Accurate Estimation of Combinational Circuit Activity

Huzefa Mehta, Manjit Borah, Robert Michael Owens, Mary Jane Irwin
Department of Computer Science and Engineering,
The Pennsylvania State University
University Park, PA 16802

Abstract

Several techniques to estimate power consumption of a combinational circuit using probabilistic methods have been proposed. However none of these techniques take into account circuit activity when two or more inputs change simultaneously or when glitching occurs. A formulation is presented in this paper which includes signal correlation and multiple gate input switching. Work is also presented in estimating the glitching contribution to the switching activity. Results obtained from benchmarks and test circuits show very good accuracy when compared to actual activities as measured by SPICE and IRSIM.

1 Introduction

With the emergence of battery-operated applications that demand intensive computation in a portable environment, power analysis and optimization have become important functions in order to reduce power dissipation in circuits. Accurate estimation of power dissipation is a significant task needed to assist design and synthesis tools in their attempt towards designing low power circuits. Power dissipation in a CMOS circuit is directly related to the extent of switching activity of the nodes in the circuit. A direct and simple approach to estimate power is to simulate the circuit[13]. Given the speed of circuit simulation, these techniques cannot be used to simulate long-enough input vector sequences to get meaningful power estimates. A Monte Carlo simulation based technique was proposed in [11]. The use of symbolic simulation in order to produce a set of boolean functions representing conditions for switching at each gate in the circuit has been proposed in [2][14]. Other approaches [3][4][9] have been proposed that require the user to specify the typical behavior at the circuit inputs using probabilities. These techniques allow the user to cover a large set of possible input patterns with little effort. The switching activity is propagated along the network from the inputs and the activity for all nodes in the network are estimated. The transition density concept was introduced in [4]. The model in [4] however did not take into account signal correlation. Recently, a way to consider signal correlation was proposed in [3]. In [8] it was shown that hazard contribution to power dissipation in CMOS ICs (glitch power) cannot be neglected.

Presently none of the techniques on improving transition density consider multiple input switching or glitching contributions to the signal activity. This paper incorporates both multiple input switching and glitching into the power estimation model.

The rest of the paper is organized as follows. Section 2 presents the definitions for the terms used in the paper. Section 3 discusses the activity estimation using the zero delay model, in particular multiple input switching. Section 4 discusses the activity estimation using the unit delay model. In section 5 we demonstrate the results from our implementation. Section 6 concludes with comments on future work.

2 Power model definitions

We define some commonly used terms in the paper.

- *Signal probability* of a node is defined as the duty cycle of the signal or the probability of the signal being ONE in a vector event.
- *Switching probability* of a node is the probability of the signal switching from one state to another.
- *Transition density* [4] has been defined as the "average switching rate" at the gates in the circuit. This is denoted by $D(f)$. The equation for power can be defined as

$$Power = 0.5{V_{dd}}^2 \sum_{f \in N} C_f D(f)$$

  where f is an internal or output node in the circuit, V_{dd} is the supply voltage and C_f is the capacitance at node f and N is the set of internal and output nodes. A *vector event (ve)* is a change of the input vector to the circuit. *Normalized transition density* is the number of transitions per vector event or the amount of switching per vector event. This is similar to switching probability and both the definitions are used interchangeably. Thus, the equation for power can also be rewritten as

$$Energy/ve = 0.5{V_{dd}}^2 \sum_{f \in N} C_f D(f)$$

  An uncorrelated input with this definition has a normalized switching density of 0.5 *events per vector event(epv)*. The clock has a normalized switching density of 1 epv.
- In *Zero delay model* all gates are assumed to have zero delay and in *unit delay model* all gates have unit delay.
- *Glitching* is the spurious transition(s) which occur when the signal switches more than the required logic switching. A *static hazard* exists if a signal that should remain constant changes twice(in opposite directions). It is a *0-hazard* if the signal should stay at 0, and a *1-hazard* if the signal should stay at 1. A *dynamic hazard* exists if a signal which should change its state once changes multiple times.

