

# Power-Supply Noise in SoCs: ATPG, Estimation and Control

Mehrdad Nourani and Arun Radhakrishnan

Center for Integrated Circuits & Systems

The University of Texas at Dallas, Richardson, TX 75083

{nourani,axr039100}@utdallas.edu

## Abstract

*Noise on the power-supply lines has become a critical factor especially in low-voltage designs as excessive noise may cause intermittent functional error and eventually system failure. Having a realistic picture of the worst case noise throughout a chip is important as we can devise a remedy to alleviate the problem before it becomes too late. This paper provides a fast automatic pattern generation method to estimate the maximum simultaneous switching noise for a non-embedded core. When the cores are integrated within a SoC, the combined noise is far from a simple additive behavior. We offer a heuristic to estimate the power-supply noise in core-based SoCs. This heuristic can be employed to minimize the number of power-supply lines/pins while keeping the noise under control. The experiments using ISCAS'85 benchmarks verify that our power-supply noise methodology is fairly fast and accurate and can be used for effective power-supply distribution.*

## 1 Introduction

### 1.1 Prior Work

In deep submicron (DSM) technology, large fluctuations in supply voltage can affect functionality of some gates and eventually may lead to failure [1]. To be on the safe side, designers often perform pre-synthesis analysis to get a better picture of power-supply noise (PSN). An interconnect and circuit modeling technique for power-supply noise analysis is offered in [2] that allows hierarchical modeling and identifies hot-spots. The work in [3] builds a charge/discharge current and output voltage library to estimate PSN. Reference [4] focused on detecting power-supply noise in synchronous systems exceeding a tolerance bound. Finding the maximum value PSN or its corresponding vector requires exhaustive spice simulation. Researchers presented various ways of estimating PSN in limited forms and accuracy. To name a few, [5] uses scaling model, [6] employs a simulated switching model of power bus and [7] focuses on distributed power network model. Authors in [8] present the generation and characterization of three different types of noise induced by electrostatic discharge in power-supply systems.

To speedup the PSN analysis and test generation process some works exploited the concept of random search. For example [9][10] uses a genetic algorithm (with random basis) to stimulate the worst case PSN. Another group of researchers [11] pre-characterize cells using transistor level simulators and annotate the information into PSN analysis phase. A technique

for vector generation for power-supply noise estimation and verification is offered in [12]. The authors used a genetic algorithm to derive a set of patterns producing high power-supply noise. A pattern generation method to minimize the PSN effects during test is presented in [13].

A PSN monitor circuit is presented in [14] by which the authors claimed to catch high resolution (100 ps) PSN at the power/ground lines. Power-supply distribution model to control PSN has been also reported in the literature. The model presented in [15] identifies the hot-spots on the chip and optimizes power-supply distribution to minimize the noise. A cascaded power/ground ring for on-chip power distribution is proposed in [16]. Another methodology for multiple power-supply distribution systems are presented in [17]. The authors in [18] argue that the peak PSN can be significantly reduced based on modules physical correlations. They have proposed a power-supply noise aware floorplanning methodology.

### 1.2 Main Contribution

The key novelty in our approach is twofold. First, we identify three design metrics (i.e. level, fan-in, and fan-out) that capture realistic PSN to a large extent. We will show that careful ordering of these metrics will help PSN analysis to achieve more accuracy. Although the goal of fault simulation per se is not pursued, we employ a conventional fault simulator to efficiently provide crucial data for PSN analysis. Then, we apply a greedy algorithm that interacts with a fault simulator can accurately and quickly construct pattern pairs that simulate the worst case PSN based on circuit topology and regardless of its functional or testing mode. Second, we present an analytical way of combining PSN of non-embedded cores to quickly and accurately estimate the PSN of the SoC. To show the effectiveness of our method in system-on-chip design, we will use such analysis to group cores and design a power-supply distribution network while keeping PSN under control.

### 1.3 Paper Organization

The rest of this paper is organized as follows. Section 2 describes the motivation and the power-supply noise modeling and design parameters affecting it. The analysis methodology and test pattern generation to stimulate the worst case PSN for non-embedded cores will be discussed in Section 3. Section 4 expands the PSN analysis to embedded case (i.e. core integration within a SoC environments). In Section 5 we elaborate on a methodology to partition SoC into group of cores that use separate power supply (e.g.  $V_{dd}$  and  $V_{ss}$ ) lines/pins required to

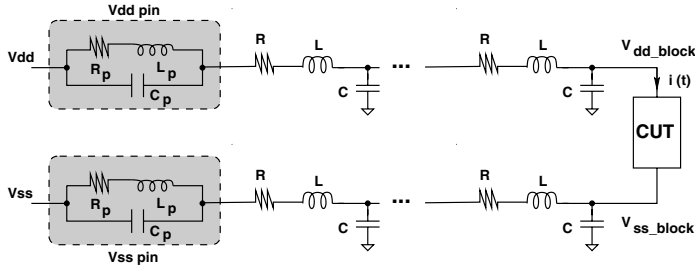


Figure 1: RLC Model for the  $V_{dd}$  and  $V_{ss}$  pin

limit PSN. The experimental results are presented in Section 6. Finally, the concluding remarks are in Section 7.

## 2 PSN Modeling

### 2.1 Power-Supply Model

In an ideal power-supply line parasitic elements are not considered, and hence drawing current from it does not produce fluctuation in voltage. Since power-supply is distributed in the circuit through wires, they contain parasitic elements. Noise is mainly created in the power-supply lines due to the resistive and inductive parasitic elements. Inductive noise, also known as  $di$  noise, is caused due to the instantaneous change in current drawn from the power-supply. Inductive noise becomes significant in high frequency designs due to the sharp rise and fall times. High frequency switching causes the current to be drawn (from the power-supply) for very short duration (usually in hundred's of picoseconds) causing very high  $di/dt$ . Resistive noise (IR drop) is dependent on the current drawn from the power-supply. Hence the noise voltage in the power-supply, shown as  $PSN(t)$  or simply  $PSN$  (or  $V_{noise}$  in some of the curves) through-out this paper, is given by:

$$PSN(t) = L \cdot \frac{di(t)}{dt} + R \cdot i(t) \quad (1)$$

Power-supply pins and wires were modeled as a series of RLC elements shown in Figure 1. Coupling between  $V_{dd}$  and  $V_{ss}$  is ignored. Power-supply noise is calculated using:

$$PSN(t) = (V_{dd} - V_{dd\_block}(t)) - (V_{ss} - V_{ss\_block}(t)) \quad (2)$$

### 2.2 Metrics Affecting PSN

Among several factors that affect power consumption - level, fan-out and fan-in a circuit are the most important ones. A brief justification of these three metrics follows:

1. **Level (L Metric):** The level of a node in a circuit is defined as the maximum number of gates a pattern has to propagate through to reach it. All primary inputs are considered as level 0 nodes, output node of a gate whose inputs are primary inputs is considered level 1. Intuitively, switching a lower level node will probably cause more

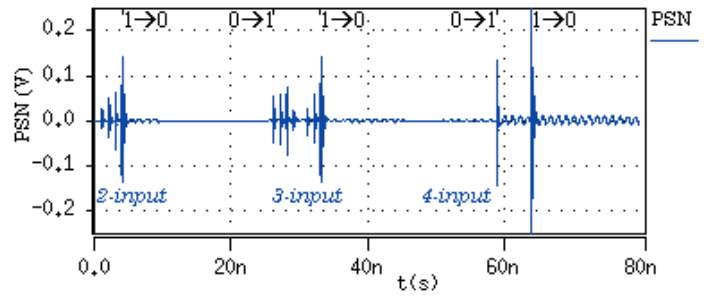


Figure 2: Simulation of NAND gates with fan-in 2, 3, and 4

switching in the circuit compared to a higher level node. Input pattern used to force a lower level node from  $1 \rightarrow 0$  or  $0 \rightarrow 1$  is likely to have more don't-cares and hence allowing us to combine the more number of test patterns together (allowing to switch more number of gates) [19].

