On the Improvement of Statistical Timing Analysis

Rajesh Garg rajeshgarg_at_tamu.edu
Nikhil Jayakumar nikhil_at_ece.tamu.edu
Sunil P Khatri sunilkhatri_at_tamu.edu
Department of Electrical & Computer Engineering,
Texas A&M University, College Station TX 77843.

***Abstract*— As the minimum feature sizes of VLSI fabrication processes continue to shrink, the impact of process variations is becoming increasingly significant. This has prompted research into extending traditional static timing analysis so that it can be performed statistically. However, statistical static timing analysis (SSTA) tends to be quite pessimistic. In this paper we present a sensitizable statistical timing analysis (StatSense) technique to overcome the pessimism of SSTA. Our StatSense approach implicitly eliminates false paths, and also uses different delay distributions for different input transitions for any gate. These features enable our StatSense approach to perform less conservative timing analysis than the SSTA approach. Our results show that on average, the worst case ($\mu + 3\sigma$) circuit delay reported by StatSense is about 20% lower than that reported by SSTA.**

I. Introduction

In recent times, statistical timing analysis has received significant attention in both academe and industry. This has been primarily due to the fact that process variation control has not kept pace with the rapidly diminishing feature sizes. While a lot of research has suggested that statistical timing analysis is essential for timing closure in VLSI design today, the use of this new method of timing analysis has not been readily welcomed by all chip designers. It is not just the reticence of designers towards adopting a new design methodology that is preventing/slowing the adoption of this new timing approach. There is also a legitimate concern that the results of statistical timing analysis tend to be overly pessimistic. Besides, statistical timing analysis takes longer to run. It also requires a greater effort during the gate library characterization phase. Designers are hence skeptical about the benefits of this new timing analysis methodology.

The are many sources of pessimism in statistical timing analysis and many of them are dependent on the method used for the analysis. Some of the common sources are:

1) Spatial correlations
2) Path correlations
3) Approximation of PDFs (Probability Density Functions) to Gaussian distributions (usually done during calculation of MAX of two PDFs)
4) False paths
5) The assumption that gate delays are Normally distributed

In this paper we deal with the last two sources of pessimism. The approach discussed in this paper also implicitly considers path correlations and does not approximate PDFs to Gaussian distributions. In particular, each input transition at a gate (which results in an output change) is assumed to have a Normal distribution. Since there may be several such input transitions that cause some output transition, the resulting delay at the output consists of several Normal distributions (one for every input transition that causes an output change).

II. Previous Work

Most techniques for statistical timing analysis are essentially based on the principles of Static Timing Analysis (STA). Hence statistical timing analysis is often called statistical static timing analysis (SSTA). The fundamental operations in a SSTA tool are the SUM and the MAX operations. Most SSTA algorithms rely on smart ways to implement these SUM and MAX operations for delay distributions, rather than use a single discrete delay value.

In [1], the authors use PCA (Principal Component Analysis) to handle spatial correlations. They assume all delay distributions to be Gaussian and approximate the MAX of 2 or more Gaussian distributions to be Gaussian as well. In [2], a canonical first-order delay model is proposed and an incremental block based timing analyzer is used to propagate arrival times and required times through a timing graph in this canonical form. One of the major contributions of the algorithm proposed in [2] is that it allows the statistical timing engine to be used incrementally. In [3], [4], [5], the authors note that accurate statistical timing analysis can become exponential. Hence, they propose faster algorithms that compute bounds on the exact result rather than the exact result itself. In [6], the authors propose representing the arrival times as CDFs (Cumulative Distribution functions) and the gate delays as PDFs to help perform the SUM and MAX operations efficiently. In [7], the authors propagate delay distributions (PDFs) through a circuit. The PDFs are discretized to help make the operation more efficient.

The common theme in all the above works is that they are based on the static timing analysis framework. Hence only the structurally long paths are identified through these algorithms. The authors of [8] identify this deficiency and come up with a statistical timing analysis flow that considers false paths. While the authors of [8] reduce pessimism by considering false paths, they do not address the pessimism that arises from considering the gate delay distribution to be a single Gaussian.

Our approach eliminates false paths implicitly. In the statistical timing analysis flow discussed in [8], a traditional SSTA is done, followed by an attempt to find sensitizable paths. In our approach, this order is reversed. We first find the primary input vector transitions that result in the sensitizable longest delays for the circuit, and then do a statistical analysis on these

1-4244-9707-X/06/$20.00 ©2006 IEEE

vector transitions. This statistical analysis utilizes, for each gate, the particular Normal distribution which corresponds to the input transition that the gate undergoes for the longest delay to be sensitized.