0

32nd ACM/IEEE Design Automation Conference ®

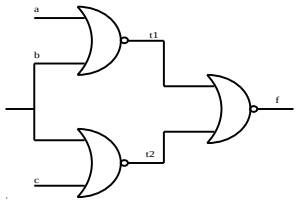

Figure 1: Overestimation of switching activity

3 Activity estimation

Estimation techniques under two delay models - zero delay model and the unit delay model are described.

3.1 Zero delay model

The zero delay model is used to calculate the switching activities due to logic alone. The transitions are assumed to be instantaneous, therefore glitching cannot occur.

3.1.1 Previous work

It was shown in [4] that transition density $D(f)$ is $D(f) = \sum_{x \in I} p(df/dx)D(x)$ where f is the output of a circuit and x is an input of the circuit, df/dx is the *boolean difference* of f with respect to x and I is the set of all inputs. $D(x)$ is the transition density for node x. There are a couple of problems with this definition for $D(f)$.

1. It only considers single input switching.

2. The partitioning of the nodes is at the lowest level (i.e., individual gates) and hence does not take into account the correlation between the signals.

Attempts have been made to take care of the correlation by partitioning[3] however multiple input switching has not been considered. Consider Figure 1. The inputs nodes (a, b, c) are assumed to have switching probabilities 0.5 and signal probabilities of 0.5. The node at $t1$ is $\overline{a+b}$ and has switching probability of 0.5 and signal probability of 0.25. Similarly the node at $t2$ is $\overline{b+c}$ and also has switching probability of 0.5 and signal probability of 0.25. If the model proposed by [4] is used then the node at f will be calculated to have a switching probability of 0.75 and signal probability of 0.5625. A flaw with this gate level partitioning is that signals $t1$ and $t2$ are correlated. The partitioning of circuits so as to "see" independent signals only is proposed in [3] and this would estimate the switching probability at node f to be 0.625 and the signal probability to be 0.625. This estimation model is still inaccurate because it overestimates the switching probability, since it considers switching by single inputs only and does not take into account simultaneous switching by multiple inputs. The switching probability derived from experiments using logic simulation(IRSIM) is 0.47, demonstrating the lack of accuracy in the single input switching model. We have derived a model which takes simultaneous switching into account and also the correlation between the intermediate signals.

Given a circuit realizing function f with input vector $I = (x_0, x_1..x_{n-1})$, the *boolean difference* df/dx_i of a circuit with respect to one of the inputs x_i is defined to be

$$df/dx_i = f(x_i) \oplus f(\bar{x_i}) = f_{x_i=0} \oplus f_{x_i=1}$$

where $\oplus$ is the logical XOR operation. It was first suggested in [4] to calculate the boolean difference by using BDDs(Binary Decision Diagrams)[10]. This is a very efficient tool to carry out probability computation. The scan function of the BDD package can be used to compute the probabilities at the nodes. Similarly the boolean difference for two inputs is defined as

$$df/d(x_i x_j) = f_{x_i=0,x_j=0} \oplus f_{x_i=1,x_j=1}$$

Several properties of the boolean difference are listed in [5]. These define the boolean difference functions for operations NOT, AND, OR and XOR. A complex function is recursively computed by breaking down the function into smaller ones. The boolean difference can hence be computed in a bottom-up fashion.