2. **Fan-Out (O Metric):** The switching time of a gate is inversely proportional to its fan-out [20]. Lower fan-out gates have less capacitance at the output hence it will switch faster producing a smaller  $dt$  (and thus higher  $di/dt$ ).
3. **Fan-In (I Metric):** To maximize  $i(t)$  and  $di(t)/dt$ , it is essential to switch the maximum number of transistors. Hence switching a gate with large fan-in would induce more noise than a gate with lower fan-in. To show the importance of fan-in ignored by many researchers, NAND gates with fan-in 2, 3 and 4 each with an inverter load was simulated ( $V_{dd} = 1.2V$ ,  $\lambda = 0.13\mu m$ ) using HSPICE simulator [21]. The output of each of the NAND gates were switched from  $0 \rightarrow 1 \rightarrow 0$ . It is evident from Figure 2 that the noise induced in the power-supply increases with fan-in. Among 2, 3 and 4 input NAND gates maximum noise is induced in the power-supply when the 4-input NAND is switching. Another interesting observation is that gates often don't show identical behavior for  $1 \rightarrow 0$  and  $0 \rightarrow 1$ . For example, for all 3 NAND gates, transition from  $1 \rightarrow 0$  produced more noise than  $0 \rightarrow 1$  transition of the same gate. Thus, in our model both transitions are checked in this order.

To generate a pattern to maximize power consumption, our algorithm uses a combination of the factors outlined in Section 3. One of these factors can be considered as the dominant rule to prioritize switching of certain gates over other and the others can be used as tie breaking criterion. Using these factors, there are a total of 6 possible combinations.

Table 1 illustrates the choices and the method names by which they will be referred to in rest of the paper. Second row in this table indicates the order in which the factors are considered, for example LIO indicates that level of the gate is criterion 1 (dominant), fan-in is criterion 2 and fan-out is the least important criterion. Among these methods, M5 and M6 are expected to switch more gates in roughly same time interval (same level).

Table 1: Order in which the metrics are chosen

Method	M1	M2	M3	M4	M5	M6
Order of Metrics	ILO	IOL	OLI	OIL	LOI	LIO

### 3 PSN Analysis and ATPG for Non-Embedded Cores

In any  $n$ -input circuit,  $2^n \cdot (2^n - 1) \approx 2^{2n}$  transitions are possible. Simulating all possible transitions is unrealistic and hence it becomes necessary to select a pattern pair without exhaustive simulation. Random pattern based simulation can also be used but randomness cannot guarantee maximum power in short time. Based on the characteristics explained in the previous section, we can maximize switching in the circuit by selecting a metric in which the gates should be prioritized (sorting criteria) and then generating patterns to switch the dominant gates in the sorted list. Other less important metrics can be used for tie-breaking or applying hill-climbing heuristics. This approach efficiently generates a pattern pair  $\{V1, V2\}$  such that switching from  $V1$  to  $V2$  induces the maximum amount switching in the circuit.

#### 3.1 Algorithm

The algorithm shown in Figure 3 was implemented in C code. There are three phases to this algorithm - Initialize, Generate and Analyze. In Phase 1, Verilog or VHDL implementation of the circuit is parsed and stored internally in a data structure.

In Phase 2, stuck-at fault patterns for all possible nodes in the circuit are generated. A test vector used to detect s-a-0 at a node forces that node to 1 similarly a test vector used to detect s-a-1 at a node forces that node to 0. Therefore, transition from  $1 \rightarrow 0$  on a node can be viewed as a transition in the vectors that can detect s-a-0 and s-a-1 respectively. Hence this phase can be performed by implementing a typical stuck-at fault detection algorithm such as PODEM, D algorithm, etc. Note that here the stuck-at-fault model is used solely for the purpose of getting test vectors for each node. Other fault models, such as transition-fault or delay-fault models that are capable of finding pattern pairs to stimulate  $0 \rightarrow 1$  or  $1 \rightarrow 0$  on nodes, can also be used. In any case organizing patterns is performed by the PSN analysis algorithm.

Instead of implementing stuck-at fault detection algorithms, we used TetraMAX [21] a tool from Synopsys to obtain stuck-at fault patterns. TetraMAX generates the pattern required for a node s-a-0 and s-a-1 such that the input test pattern activates and propagates the fault. Note carefully, for the purpose of our work propagation of the fault to the output is not required; hence implementing a detection algorithm rather using TetraMAX might reduce run-time of the program. If input pattern for s-a-0 or s-a-1 cannot be determined (undetectable fault), the gate is removed from the list.

Analysis phase (Phase 3) is the focal point of the algorithm.

#### Phase 1 (Initialize)

- i. Parse HDL file to build a list of gates

#### Phase 2 (Generate)

- i. Generate s-a-0 and s-a-1 patterns for all nodes using TetraMax.
- ii. Remove any gates whose s-a-f is undetectable from the list.

#### Phase 3 (Analyze)

- i. Sort gates according to M5 and M6 in Table 1.
- ii. Initialize the  $\{V1, V2\}$  to all don't-cares.
- iii. While  $\{V1, V2\}$  contains don't-cares or untried gates in the list
  - { Try adding  $\{s-a-0, s-a-1\}$  patterns of current gate to  $\{V1, V2\}$ .
  - If  $\{s-a-0, s-a-1\}$  failed
    - Try adding  $\{s-a-1, s-a-0\}$  pattern of current gate to  $\{V1, V2\}$ .
    - If both tries failed (i.e. gate will not switch)
      - Continue to next gate in list.
  - }
- iv. Replace all don't-cares in  $\{V1, V2\}$  with 0 or 1 randomly.

Figure 3: Pseudocode of the PSN analysis algorithm.

As shown in Figure 3, Phase 3 consists of sorting the list of gates and then combining the patterns to form one transition pair. The sorting algorithm is based on methods M5 and M6 (LOI and LIO) discussed in Section 2. The gates at the lowest level were given the highest priority, and then the largest fan-in or smallest fan-out was considered.