The main contributions of this paper are two-fold:

- By utilizing a sensitizable timing analysis tool, our approach implicitly eliminates false paths. SSTA does not eliminate false paths, leading to pessimistic results.
- For each input transition on the gate (which causes an output change), our approach utilizes separate Normal distributions, hence the statistical delays reported at the circuit level are more representative of the true circuit behavior. In SSTA, a single normal distribution is used for any gate, regardless of the input transitions that the gate undergoes.

III. Our Approach

Our approach eliminates false paths and also accounts for the fact that the delay of a gate has different Normal distributions for different input transitions (which cause an output transition). Our approach consists of two phases. In the first phase we find a set of logically sensitizable vector transitions that result in the largest delays for the circuit. In the second phase, we use Monte-Carlo based techniques to propagate the arrival times for these delay-critical sensitizable vector transitions, and come up with a delay distribution at the outputs. The input transitions at any gate are known after the first phase, and so the gate delay distribution corresponding to this input transition is utilized in the second phase. The second phase therefore performs SSTA, using the appropriate gate delay distribution corresponding to the input transition for each gate. In the remainder of this section, these two phases are described, along with a discussion on how input arrival times are propagated for any gate.

A. Phase 1: Finding Sensitizable Delay-critical Vector Transitions

To make sure that we don't spend time performing statistical analysis on false paths, we first find a user-specified number of sensitizable vector transitions that result in the largest delays for the circuit. This is done using the *sense* [9] package in SIS [10]. *Sense* uses a SAT solver to verify if a particular delay (initially set to the delay found from a static timing analysis) is sensitizable. If there is no satisfiable input vector that produces this delay, then the delay value is reduced in steps till we reach a delay D that has a satisfying vector (a vector on the primary inputs that has a delay D). In its original implementation, *sense* returns only the critical delay of the circuit. We augmented the *sense* routine to return the vector (final vector) at the primary inputs, as well as all the possible previous vectors at the primary inputs that cause this delay. A change from any previous vector to a final vector is referred to as a *vector transition*. The set of input transitions is stored in an array for use in the second phase of our statistical timing flow. We then insert the complement of this largest sensitizable delay vector as a SAT clause in the *sense's* SAT routine and run *sense* again to get the next critical vector. We continue this till we get a large enough set of delay-critical vector transitions. The number of vector transitions collected before we move on to the second phase of the flow is decided based on desired accuracy and available time for computation. In the second phase of the flow, we propagate arrival times in a manner that exploits the fact that we know the input transition at each gate. This is explained in the following section.

Rising Transition #	ab → ab	Delay(ps)
1	11 → 00	30.5
2	11 → 01	50.5
3	11 → 10	53.0
Falling Transition #	**ab → ab**	**Delay(ps)**
1	00 → 11	55.3
2	01 → 11	46.5
3	10 → 11	42.7

TABLE I

TRANSITIONS FOR A NAND GATE THAT CAUSE ITS OUTPUT TO SWITCH

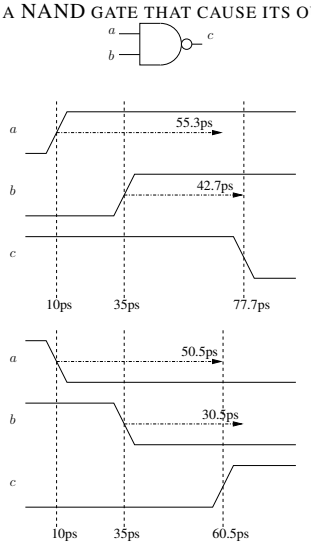

Fig. 1. Example of Timing Analysis using a NAND2 gate

B. Propagating Arrival Times

In a regular static timing analysis, we find the structurally worst case delay. In our timing analysis we take advantage of the fact that we know exactly what transitions cause a node to switch. The details of how we do this is explained with the example of a NAND2 gate. Let us first consider just the nominal delay of a NAND2 gate.

Table I is a list of input transitions that cause the output of the NAND gate to change its logic value. Let AT_i^{fall} denote the arrival time of a falling signal at node i and AT_i^{rise} denote the arrival time of a rising signal at node i.