3.1.2 Activity considering multiple input switching

Given a circuit realizing function f with input vector $I = (x_0, x_1..x_{n-1})$ the switching probability at the output f is given by

$$D(f) = \frac{1}{{}^nC_1} \sum_{x_i \in I} p(\frac{df}{dx_i}) D(x_i) +$$
$$\frac{1}{{}^nC_2} \sum_{x_i, x_j \in I} p(\frac{1}{2}(\frac{df}{d(x_i x_j)} + \frac{df}{d(\bar{x}_i x_j)})) D(x_i) D(x_j) +$$
$$\frac{1}{{}^nC_3} \sum_{x_i, x_j, x_k \in I} p(\frac{1}{4}(\frac{df}{d(x_i x_j x_k)} + \frac{df}{d(x_i x_j \bar{x_k})} +$$
$$\frac{df}{d(x_i \bar{x}_j x_k)} + \frac{df}{d(\bar{x}_i x_j x_k)})) D(x_i) D(x_j) D(x_k)$$
$$+ \cdots \quad (1)$$

The first term is the contribution of signal input switching, the second term of two input simultaneous switching, the third of three input simultaneous switching and so on.

Single input switching

The single input switching probability of f is

$$\sum_{x_i \in I} p(f_{x_i=0} \oplus f_{x_i=1}) D(x_i)$$

Using the boolean difference definition and since there are ${}^nC_1$ ways to select signal x_i the contribution due to single input switching for all inputs to the overall switching is

$$\frac{1}{{}^nC_1} \sum_{x_i \in I} p(\frac{df}{dx_i}) D(x_i)$$

Multiple input switching

The condition for two input simultaneous switching is derived and we will extrapolate the result for multiple input switching.

The two input switching probability of f is

$$\sum_{x_i x_j \in I} p(\frac{2}{4}(f_{x_i=0 x_j=0} \oplus f_{x_i=1 x_j=1} + f_{x_i=0 x_j=1} \oplus f_{x_i=1 x_j=0})) D(x_i) D(x_j)$$

Using definition of boolean difference and since there are ${}^nC_2$ ways to select signals x_i and x_j, the contribution due to two input switching to the overall switching probability is

$$\frac{1}{{}^nC_2} \sum_{x_i x_j \in I} p(\frac{1}{2}(\frac{df}{d(x_i x_j)} + \frac{df}{d(\bar{x_i} x_j)})) D(x_i) D(x_j)$$

Similarly the three input switching contribution is

$$\frac{1}{{}^nC_3} \sum_{x_i, x_j, x_k \in I} p(\frac{1}{2^{3-1}}(\frac{df}{d(x_i x_j x_k)} + \frac{df}{d(x_i x_j \bar{x_k})} +$$
$$\frac{df}{d(x_i \bar{x}_j x_k)} + \frac{df}{d(\bar{x}_i x_j x_k)})) D(x_i) D(x_j) D(x_k)$$

The *support set* f_{sup} of a function f are the set of variables f depends on. Using the support set allows us to partition

Algorithm calculateMultipleInputDensity(f, I, G)

```
/* I is the primary input vector */
/* f is the output vector */
/* G is the set of gates*/
(∀i ∈ I)i_sup := ∅;
(∀g ∈ G)g_visited := FALSE;
nodeList := I_fanOuts;
∀k ∈ nodeList
    /*k_sup is the support set for node k*/
    k_sup := ∅
    ∀i, j ∈ k_fanIns
        if(i_sup ∩ j_sup ≠ ∅)
            k_sup := k_sup ∪ i_sup ∪ j_sup;
        else
            k_sup := k_sup ∪ i ∪ j;
    k_formula :=calculateFormula(k, k_sup);
    k_density :=evalDensity(k_formula, k_sup);
    ∀g ∈ k_fanOuts
        /*if all gates driving gate g have been visited
         then add g to nodeList */
        if((∀h ∈ g_fanIns)h_visited = TRUE)
            nodeList := nodeList ∪ g;
    k_visited := TRUE;
```

Figure 2: Algorithm for calculating multiple input density

Table 1: Activity estimation results (zero delay)

Gates	*Zero Delay Model*		*IRSIM results*
	Single input switching	*Multiple input switching*	
nand2	0.5	0.375	0.38
nand3	0.375	0.21875	0.23
nand4	0.25	0.1172	0.15
nor2	0.5	0.375	0.38
nor3	0.375	0.21875	0.20
nor4	0.25	0.1172	0.13
xor2	1.0	0.5	0.48

the circuit so that every node only depends on the directly correlated signals. The algorithm to calculate the minimal support set so that each partition packs more correlated nodes is shown in [6]. The pseudo-code for the multiple input density calculation algorithm is given in Figure 2.