Formation of test vector pair starts by initializing the pattern pair  $\{V1, V2\}$  to all don't-cares (X). Then, the program tries to combine the input vector for  $\{s-a-0, s-a-1\}$  of each gate (i.e. pattern for  $1 \rightarrow 0$ ) in the sorted list to the current test vector  $\{V1, V2\}$ . If the pattern required for  $1 \rightarrow 0$  transition of a node conflicts with the current  $\{V1, V2\}$ , pattern from  $0 \rightarrow 1$  transition will be tried. If both are not possible, the program continues to try to add the test patterns corresponding to the next gate in the sorted list to  $\{V1, V2\}$ . This process (adding the input-vectors of each gate in the list) continues till test vector  $V1$  or  $V2$  does not contain anymore Xs or all nodes in the list have been tried. Any remaining Xs are replaced by random 1s and 0s.

To determine the worst case PSN and find the corresponding pattern pair, our algorithm gets help from stuck-at fault analysis and the functionality of the system rather than other highly variable parameters. In other words, parameters that contribute to noise, excessive transitions and integrity loss in a system, such as physical dimensions, couplings, process variation and so on, are considered indirectly through PSN analysis. This provides a speedy processing mechanism while the factors are taken into account indirectly. This program produces a pair of test vectors according to the selected method (M1 through M6) that stimulates noise on the power-supply that is fairly close to the maximum value. Since ATPG based stuck-at fault pattern generation are several orders of magnitude faster than HSPICE simulation [21], it is possible to reduce the simulation time to seconds rather than hours.

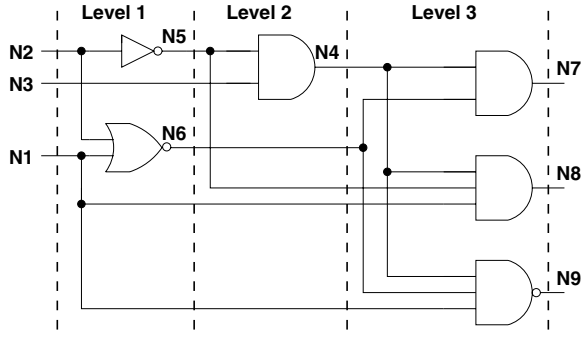


Figure 4: Gate-level schematic of an example circuit

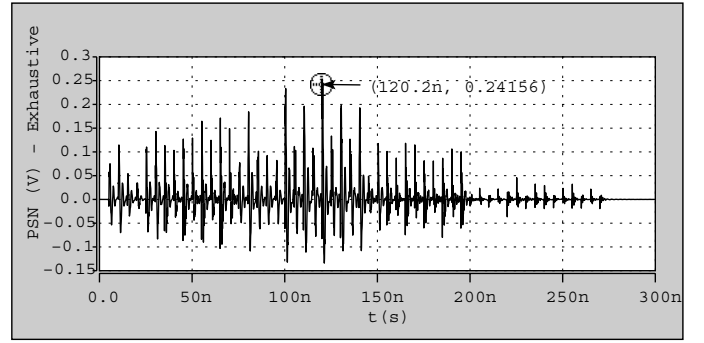
Table 2: Vector pair  $\{V1, V2\}$  formation for the example circuit of Figure 4

Node (Gate)	Trans.	S/F	Pattern Pair Chosen	$\{V1, V2\}$
				$\{XXX, XXX\}$
N6(NOR2)	$1 \rightarrow 0$	S	$\{00X, X1X\}$	$\{00X, X1X\}$
N5(NOT)	$1 \rightarrow 0$	S	$\{X0X, X1X\}$	$\{00X, X1X\}$
N4(AND2)	$1 \rightarrow 0$	S	$\{X01, X1X\}$	$\{001, X1X\}$
N7(AND2)	$1 \rightarrow 0$	S	$\{001, X1X\}$	$\{001, X1X\}$
N8(AND3)	$1 \rightarrow 0$	F		$\{001, X1X\}$
	$0 \rightarrow 1$	F		$\{001, X1X\}$

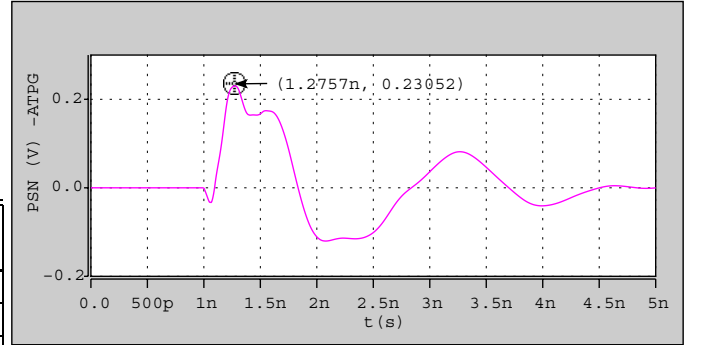
### 3.2 Example Circuit

Working of this algorithm is demonstrated by means of a small example circuit shown in Figure 4. Column 2 and 3 of Table 2 show the transition tried and its success (S) or failure (F) respectively. TetraMAX tool is used to obtain the stuck-at fault pattern pairs (Column 4). Gate N9 with undetectable stuck-at faults will be removed from the list. After the pattern generation phase, gates are sorted so as to prioritize based on a particular method (M1 to M6). Phase 3 of the algorithm using method M6 (LIO) is demonstrated in the last column. After initializing the  $\{V1, V2\}$  to  $\{XXX, XXX\}$ ,  $1 \rightarrow 0$  transition on node N6 chosen and  $\{V1, V2\}$  is updated to  $\{00X, X1X\}$ . Then  $1 \rightarrow 0$  transition on node N5 is selected and  $\{V1, V2\}$  changes to  $\{00X, X1X\}$ . It continues on to merge the patterns of as many gates as possible together till the list is complete or there are no more Xs in  $\{V1, V2\}$ . In this case N8 is the only node that is not added, but it is possible that if there existed nodes below (lower priority) N8 in the list and they could be added to  $\{V1, V2\}$  if their patterns don't conflict.

The example circuit was simulated in HSPICE simulator, using  $0.13\mu m$  technology. An exhaustive simulation of  $2^3 \cdot (2^3 - 1) = 56$  pattern pairs took roughly 5 times the amount of time needed to apply our PSN analysis algorithm, find and simulate the vector pair that induce maximum PSN. The ATPG based vector pattern produced almost the same peak noise as the exhaustive simulation in much shorter time as shown in Figure 5.



(a) Using exhaustive simulation.



(b) Using our ATPG.

Figure 5: PSN waveforms for example circuit in Figure 4

## 4 PSN Analysis for Embedded Cores

An accurate PSN analysis (Section 3) can be quite useful for core design as well as system level design. When applied at the core level, this allows designers to estimate and tune the design to meet power specification. At the system level, a known dilemma involves distribution of global interconnects (power, clock, etc). Pre-processing the individual cores (to be integrated into the system), provides the designer with PSN behavior. Such information can be used for many purposes, e.g. the worst case analysis, limiting PSN in design, choosing minimum number of power-supply lines/pins required and allocating cores to various power lines.