In the case of a regular STA, the rising time (delay) at the output c of a NAND2 gate is calculated as

$$AT_c^{rise} = MAX[(AT_a^{fall} + MAX(D_{11\rightarrow 00}, D_{11\rightarrow 01})), \\ (AT_b^{fall} + MAX(D_{11\rightarrow 00}, D_{11\rightarrow 10}))]$$

where, $MAX(D_{11\rightarrow 00}, D_{11\rightarrow 01})$ is often referred to as the pin-to-pin rising output delay from the input a, while $MAX(D_{11\rightarrow 00}, D_{11\rightarrow 10})$ is referred to as the pin-to-pin rising output delay from the input b.

Similarly, in STA the falling time (delay) at the output c of a NAND2 gate is given by

$$AT_c^{fall} = MAX[(AT_a^{rise} + MAX(D_{00\rightarrow 11}, D_{01\rightarrow 11})), \\ (AT_b^{rise} + MAX(D_{00\rightarrow 11}, D_{10\rightarrow 11}))]$$

where, $MAX(D_{00\rightarrow 11}, D_{01\rightarrow 11})$ is often referred to as the pin-to-pin falling output delay from the input a, while $MAX(D_{00\rightarrow 11}, D_{10\rightarrow 11})$ is referred to as the pin-to-pin falling output delay from input b.

For example, if the worst case falling or rising arrival time at inputs a and b was 10ps and 35ps respectively, then the rise delay at c would be calculated to be = MAX(10+50.5, 35+53.0) = 88.0ps. Similarly for a falling c output, the delay would be MAX(10+55.3,35+55.3) = 90.3ps. However this is a pessimistic method of calculating the delay. In our approach we attempt to remove some of this pessimism.

Let us first consider the rising output. The output of the NAND2 gate switches high when any of the two inputs switches low. From the output of *sense* we can find the actual vector transition that causes the largest delay for a given circuit. This primary input vector transition induces a transition on the gate inputs. Let us assume that this input transition was $11 \rightarrow 00$ for the NAND2 gate. A naive way of calculating the delay would be to state that the delay would be given by

$$AT_c^{rise} = MAX(AT_a^{fall}, AT_b^{fall}) + D_{11\rightarrow 00}$$

Assuming again that the arrival times at inputs a and b were 10ps and 35ps respectively, the delay would be then be calculated as MAX(10,35)+30.5 = 65.5. However, we do know that the output would start switching before 65.5 since signal a arrives earlier than signal b. As a result, we can say that the gate effectively goes through the transition $11 \rightarrow 01 \rightarrow 00$ rather than $11 \rightarrow 00$ directly. Note that the output of the NAND2 gate falls for the vector 01 as well. Hence, we calculate the delay to be

$$AT_c^{rise} = MIN((AT_a^{fall} + D_{11\rightarrow 01}), (AT_b^{fall} + D_{11\rightarrow 00}))$$

In our example, the delay is hence MIN(10+50.5,35+30.5) = 60.5. Note that we used the minimum of two delays in this case since any one input falling causes the output to switch. Also note that the delay calculated (60.5ps) is much smaller than the worst case delay calculated using regular STA (88.0ps). The reduction in pessimism in our approach occurs due to the fact that we have information about the input transition for the gate.

Now consider the case of the falling output. The output of the NAND2 gate switches low only when both the inputs switch high. Again, we exploit the fact that *sense* provides the actual vector transition that caused the critical delay. Let us assume that the induced input transition for the NAND2 gate was $00 \rightarrow 11$. A naive way of calculating the delay would be to state that the delay is

$$AT_c^{fall} = MAX(AT_a^{rise}, AT_b^{rise}) + D_{00\rightarrow 11}$$

Assuming again that the arrival times at inputs a and b were 10ps and 35ps respectively, the delay would be calculated as MAX(10,35)+55.3 = 90.3. However, we do know that a arrives earlier than b. As a result, we can say that the gate effectively goes through the transition $00 \rightarrow 10 \rightarrow 11$ rather than $00 \rightarrow 11$ directly. Hence, in our approach, we calculate the delay to be

$$AT_c^{fall} = MAX((AT_a^{rise} + D_{00\rightarrow 11}), (AT_b^{rise} + D_{10\rightarrow 11}))$$

In our example, the delay is hence MAX(10+55.3,35+42.7) = 77.7. Note that we used the maximum of two delays in this case since both inputs need to switch to cause the output to switch. Also note that the delay calculated (77.7ps) is smaller than the worst case delay calculated using regular STA (90.3ps).