The procedure *calculateFormula* calculates the formula for the node using the support sets and is implemented using BDDs. The procedure *evalDensity* evaluates the switching density given the multiple input switching equation(Equation 1). The algorithm makes one pass through all the gates of the circuit. The procedure *evalDensity* depends exponentially on the size of the support set. For a tree based circuit support sets for each node consist only of its immediate inputs.

3.1.3 Examples

Table 1 shows the accuracy obtained using our multiple input switching and signal correlation model along with single input switching model with respect to the experimental results. It is obvious that multiple input switching gives more accurate results. The experimental results were corroborated with IRSIM results by calculating the switching activity for about 400 vectors. The IRSIM linear model is used which models transistors as a resistor in series with a voltage controlled switch. This model uses a single-time-constant computed from the resulting RC network and uses a two-time-constant model to analyze charge sharing and spikes[6]. All the gates were implemented in CMOS and analyzed by IRSIM by counting the transitions.

The following example shows the calculation of switching activity for the NAND gate. The input switching probability is 0.5 and the signal probability is 0.5.

2 input nand gate

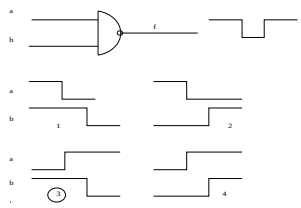

Figure 3: Glitching

A 2 input nand gate implements $f = \overline{ab}$. The boolean differences are $df/da = b$ and $df/db = a$. The second order boolean differences are $df/d(ab) = 1$ and $df/d(\bar{a}b) = 0$. The switching probability of f is thus

$$\frac{1}{2}(p(a)D(b) + p(b)D(a)) + \frac{1}{1}(p(1/2(1+0))D(a)D(b))$$

The switching probability of the output is calculated to be 0.375.

3.2 Unit delay model

The contribution of hazards to power dissipation (glitch power) is a critical issue in power estimation and low power circuit design[8]. Glitching of a gate depends on several factors.

1. The difference in delay of signals arriving at a gate.
2. The sensitivity of the gate to the signal arrivals.

It is known that some gates are more sensitive to glitching than others. This sensitivity also depends on the arrival sequence of the signals at the gates. Using the unit delay model one can estimate the switching due to glitching. We assume in a unit delay model that all gates have same propagation delay(1 gd). The arrival times at the gates can simply be calculated by counting the number of gates encountered in the path. The first condition is easily detected when the signals arrive at the gates differing by more than certain δ time. We consider $\delta \geq 2gd$ as the condition for glitching in our implementation. The second condition is to figure out how many of these transitions differing in time actually cause a glitch. *Glitching sensitivity (gs)* of a gate is defined as the probability of a gate glitching under a given arrival sequence of input vectors. We propose the following algorithm to calculate the glitching sensitivity of the gates given the arrival times to the inputs of gates. Step functions (and their inverse) are applied on the inputs of the gate. The step functions are staggered by their arrival times. The gate transforms the steps at the input into steps or pulses on the output depending on the gate and the arrival sequence. The number of pulses which occur on the gate divided by the number of all possible input step patterns (2^n) gives the glitching sensitivity of the gate. For example, assume switching across a NAND gate (Figure 3). If signal a arrives at time t and signal b arrives at time $t + \delta$ then glitching can only occur if signal a is rising and signal b is falling which is one of the possible four conditions (3rd condition). Thus the glitching rate is .25 times the switching probabilities of signal a and signal b of a NAND gate. By observing small gates(NAND/NOR/OR/AND) it is clear that they glitch only 0.25 times the possible glitch conditions, however for a gate like XOR glitches are 1.0 times the possible glitch conditions if the arrival time at the inputs differ. Assuming input signal probabilities 0.5 and input switching probabilities of 0.5 then transition density of a NAND gate is 0.375(Section 3.1.3).