For simplicity, any time that we refer to  $PSN_i$  we mean the full PSN curve ( $PSN_i(t)$ ) obtained by feeding the pair of patterns that stimulate the worst case power-supply noise in  $Core_i$  obtained by ATPG method explained in Section 3. On the other hand,  $PSN_i^{MAX}$  is a scalar referring to the maximum noise value found in  $PSN_i$  curve.

### 4.1 Difficulties of PSN Analysis in SoCs

It's certainly easy to consider  $PSN_i^{MAX}$  values only. However, using one number to reflect the worst case scenario is quite pessimistic and overly conservative. While such conservatism may be useful for highly-reliable and mission-critical systems, it's an overkill for others. From the earlier discussion it's obvious that PSN is a data-dependent, circuit-dependent and time-sensitive phenomena whose behavior cannot be realistically captured with only one number.

What we suggest is to use the information annotated within

$PSN_i(t)$  curves more intelligently. With a more realistic view, one can assume the worst case scenario for all the cores will not happen at the same time. Unfortunately, there are two practical problems. First, accurate simulation of various combination of cores is very time consuming. While we can afford applying PSN analysis and ATPG for non-embedded cores, we cannot test all combinations of cores to find the best groupings and power-supply line/pin distribution. Second and more importantly, the behavior of cores, in terms of PSN distribution, in non-embedded and embedded cases are different. Specifically, the idleness of different cores, operating frequency, time delay between core activations/transition, number of levels and topology of cores and the occurrence of the peak noise, etc. all determine how the PSN in an embedded PSN will be.

At the system level it becomes harder to deterministically predict the switching activity at the higher levels. Hence generating patterns for the worst case PSN based on switching/PSN behavior of individual (non-embedded) core will not often provide the worst case PSN in an embedded system. On the other hand, accurate analysis of large multi-core SoC for PSN is not practical (due to size) or even possible (due to protecting the intellectual property rights). Hence, similar to integrating various blocks into a system based on the functional specifications (input-output characteristics) of these cores it is also necessary to account for the PSN created by the cores when integrating in a SoC. In particular, drawing too much current (due to simultaneous switchings) from any one supply can cause failure of power-supply wire. Therefore, such estimation is necessary to partition as to which cores are connected to available  $V_{dd}$  and  $V_{ss}$  pins or to calculate the minimum number of pins required to the retain PSN on all the pins below a certain threshold.

## 4.2 PSN Analysis Strategy

PSN behavior of non-embedded cores are significantly different from the embedded scenarios. When several cores are added to a system, several factors change including: 1) wire length required to distribute power-supply, 2) load seen by the power-supply, 3) parasitic capacitances that inject currents, and 4) internal delays of each of circuits. Propagation delay of a gate is inversely proportional to the noise on  $V_{dd}$  [14], and thus the delay of various cores change and the switching events get distributed in time, changing the peaks in the PSN of that core. At the system level, the  $V_{dd}$  and  $V_{ss}$  wires are connected to the parasitic capacitors of several cores. Parasitic capacitances will contribute to the current and hence the peak value of the current drawn from  $V_{dd}$  will be smaller compared to non-embedded case. To show all above, we have simulated three of ISCAS'85 benchmarks as cores before and after integration. The result is shown in Figure 6. We assumed that a  $5mm$  power-supply wire is used. Note that longer wire, will create slight variation in time it takes for the graph to settle and will increase the peak point. However, the general trend is preserved and our argument remains valid.

As shown in Figure 6, PSN of the SoC (all three cores embedded) is greater than each individual PSN. However, it is clearly not additive. If we were to have designed a system expecting the individual PSN will add it would be gross over estimation.

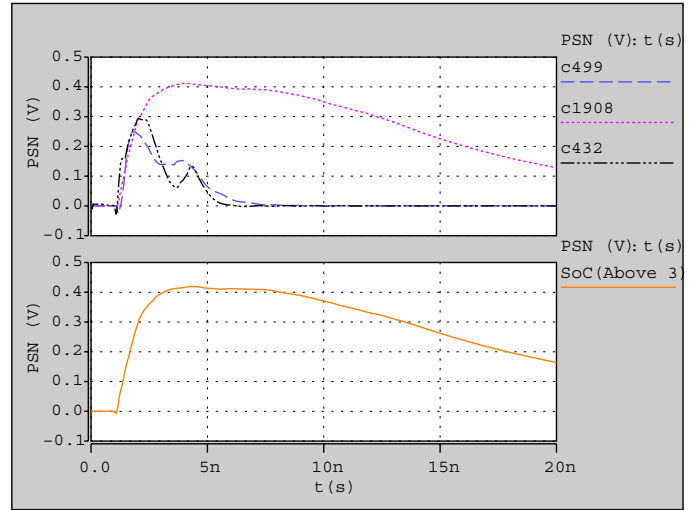


Figure 6: PSN dominance in SoC example.

The best we can say is that:

$$\max_{1 \leq i \leq k} \{PSN_i\} \leq PSN_{SoC}^{MAX} \leq \sum_{i=1}^k PSN_i^{MAX} \quad (3)$$

Obviously, the threshold PSN ( $PSN_{thr}$ ) cannot be defined to be smaller than  $\max_{1 \leq i \leq k} \{PSN_i\}$ .

■ **Effect of Dominant Core:** As shown in Figure 6 the PSN characteristic in SoC can be estimated by  $PSN_{dom} + \Delta PSN$ , where  $PSN_{dom}$  corresponds to the core with the *dominant* PSN. More formally, the dominant core has the maximum statistical PSN average:

$$dom = i^* \ni \frac{1}{T} \int_T PSN_{i^*}(t) \cdot dt = \max_{1 \leq i \leq n} \left\{ \frac{1}{T} \int_T PSN_i(t) \cdot dt \right\} \quad (4)$$

where  $T$  is the PSN evaluation time window. This should not be a surprise as the switching activity in the dominant (the one with the largest transitions or largest  $PSN^{MAX}$ ) circuit does not change significantly. However, the switching current drawn from  $V_{dd}$  by dominant core will change because of additional parasitic capacitors (from other wires and other cores). Hence, its contribution to total noise will only slightly decrease. But other remaining cores whose PSN values are significantly different compared to the dominant core, will have considerable change in propagation delays causing the switching activity to be re-distributed (spread-out) in time.

The propagation delays increase progressively in time. As observed in Figure 7 the nodes switching initially experience less delay since the variation (between the cores) in PSN is less. Due to the distributed switching activity, the contribution of PSN of smaller cores in overall SoC PSN will decrease. Additionally, due to these time variations the PSN of smaller cores might add favorably reducing the total PSN.