These results are shown graphically in the Figures 2 and 3. These plots show the arrival time of the output c of a NAND2 gate, for the $00 \rightarrow 11$ and $11 \rightarrow 00$ transitions respectively. The arrival time of one of the inputs a is fixed to zero and the arrival time of the other input b swept between -150ps to 150ps. The propagated delays are shown for STA and our method, along with the delay found by SPICE [11]. As can be seen from these plots, our method of calculating the arrival times for multiple switching inputs matches SPICE quite accurately and is significantly better (less pessimistic) than a traditional STA method for computing arrival times.

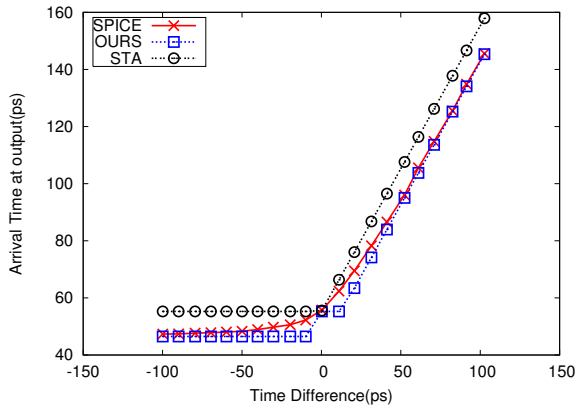

Fig. 2. Plot of arrival times at output of NAND2 gate calculated through various means for the transition $00 \rightarrow 11$

We can similarly derive the equations to calculate the arrival times for any arbitrary gate, depending on the input transitions at that gate. Let us consider a NAND3 gate with inputs $\{a, b, c\}$. Let us first consider the inputs to the NAND3 gate changing as follows:

$$000 \rightarrow 100 \rightarrow 110 \rightarrow 111$$

The output of the NAND3 gate switches low only when the inputs are 111. Hence the delay of the gate would be calculated as follows:

$$AT_{out}^{fall} = MAX[(AT_a^{rise} + D_{000\rightarrow 111}), \quad (1)$$
$$(AT_b^{rise} + D_{100\rightarrow 111}), \quad (2)$$
$$(AT_c^{rise} + D_{110\rightarrow 111})]$$

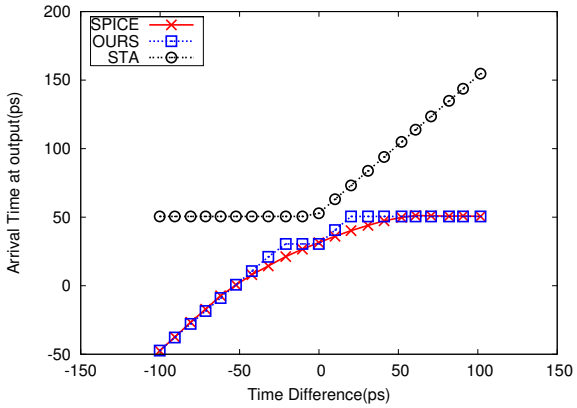

Fig. 3. Plot of arrival times at output of NAND2 gate calculated through various means for the transition $11 \rightarrow 00$

Now let us consider a NAND3 gate with its output rising. Let the inputs change as below

$$111 \rightarrow 011 \rightarrow 001 \rightarrow 000$$

In this case, the output of the NAND3 gate starts switching high when at least one of the inputs is logic 0. Hence the delay of the gate would be calculated as:

$$AT_{out}^{rise} = MIN[(AT_a^{fall} + D_{111\rightarrow011}), \quad (3)$$
$$(AT_b^{fall} + D_{111\rightarrow001}), \quad (4)$$
$$(AT_c^{fall} + D_{111\rightarrow000})]$$

An extension to handling delay distributions is easily done by simply considering the distribution to be made of several distinct delay values, obtained from the PDF of the gate delay.

C. Phase 2: Computing the Output Delay Distribution

In the second phase of the computation, we perform Monte Carlo analysis on the sensitizable vector transitions that result in the largest delays for the circuit (which were computed in the first phase, described in Section III-A). In each of the STA runs for Monte Carlo analysis, we perform arrival time propagation as described in Section III-B. Since the primary input vector transitions may induce transitions on the input of each gate, the delay distribution of the gate for the corresponding gate input transition is used. A random value of the gate delay is computed from this distribution. This is done for each gate in the circuit. Finally, STA is performed, using these delay values. The resulting maximum delay over all the outputs is used to compute the worst case delay distribution of the circuit.