$pt(f) = 0.375; ptg(a) = 0.5; ptg(b) = 0.5$

$ptg(f) = pt(f) + 2 * ptg(a) * ptg(b) * .25$

$ptg(f) = 0.5$

where $pt(f)$ is the transition density at f without glitching and $ptg(f)$ is the transition density at f with glitching.

Algorithm calculateGlitchingSensitivityOfGate(f, I, α)

```
/* I is the primary input vector */
/* f is the output vector */
/* α is the arrival vector*/
pulseCount := 0;
∀ 2^n input assignments of I with u(α) and ¬u(α)
        evaluate f(I);
        ∀ pulse component ∈ f(I)
                pulseCount := pulseCount + 1;
/*set glitching sensitivity of gate f*/
f_gs := pulseCount/2^n;
```

Figure 4: Algorithm to calculate glitching sensitivity

The logic operations(NOT, AND, OR) on step, pulse and step/pulse functions is given in [5].

3.2.1 Activity considering glitching

Let $I = (x_0, x_1..x_{n-1})$ be the input vector with arrival vector $\alpha = (\alpha_0, \alpha_1..\alpha_{n-1})$ at a gate with n inputs satisfying function f. An ordering on the arrival vector at the gate is known. For each 2^n combinations of the input vector with step response and its inverse, $f(I)$ is calculated using rules from [5]. The total number of pulse components for each input assignment is then incremented for all inputs. The probability of glitching is calculated by dividing the total number of pulse components for all inputs by the total number of input patterns 2^n. For most gates in practice n is small ($n \leq 5$), so the algorithm given in Figure 4 can be used to calculate the sensitivity for most of the commonly used gates and then used by table lookup in the estimation.

The pseudo-code to calculate the sensitivity and to update the activities are given in Figure 4 and Figure 5.

After the glitching sensitivity has been calculated, the new activity factor can be calculated as

$$D^1(f) = D^0(f) + 2 * f_{gs} * D^1(x_i) * D^1(x_j) \cdots \qquad (2)$$

where $x_i x_j ..$ are the inputs which arrive at different times and f_{gs} is the glitching sensitivity of the gate with the given arrival times. D^0, D^1 are the zero delay switching density and the unit delay switching density respectively. The factor 2 is multiplied because glitching causes two transitions at the output (one in each direction). The procedure *updateDensity* (Figure 5) updates the transition density for unit delay model with the Equation 2. The procedure *calculateUnbalancedGates* (Figure 5) calculates the arrival times on the gate and computes whether the gate is unbalanced ($\delta \geq 2$). It is assumed that glitching propagates through the network.

Table 2 shows the circuit activity of simple gates when considering glitching under various arrival patterns. α,β and γ are the arrival times at different input. The activity is corroborated using IRSIM for about 400 vectors. Note that for a gate implementation like XOR glitching could double the activity factor of the gate.

4 Experimental results

We have incorporated the algorithms for multiple input switching and glitching estimation into an estimator (PSIM) built using a BDD package. This estimator gives two sets of activity results, one considering only multiple input switching and the other considering multiple input switching and glitching. This estimator operates on the logical description (eqn format) of the circuit. This estimator also gives a *power factor*

Algorithm calculateDensityWithGlitching(f, I, G)

```
/* I is the primary input vector */
/* f is the output vector */
/* G is a set of gates*/
∀g ∈ G
    g_visited := FALSE;
    g_balanced := TRUE;
/* Compute unbalanced gates and their arrival times */
calculateUnbalancedGates();
nodeList := I_fanOuts
∀k ∈ nodeList
    if(k_balanced = FALSE)
        /* k_density is the zero delay density at gate k*/
        /* k_α is the arrival vector at gate k*/
        /* k_gs is the glitching sensitivity of gate k*/
        k_density := updateDensity(k_density, k_α, k_gs);
    ∀g ∈ k_fanOuts
        if((∀h ∈ g_fanIns)h_visited = TRUE)
            nodeList := nodeList ∪ g;
    k_visited := TRUE;
```