■ **Concept & Analysis of Shift Delay:** *Shift delay* is defined as the time between a node switching in a non-embedded environment versus a particular embedded (SoC) environment it will be located in. Without analytical calculation of shift delay of individual gates, we managed to estimate the shift delay

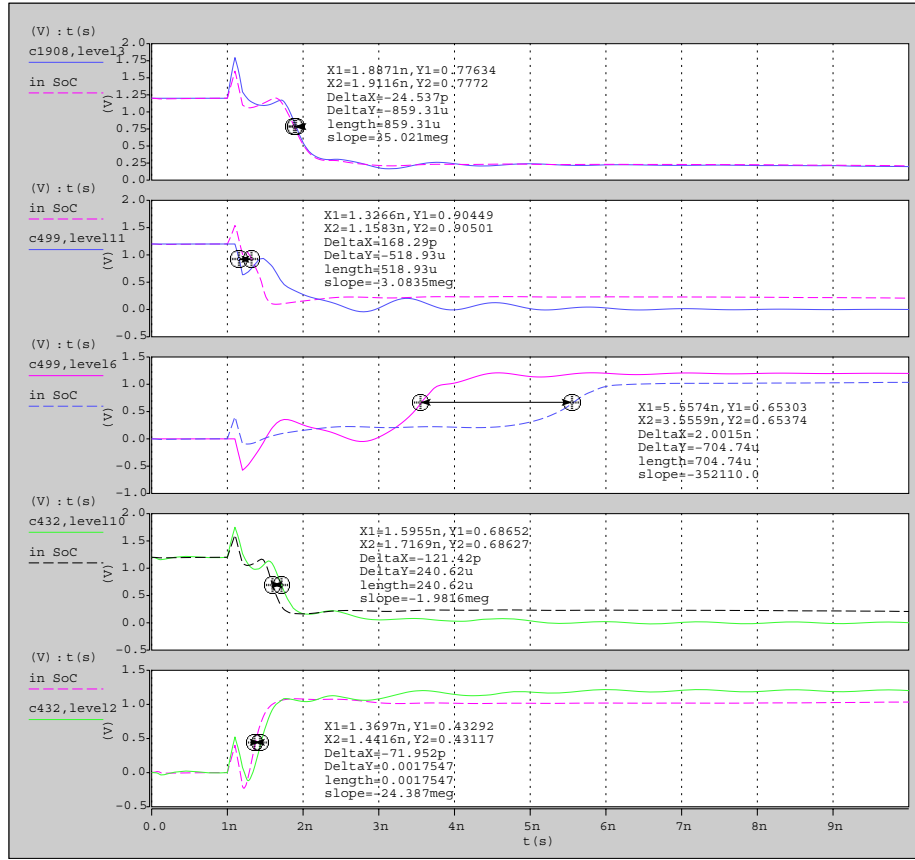


Figure 7: Switching delay in certain nodes in non-embedded and embedded SoC environment.

of each time interval using a probabilistic approach. The time axis is divided into  $M$  time intervals and a probability density function (pdf) is defined in each time interval. The size of intervals is arbitrary. However, the smaller interval (larger  $M$ ) is expected to produce a more accurate PSN estimation with the cost of more computation. This pdf models how shift delay will affect PSN curves.

The increase in shift delay as  $PSN_i(t)$  varies from  $PSN_{SoC}(t)$  is shown in Figure 7. If we choose  $[t_{m,1}, t_{m,2}] = [0, 2]ns$  time interval, we can observe from Figure 6 that the variation between  $PSN_{C432}(t)$  (non-dominant core) and  $PSN_{dom} = PSN_{C1908}(t)$  (dominant core) is less than variation between  $PSN_{C499}(t)$  and  $PSN_{C1908}(t)$ . This explains the increase in shift delay (DeltaX in Figure 7) of C499 gate outputs compared to C432 gate outputs shown in 7. Similarly the variation of  $PSN_{C1908}(t)$  to itself is 0 causing the least delay for its outputs. Note that PSN of dominant is not exactly same as PSN of SoC hence even the dominant one will see a slight variation and shift delay. Additionally, since there is larger PSN variation in time interval  $[2, 4]$  compared to  $[0, 2]$  (clearly seen in Figure 6) we observe more shift delay in  $[2, 4]$  time interval in Figure 7 for C499 (the third curve).

Considering dependency of shift delay and time intervals, we found out that the *lognormal distribution* [22] can closely reflect the relationship. The standard lognormal distribution function with a time shift is:

$$f(x) = \frac{e^{-((\ln x)^2/2\sigma^2)}}{x\sigma\sqrt{2\pi}} \quad x \geq 0, \sigma > 0 \quad (5)$$

where  $\sigma$  is the standard deviation, also called the shape parameter and  $x$  represents the shift in time. The probability density function is shown in Figure 8. Lognormal function is a normal-like behavior in which chance of having too small delays is high while a long delay with small probability may occur. The lognormal function essentially says that the shift delay values less than the mean are quite unlikely and the values greater than mean are likely but tightly bounded.

For the purpose of our application, we use  $f(t - \Delta t)$  to shift the pdf into different time intervals, where

$$SD_{i,m} = f(t - \Delta t) \quad (6)$$

where  $m = [t_{m,1}, t_{m,2}]$  is a time interval in which PSN is estimated,  $f(t)$  is the lognormal behavior (Equation 5) with:

$$\begin{cases} \Delta t_m = (t_{m,1} + t_{m,2})/2 \\ \sigma_m = |\max_m\{PSN_{dom}(t)\} - \max_i\{PSN_i(t)\}| > 0 \end{cases}$$

Intuitively,  $\Delta t_m$  is midpoint of each time interval ( $m$ ) and  $\sigma$  is defined as the magnitude of difference between the maximum of  $PSN_{dom}(t)$  and maximum of  $PSN_i(t)$  in that interval.

■ **Estimating PSN for SoC:** After calculating  $\Delta t$  and  $\sigma$  for each time interval for all cores we first compute PSN for non-dominant embedded cores by adding the weighted sum of its

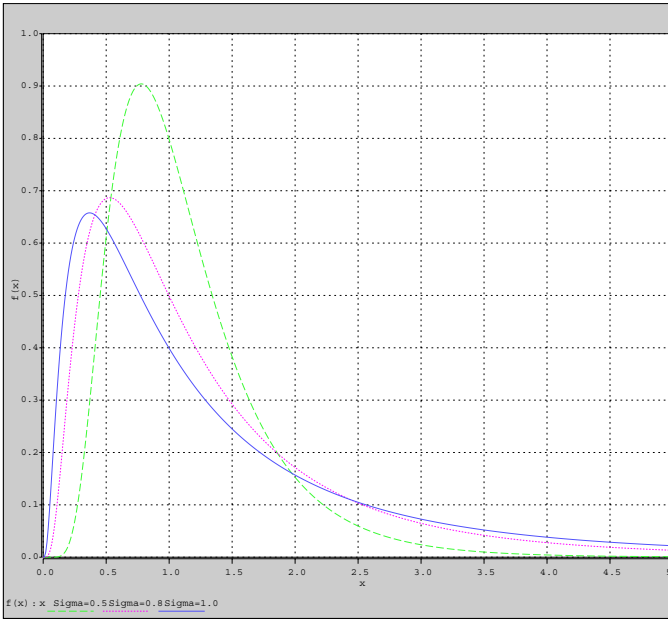


Figure 8: Standard lognormal distribution.

