |
| c880 | 100 | 60 | 26 | 447 | 71.65 | 44600 | 74.32 | 41.41 | 3.73 | 1077 |
| c432 | – | 36 | 7 | 267 | – | – | 73.14 | 28.72 | – | – |
| cs1196 | – | 31 | 31 | 623 | – | – | 76.33 | 85.38 | – | – |
| c1355 | – | 41 | 32 | 682 | – | – | 126.45 | 58.36 | – | – |
| c1908 | – | 33 | 25 | 1049 | – | – | 213.62 | 241.0 | – | – |
| c2670 | – | 157 | 64 | 1385 | – | – | 378.12 | 258.0 | – | – |

Table 4 shows the simulation results on several circuits when exhaustive simulation is performed using both the SPICE and our analyser. These test benchmarks include decoders, full adders, and a multiplexer. The 'cs*' benchmarks (cs27 .. cs1196) are the ISCAS-89 sequential circuits in which the feedback loops and FFs are taken out. The 'c*' benchmarks (c17 .. c2670) come from the ISCAS-85 benchmark circuits. All the primary inputs are assumed to be temporally and spatially independent and with a signal probability of 0.5 in the experiments. Our analyser can be four orders faster than the SPICE with less than 5% error. Better accuracy could be further achieved by fine tuning the *G*-cube calculation algorithm. From the cs208 to c2670, we use 50 groups of 100 random patterns for SPICE simulation and then average the power of the different 50 groups to compare with our simulation. Due to the partitioning, the results of our simulator for these benchmarks are three orders faster than the SPICE. However, we must remember that the result of SPICE is from the average of 50 groups of 100 random patterns. It means that the proposed method is more efficient than the SPICE method. From the c432 to c2670, it takes an enormous time to get the results of SPICE simulation results. The simulation time of a single input pattern is more than three days. Therefore, the results by the SPICE simulation are ignored.

Since our analyser can recalculate the results by using the existing cube sets, whenever the transition density of any node is changed, our algorithm has the incremental capability which the SPICE does not have. Therefore, our estimator is particularly useful in the synthesis environment for power optimisation.

## 5 Conclusions and future work

We have proposed a novel static power analyser for CMOS combinational circuits. The analyser can estimate the power consumption of a circuit very fast (4 orders faster than SPICE) and very accurately (with a 5% error compared with SPICE). The analyser is also applicable for different delay models with or without inertial delays. Furthermore, it can distinguish functional transitions from spikes and has incremental capability. Last but not least, the analyser can identify the input transitions which cause the large power consumption so that power optimisation can be appropriately applied to improve the circuit. Therefore, this analyser is very useful in the synthesis environment for low power.

One of the main future pieces of work is to solve the spatial and temporal dependency of the primary inputs. By giving different weights to different cubes of the *S*-cube set and the *G*-cube set, which means the patterns of some cubes may appear more frequently in the input vector than others, we can calculate the temporal dependency approximately. Using a table lookup method instead of the simple multiplication to calculate the $E(sw)$, which means different primary inputs are not independent of each other, we can approximate the spatial dependency among different primary inputs. Another direction of future works is to expand this approach to cover other abstraction levels such as circuit level and functional level by extending the symbols and the definitions of cube and cube operations.

## 6 Acknowledgment

This research was supported by the National Science Council, Taiwan, ROC under Grant NSC 84-2213-E005-035.

## 7 References

1 NAJM, F.N.: 'A survey of power estimation techniques in VLSI circuits', *IEEE Trans.*, 1994, **VLSI-2**, (4), pp. 446–455
2 LANDMAN, P.E.: 'Low-power architectural methodologies'. Technical report, Dept. of EECS, University of California, Berkeley, 1994
3 GAJSKI, D., DUTT, N., WU, A., and LIN, S.: 'High-level synthesis introduction to chip and system design' (Kluwer, 1992)
4 CHANDRAKASAN, A.P., and BRODERSEN, R.W.: 'Low power digital CMOS design' (Kluwer Academic Publishing Co., 1995)
5 DENG, C.: 'Power analysis for CMOS/BiCMOS circuits'. Proceedings of 1994 international workshop on *Low power design*, 1994, pp. 308–318
6 SALZ, A., and HOROWITZ, M.A.: 'IRSIM: An incremental MOS switch-level simulator'. Proceedings of the 26th *Design automation* conference, 1989, pp. 173–178

7 GHOSH, A., DEVADAS, S., KEUTZER, K., and WHITE, J.: 'Estimation of average switching activity in combinational and sequential circuits'. 29th ACM/IEEE *Design automation* conference, 1992, pp. 253–259
8 WESTE, N., and ESHRAGHIAN, K.: 'Principles of CMOS VLSI design' (Addison-Wesley Publishing, Reading, MA, 1993)
9 DRESIG, F., LANCHES, PH., RETTIG, O., and BAITINGER, U.G.: 'Simulation and reduction of CMOS power dissipation at logic level'. Proceedings of 1994 European conference on *Design automation*, 1994, pp. 341–346
10 CHEN, K.-H., YUAN, S.-Y., JOU, J.-Y., and KUO, S.-Y.: 'Cell-based power estimation for CMOS combinational circuits using a logic simulator'. The 7th VLSI *Design/CAD* symposium proceedings, 1996, pp. 81–84