Figure 5: Algorithm to calculate activity with glitching

Table 2: Activity estimation results (unit delay)

Gates	*Arrival seq.*	*Glitching sensitivity*	*Unit delay model (multiple input)*	*IRSIM* results
nand2	$\alpha = \beta$	0.0	0.375	0.38
	$\alpha < \beta$	0.25	0.5	0.47
nand3	$\alpha = \beta = \gamma$	0.0	0.21875	0.19
	$(\alpha = \beta) < \gamma$	0.125	0.2795	0.29
	$\alpha < \beta < \gamma$	0.25	0.3425	0.36
nor2	$\alpha = \beta$	0.0	0.375	0.38
	$\alpha < \beta$	0.25	0.5	0.48
nor3	$\alpha = \beta = \gamma$	0.0	0.21875	0.23
	$(\alpha = \beta) < \gamma$	0.125	0.2795	0.30
	$\alpha < \beta < \gamma$	0.25	0.3425	0.35
xor2	$\alpha = \beta$	0.0	0.5	0.48
	$\alpha < \beta$	1.0	1.0	0.90

Table 3: Circuits used for comparison

Circuits	*Inputs*	*Outputs*	*Gates*
hpar	9	2	17
mux8	11	1	49
hcomp	11	3	51
rca4	8	5	34
cla4	9	5	76
sd	6	5	66
rca8	9	2	74
halu	14	8	93

Table 4: Comparison of SPICE results with PSIM

Circuits	*Power*			*CPU Time*	
	SPICE (mW)	*PSIM (power factor) (multiple inputs)*	*(& glitching)*	*SPICE* (secs)	*PSIM* (secs)
hpar	0.1399	16.01	17.23	1500	2
mux8	0.3135	19.58	24.2	2880	54
hcomp	0.3605	20.01	27.32	3600	32
rca4	0.4402	20.13	35.26	1200	3.5
cla4	0.7222	55.69	68.45	6420	9
sd	0.898	85.01	90.27	6600	12.5
rca8	1.080	87.54	101.34	7440	93
halu	1.18	92.03	105.0	2440	72

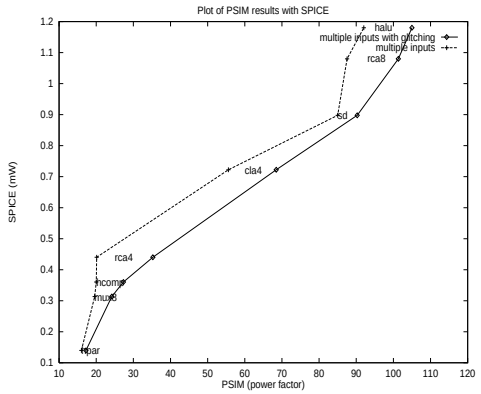

Figure 6: Plot of SPICE results with PSIM

for the circuit which is

$$Powerfactor = \sum_{f \in allnodes} fanouts_f D(f)$$

where $fanouts_f$ are the fan-outs of signal f. The $fanouts_f$ is proportional to the gate capacitance driven by signal f. Power is hence proportional to the $powerfactor$.

The glitching sensitivity of gates which occur frequently within the circuits are evaluated. This is used to compute the glitching for gates within the large circuits by looking up the right sensitivity values depending upon the actual arrival times of the inputs at the gates.