PSN where shift delay is approximated using the lognormal function:

$$PSN_i^{emb}(t) = \sum_{m=1}^M PSN_i(t) \cdot SD_{i,m}(t) \quad (7)$$

Now, the non-normalized PSN is the sum of PSN corresponding to the dominant core and PSN of non-dominant embedded cores:

$$PSN_{SoC}^{Non-Norm}(t) = PSN_{dom}(t) + \sum_{i=1; i \neq dom}^n PSN_i^{emb}(t) \quad (8)$$

Finally, we normalize (average) the effect of the peak PSN values over each interval:

$$PSN_{SoC}(m) = \frac{\int_{t_{m,1}}^{t_{m,2}} PSN_{SoC}^{Non-Norm}(t) dt}{|t_{m,2} - t_{m,1}|} \quad (9)$$

The above computation is repeated for all intervals  $1 \leq m \leq M$  to get  $PSN_{SoC}$  in the time window of interest.

### 4.3 Example

To show the accuracy of our model, we estimated PSN for the example shown in Figure 6 discussed earlier. Figure 9 shows the procedure of calculating  $PSN_{C432}^{emb}$ . The top and middle curves show the lognormal-based shift delay and the PSN for this core, respectively. To find the contribution of this core when embedded into a large system, we use Equations 7, 8 and 9 which essentially blend the top two curves in Figure 9 and produces the bottom curve. This needs to be repeated for every core and the summation of these contributions will form  $PSN_{SoC}$ .

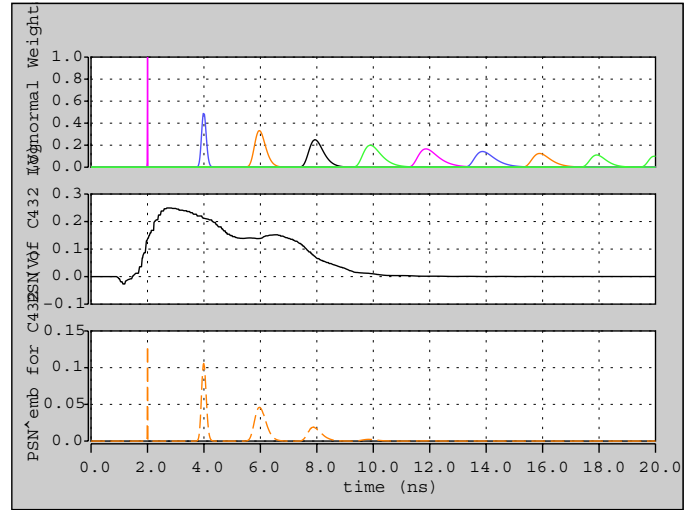


Figure 9: Computing PSN for embedded C432.

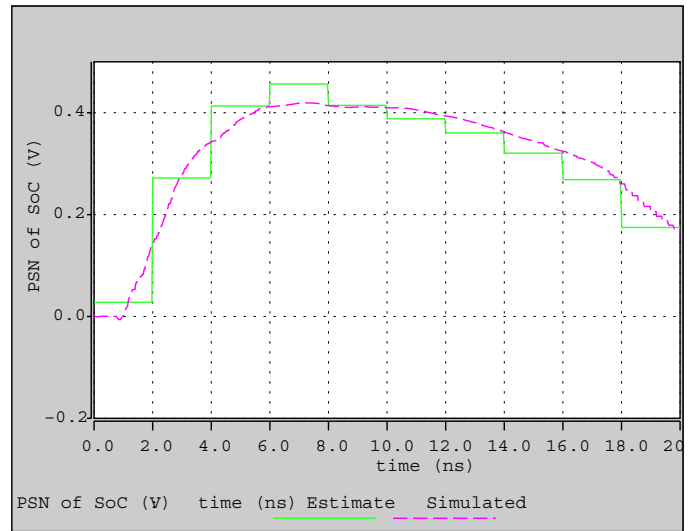


Figure 10: Comparing simulation and estimated of PSN.

Figure 10 compares the approximated PSN (using our analysis based on concept of shift delay) with the simulated one (using HSPICE [21]) and demonstrate high accuracy.

## 5 PSN Control

The PSN analysis for SoCs, explained in the previous section, will be used now in a typical application that targets controlling PSN by using separate power-supply lines/pins. Equations 7, 8 and 9 provide accurate PSN and thus can be employed by any partitioning heuristic to make the decision based on PSN metric. For showing the effect of partitioning in this section we use a straightforward greedy heuristic to perform the partitioning.

### 5.1 PSN-Driven Partitioning Method

Assuming that some floorplan information is available, for each core we have a relatively accurate physical dimension and also

length of power-supply line. Thus, our plan is to do these for  $Core_i$  ( $i = 1, \dots, k$ ):

1. Estimate (or ask floorplanner to provide) the length of power-supply lines feeding  $Core_i$ .
2. Run our ATPG to find  $PSN_i$  and  $PSN_i^{MAX}$ .
3. Use Equations 7, 8 and 9 to group cores in partitions for which a separate power-supply pins/lines are used.

The basic idea is to find minimum number of independent power-supply pins/lines to reduce PSN to an acceptable (threshold) limit. Recall from Section 4 that  $PSN_{thr}$  cannot be set smaller than  $\max_{1 \leq i \leq k} \{PSN_i\}$ . Note that having multiple power-supply pins/lines does not mean having multiple power-supply values. Even when the entire SoC uses only one power-supply value (e.g.  $V_{dd} = 1.2v$ ) still using multiple power-supply lines/pins will be beneficial as the goal of partitioning is to control the maximum level of PSN in all lines. Extending this approach when multiple power-supply values are used is straightforward as we know a priori which core uses which power value. Details are beyond the scope of this paper.

Our partitioning algorithm first uses the floorplan information (estimate of length of power-supply line assuming that core is alone) to analyze worst case and obtain PSN curves (lines 1-2). Then, the procedure essentially follows the basic *graph coloring* heuristic which is a greedy algorithm that combines two curves at a time. The cores are sorted based on their non-embedded (stand-alone) behavior (line 3). An inner while loop (lines 9-20), chooses one core at a time and adds its curve to the accumulated curves up to that point. At every step if the result (according to Equations 7, 8 and 9) shows an acceptable PSN behavior (lines 12-17) then it proceeds and adds it to that partition (line 14). Otherwise, it leaves it out (line 19). The outer while loop makes sure that all of the cores find a home partition. The pseudocode for this algorithm is shown in Figure 11. The running time is  $O(k^2)$ , where  $k$  is total number of cores in SoC.

## 6 Simulation and Results

Several ISCAS'85 benchmarks were implemented in  $0.13\mu m$  technology. A distributed RLC (Figure 1) equivalent network was used to model the power-supply network. RLC values used for the pin are  $R_p = 0.3\Omega$ ,  $L_p = 8nH$  and  $C_p = 4pF$ . and RLC components used for the wire  $R = 0.04\Omega/\mu m$ ,  $L = 10pH/\mu m$  and  $C = 10aF/\mu m$ . Two wire lengths  $10mm$  (first 3 circuits) and  $12mm$  were assumed for the power-supply. All the simulations were performed using  $0.13\mu m$  technology. Table 3 shows the peak noise values generated by the input vectors produced by our algorithm. Simulation results for the maximum noise among M1 to M4 are shown to provide a comparison for M5 and M6.