The results for some circuits and their characteristics are given in Tables 4 and 3. The power factor from PSIM is calculated with glitching and without glitching (both of them including multiple input switching) and plotted along with actual power values (mW) obtained from 100 random runs of SPICE (Figure 6). The dotted curve denotes the power factor estimated considering multiple input switching only and the solid curve shows the power factor estimated when considering multiple input switching and glitching. The difference between the two curves represents the glitching power contribution. It can be noted that circuits such as rca4, rca8 and halu have more switching energy loss due to glitching. This can be attributed to the imbalanced paths in these circuits. Comparisons between CPU times of SPICE and PSIM are shown in Table 4. We see that the CPU times for PSIM are several orders of magnitude faster than SPICE.

5 Conclusions and future work

We have described a quick and accurate method of estimating switching activity from the logical description of the combinational circuit.

We would like to extend the work to use actual gate delays. This will help in determining when glitches get damped when propagating through the circuit.

Switching activity results can be used as an input to other CAD tools in order to make performance based decisions on partitioning, routing or transistor sizing.

Switching activity results could be used to compare the power consumption of two functionally equivalent but logically different implementations of a function. Currently the estimation is based only on gate level implementations however this could be extended to transistor level implementations either by extending the eqn syntax of the circuit with "if then else" statements or by including transistor level switching activity by reading from the transistor-level description of the circuit. A fallout of this work would be to compare different logic implementation styles (e.g., static v/s dynamic).

References

[1] A. Chandrakasan, T. Sheng , R. W. Brodersen, "Low Power CMOS Digital Design", *IEEE J. of Solid-State Circuits*, Apr 1992, pp. 473-484.

[2] Abhijit Ghosh, Srinivas Devadas, Kurt Keutzer, Jacob White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits", *29th IEEE Design Automation Conference*, 1992, pp. 253-259.

[3] B. Kapoor "Improving the accuracy of circuit activity measurement", *IWLPD '94 Workshop Proceedings*, pp 111-116.

[4] F. Najm, "Transition Density, A New Measure of Activity in Digital Circuits", *IEEE Trans. on Computer Aided Design*, Feb 1993, pp. 310-323.

[5] Huzefa Mehta, Robert Michael Owens, Mary Jane Irwin "Accurate Estimation of Combinational Circuit Activity", *Tech Report CSE-95-007.*

[6] IRSIM USERs Manual.

[7] J. Monteiro, S. Devadas, B. Lin, C-Y Tsui, M. Pedram, "Exact and approximate methods of switching activity estimation in sequential logic circuits", *IWLPD '94 Workshop Proceedings*, pp 117-122.

[8] L. Benini, M. Favalli, B. Ricco, "Analysis of hazard contributions to power dissipation in CMOS ICs", *IWLPD '94 Workshop Proceedings*, pp 27-32.

[9] M. A. Cirit, "Estimating Dynamic Power Consumption of CMOS Circuits" *Proceedings of the Int'l Conference on Computer-Aided Design*, Nov 1987, pp 534-537.

[10] R. Bryant, "Graph Based Algorithms for Boolean Function Manipulation" *IEEE Transactions on Computers*, volume C-35, August 1986, pp. 677-691.

[11] R. Burch, F. Najm, P. Yang, T. Trick, "McPOWER: A Monte Carlo Approach to Power Estimation" *Proceedings of the Int'l Conference on Computer-Aided Design*, November 1992, pp 90-97.

[12] R. Murgai, R. Brayton, Alberto Sangiovanni-Vincentelli, "Decomposition of Logic Functions for Minimum Transition Activity", *IWLPD '94 Workshop Proceedings*, pp 33-39.

[13] S. M, Kang, "Accurate Simulation of Power Dissipation in VLSI Circuits", *IEEE J. of Solid-State Circuits*, Oct 1986, volume 35, pp 889-891.

[14] Srinivas Devadas, Kurt Keutzer, Jacob White, "Estimation of Power Dissipation in CMOS Combinational Circuits Using Boolean Function Manipulation", *IEEE Trans. on Computer Aided Design*, March 1992, pp. 373-383.