The last two columns compare the run time of our ATPG and HSPICE [21]. Note carefully that the run time of our ATPG reflects the time needed to analyze a gate-level circuit and getting information from TetraMax to construct the pattern pairs. On the other hand, the run time of HSPICE is reported *per pattern*

```

01: for ( $i = 1, \dots, k$ )
02:   Use our ATPG to analyze PSN for  $Core_i$  and find  $PSN_i$ .
03:   Sort  $PSN_i$  in a list  $L$  in ascending order of  $PSN_i^{MAX}$ .
04:    $j = 1$ 
05:    $P_j = \{\}$  ;  $PSN_{P_j} = 0$ 
06:   while ( $L$  is not empty)
07:   {
08:     all elements in  $L$  are unmarked.
09:     while (there is unmarked elements in  $L$ )
10:     {
11:       Choose  $Core_i$  the first unmarked element in  $L$  and
       evaluate PSN in  $P_j \cup \{Core_i\}$  using Equations 7, 8 and 9
12:       if ( $PSN_{P_j}^{MAX} \leq PSN_{thr}$ )
13:       {
14:          $P_j = P_j \cup \{Core_i\}$ 
15:         Update  $PSN_{P_j}$ 
16:         Delete  $Core_i$  from  $L$ 
17:       }
18:     else
19:       Mark  $Core_i$  in  $L$ 
20:   }
21:    $j = j + 1$ 
22: }

```

Figure 11: Partitioning procedure based on PSN.

pair and for the last three benchmarks (marked with \*) it did not converge after a few hours. The PSN results for the last three circuits are obtained using Synopsys PowerMill estimator tool [21]. We conclude that using transistor level simulator (e.g. HSPICE) to analyze PSN and find a test vector pair is not practical for even medium-size circuits. Our ATPG constructs the pattern pair very quickly and its report of the worst case PSN is verified by other tools (i.e. HSPICE for the first six circuits and PowerMill for the last three).

### 6.1 Experiments for PSN Analysis & ATPG

The ATPG based power estimation algorithm provides a less time consuming (compared to exhaustive) and more deterministic (compared to random) approach to estimate peak power before implementation (on chip). This provides designers a tool to detect possible faults due to high power consumption. One of the noteworthy applications would be to assure that the power-supply network can provide enough current without damaging (open circuit). This will also provide an opportunity to redesign the power network to reduce parasitic components (if possible). High fan-in nodes that is drawing significant current can be detected and re implemented as a combination of other gates, or using different logic style.

### 6.2 Experiments for PSN Control

To verify the accuracy of our methodology, we must simulate the SoC, e.g. using HSPICE. Unfortunately, it becomes computationally impractical to do this for large SoCs. Moreover as PSN is a technology dependent factor, we are not also able to compare to others who use different libraries, technologies, etc.

Table 3:  $PSN$  results for benchmarks.

Circuit	# of Gates	$PSN_i^{MAX}$ using our ATPG				HSPICE per pair [sec]
		M1-M4 (Max)	M5	M6	ATPG [sec]	
Figure4	6	0.10	0.25	0.25	1.5	5
C432	160	0.34	0.45	0.40	3.1	47
C499	202	0.10	0.30	0.30	4.0	91
C880	357	0.42	0.42	0.43	3.5	183
C1355	514	0.10	0.30	0.42	5.6	124
C1908	880	0.15	0.48	0.47	5.2	3889
C3540	1667	0.66	0.68	0.67	5.3	*
C5315	2290	0.67	0.69	0.68	5.8	*
C6288	2416	0.67	0.70	0.70	8.3	*

Table 4: Results of applying the partitioning algorithm for Example 2.

$PSN_{thr}$ [V]	# of Partitions Found	$PSN_{SoC}^{MAX}$ [V]-HSPICE
0.45	1	0.44
0.42	2	0.41
0.35	3	0.34

Therefore, we decided to show the concept and the correctness of our partitioning using two SoCs made from a few ISCAS benchmarks as cores. The HSPICE simulation for these two examples are performed assuming a  $5mm$  power-supply line to reduce the simulation complexity.

■ **SoC Example 1:** We have put together and analyzed three cores i.e.  $Core_1$ ,  $Core_2$  and  $Core_3$  are C432, C499 and C1908 from ISCAS'85 benchmarks, respectively. Our ATPG reported maximum PSN for these three cores as:  $PSN_1^{MAX} = 0.30V$ ,  $PSN_2^{MAX} = 0.25V$  and  $PSN_3^{MAX} = 0.41V$  (see the first curve in Figure 6). To have a good tight scenario to check the quality of our partitioning algorithm we asked partitioning algorithm to determine number of power-supply lines to have  $PSN_{thr} = 0.5V$ . The analysis method is fairly fast and accurate. It finds out that  $PSN_{SoC}^{MAX} \leq 0.45V$  (see the estimated curve in Figure 10) and stays below the threshold limit. The algorithm thus suggested having only one partition and one power-supply network. The HSPICE simulation, confirmed this (see second curve in Figure 6).

■ **SoC Example 2:** In this example, we used C880 benchmarks. The PSN analysis shows that for non-embedded C880 we have  $PSN_{C880}^{MAX} \approx 0.33V$ . To construct our SoC example here we replicated C880 benchmarks three times and then ran the partitioning algorithm with three different threshold values. The results are tabulated in Table 4.

To verify the decision, we have simulated all possible partitions (one C880, two C880s and three C880s) separately using HSPICE. The maximum value is reported in the last column and stays below  $V_{thr}$  as expected. The complete PSN behaviors are shown in Figure 12. The rationale for decision made by the par-

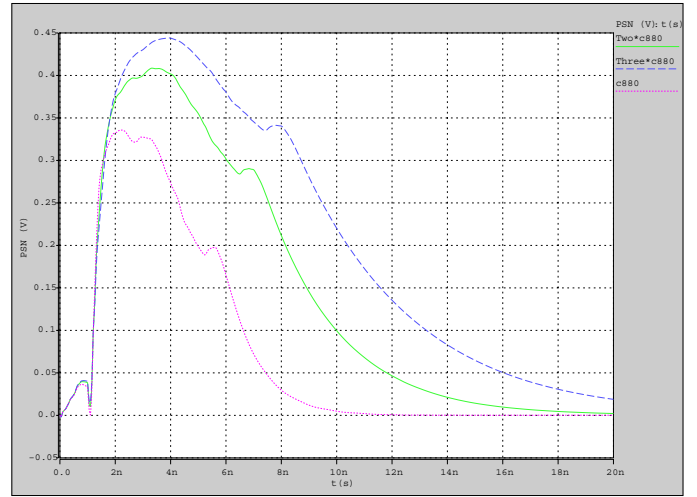


Figure 12: PSN for 3-core SoC (C880 is replicated three times).

tioning algorithm is now clear. The PSN analysis conveyed to the partitioning algorithm an accurate estimate of PSN and thus it could make the right decision in all above cases. For the first case ( $PSN_{thr} = 0.45V$ ) there is no need for additional power-supply line. For the last case ( $PSN_{thr} = 0.35V$ ), three partitions (three separate  $V_{dd}/V_{ss}$  pins) are chosen to control power supply noise. For case of  $PSN_{thr} = 0.42V$ , two partitions (partition 1 with one C880 and partition 2 with two C880s) are selected by the algorithm.

### 6.3 Limitations and Future Work

Our work in this paper was limited to combinational circuits. We intend to extend our method to sequential circuits using the same strategy. First, conventional algorithms based on stuck-at fault model (e.g. time-frame expansion) is used to find pattern pairs for each node. Then, the PSN analysis algorithm combines the patterns to stimulate maximum noise while taking into account the limited controllability of some nodes (i.e. feedback lines).

At present our PSN analysis algorithm looks for patterns that induce the worst case (maximum) noise based on circuit topology. This is done regardless of functional or testing modes in which the circuit may operate. Functional limitation (e.g. instruction execution that may be constrained) and behavior in test environment (e.g. excessive switchings during scan operation) may impose new situations that affect the worst case PSN analysis. In future, we intend to incorporate such scenarios by customizing the pattern generator scheme. This would involve considering a set of criteria (pre-defined based on limitations or specifications in functional or testing modes) by which the pattern generator chooses patterns from the pool of candidates.

## 7 Conclusion

Accurate power-supply noise analysis is a fairly complicated task. Fast and accurate analysis and pattern generation for non-embedded cores is the first challenge. The second challenge

is that we cannot simply add the PSN of non-embedded cores to estimate the PSN of a SoC. The work presented in this paper offers a remedy for both of these challenges. Our ATPG gets some aid from a fault simulator and constructs the pattern pair that stimulates the maximum PSN for non-embedded cores. The PSN behavior of non-embedded cores will be combined with a lognormal shift delay behavior to predict the PSN behavior for an embedded core. Using this model, we can reasonably predict the overall PSN in a SoC. Such information about the worst case scenario can be used in many ways, e.g. redesign according to variations not seen in design phase and decide the best way of distributing power across a chip while simultaneously keeping the noise under a desired limit.

## Acknowledgements

This work was supported in part by the National Science Foundation CAREER Award #CCR-0130513.

## References

- [1] R. Senthinatharr and J. L. Prince, *Simultaneous Switching Noise of CMOS Devices and Systems*, Kluwer Academic Publishers, 1994.
- [2] H. Chen and S. Neely, "Interconnect and Circuit Modeling Techniques for Full-Chip Power Supply Noise Analysis," *IEEE Transactions in Components, Packaging and Manufacturing Technology*, vol. 21, no. 3, pp. 209-215, Aug. 1998.
- [3] Y. Jiang, K. Cheng and A. Deng, "Estimation of Maximum Power Supply Noise for Deep Sub-Micron Designs," in Proc. of *Low Power Electronics and Design Symposium*, pp. 233-238, Aug. 1998.
- [4] C. Metra, L. Schiano and M. Favalli, "Concurrent Detection of Power Supply Noise," in *IEEE Transactions on Reliability*, vol. 52, no. 4, pp. 469-475, Dec. 2003.
- [5] Y. Chang, S. Gupta and M. Breuer, "Analysis of Ground Bounce in Deep Sub-Micron Circuits," in Proc. *VLSI Test Symp. (VTS'97)*, pp. 110-116, 1997.
- [6] H. Chen and D. Ling, "Power Supply Noise Analysis Methodology for Deep-Submicron VLSI Design," in Proc. *Design Automation Conf. (DAC'97)*, pp. 638-643, 1997.
- [7] L. Zheng, B. Li, and H. Tenhunen, "Efficient and Accurate Modeling of Power Supply Noise on Distributed On-Chip Power Networks," in Proc. *Int. Symposium on Circuits and Systems (ISCAS'00)*, pp. 513-516, 2000.
- [8] J. Lee, Y. Huh, P. Bendix, S. Kang, "Understanding and Addressing the Noise Induced by Electrostatic Discharge in Multiple Power Supply Systems," in Proc. *Int. Conf. on Computer Design (ICCD'01)*, pp. 406-411, Sept. 2001.
- [9] Y. Jiang, K. Cheng and A. Krstic, "Estimation of Maximum Power and Instantaneous Current Using a Genetic Algorithm," in Proc. *Custom Integrated Circuits Conf. (CICC'97)*, pp. 135-138, 1997.
- [10] G. Bai, S. Bobba and I. Haji, "Maximum Power Supply Noise Estimation in VLSI Circuits Using Multimodal Genetic Algorithms," in Proc. *Int. Conf. on Electronics, Circuits and Systems (ICECS'01)*, vol. 3, pp. 1437-1440, 2001.
- [11] S. Zhao, K. Roy and C. Koh, "Estimation of Inductive and Resistive Switching Noise on Power-supply Network in Deep Sub-micron CMOS Circuits," in Proc. *Int. Conf. on Computer Design (ICCD'00)*, pp. 65-72, 2000.
- [12] Y. Jiang and K. Cheng, "Vector Generation for Power Supply Noise Estimation and Verification of Deep Submicron Designs," in *IEEE Transaction on VLSI*, vol. 9, issue 2, pp. 329-340, April 2001.
- [13] A. Krstic, J. Yi-Min and C. Kwang-Ting, "Pattern Generation for Delay Testing and Dynamic Timing Analysis Considering Power-Supply Noise Effects," in *IEEE Transaction on CAD*, vol. 20, issue 3, pp. 416-425, March 2001.
- [14] J. Vazquez and J. Gyvez, "Power Supply Noise Monitor for Signal Integrity Faults," in Proc. of *Design, Automation and Test in Europe Conference*, pp. 1406-1407, Feb. 2004.
- [15] H. Chen, "A Hierarchical Power Supply Distribution Model for Full-Chip Switching Noise Analysis," *IEEE Meeting on Electrical Performance of Electronic Packaging*, pp. 60-63, 1997.
- [16] L. Cao and J. Krusius, "A New Power Distribution Strategy for Area Array Bonded ICs and Packages of Future Deep Sub-Micron ULSI," in Proc. of *Electronic Components and Technology Conference*, pp. 1138-1145, June 1997.
- [17] N. Pham, M. Cases, D. Araujo and E. Matoglu, "Design Methodology for Multiple Domain Power Distribution Systems," in Proc. of *Electronic Components and Technology Conference*, pp. 542-549, June 2004.
- [18] S. Zhao, K. Roy and C. Koh, "Power Supply Noise Aware Floorplanning and Decoupling Capacitance Placement," in Proc. of *ASP-DAC Conference*, pp. 489-495, Jan. 2002.
- [19] J. Rabaey, M. Pedram, *Low Power Design Methodologies*, Kluwer Academic Publishers, 1994.
- [20] M. Nourani, M. Tehranipoor and N. Ahmed, "Pattern Generation and Estimation for Power-Supply Noise Analysis," in *VLSI Test Symposium (VTS'05)*, pp. 439-444, May 2005.
- [21] Synopsys Inc, *User Manual for TetraMax, PowerMill, HSPICE*, 2003.
- [22] R. Ziemer, *Elements of Engineering Probability and Statistics*, Prentice-Hall, 1996.