In a NAND2 gate we have 3 different input rising transitions that cause an output falling transition (these are shown in the bottom half of Table I). For any iteration of STA, if we choose the value of delay for one of the 3 transitions (say $00 \rightarrow 11$) to be $\mu_{00\rightarrow11} + n\sigma_{00\rightarrow11}$, we choose the value of the other two transitions ($01 \rightarrow 11$, $10 \rightarrow 11$) to be $\mu_{01\rightarrow11} + n\sigma_{01\rightarrow11}$ and $\mu_{10\rightarrow11} + n\sigma_{10\rightarrow11}$ respectively.

Parameter	Nominal Value	σ
L	0.1μ	0.005μ
V_T^N	0.2607V	0.013V
V_T^P	0.3030V	0.01515V

TABLE II
PARAMETERS WITH THEIR VARIATION

IV. EXPERIMENTAL RESULTS

In order to demonstrate the effectiveness of our technique, we tested our technique on several benchmark circuits from the ISCAS89 and MCNC91 benchmark suite. For all simulations, we assumed a 0.1μm process and used the BPTM 0.1μ process [12] model card for SPICE simulations. Our standard cell library consisted of 8 cells. The 8 cells were INV, INV2X, NAND2, NAND3, NAND4, NOR3, NOR3, NOR4.

We first characterized each of the standard cells in our library to come up with a table of values for the mean and standard deviation of the delay of *each transition* (that cause a change in the output). This pre-characterization was done for a set of different capacitance values. This pre-characterization was done using SPICE. The parameters considered to be varying, along with their variations, are given in Table II. In this table, all parameters are modeled such that their σ is 5% of their μ.

The characterization results for a NAND2 gate (with a load capacitance of 6fF) are shown in Figures 4 and 5. Figure 4 shows the delay histogram for the three vector transitions which result in a rising output. These vectors are $11 \rightarrow 00$, $11 \rightarrow 01$ and $11 \rightarrow 10$. Note that each of these vector transitions exhibit different output delay distributions. Similarly, Figure 5 shows the delay histogram for the three vector transitions which result in a falling output. These vectors are $00 \rightarrow 11$, $01 \rightarrow 11$ and $10 \rightarrow 11$. Note that each of these vector transitions also exhibit different output delay distributions. The mean and standard deviation of all these distributions are computed and used in the second phase of our algorithm.

During the timing analysis phase of our approach, we interpolate between these capacitance values to find the mean and standard deviation of the delay for the given load capacitance value.

Next we carry out the first phase of our flow. We use *sense* to find the top few sensitizable critical delays and their corresponding input vector transitions. The result of the first phase of our approach is a set of vector transitions on the primary inputs of the circuit. In our experiments, we utilize the top 50 (or 25) primary input vector transitions that result in the largest circuit delay.

For the second phase of our approach, we propagate these transitions throughout the circuit. Since we have the knowledge of the input transitions at each gate, we use the arrival time propagation methodology explained in Section III-B to compute the arrival time at the gate output. This step of propagating circuit delays is done 1000 times in our experiments (or as many times as is required to get a reasonably stable and accurate estimate of the mean and standard deviation of

Ckt	SSTA				StatSense 50						StatSense 25					
	μ (ps)	σ (ps)	$\mu + 3\sigma$	Time	μ (ps)	σ (ps)	$\mu + 3\sigma$	Ratio	Time	Ratio	μ (ps)	σ (ps)	$\mu + 3\sigma$	Ratio	Time	Ratio
alu2	1008.39	19.08	1065.63	278.8	661.25	17.69	714.32	0.67	1991.5	7.14	668.64	17.27	720.45	0.68	1232.5	4.42
alu4	1234.77	18.21	1289.4	560.2	753.01	23.74	824.23	0.64	3386.8	6.04	767.45	16.52	817.01	0.63	3217.9	5.74
apex6	680.51	10.95	713.36	632.2	447.66	26.36	526.74	0.74	895.0	1.41	460.44	29.87	550.05	0.77	453.1	0.716
apex7	489.79	8.16	514.27	207.5	427.17	12.89	465.84	0.90	260.6	1.25	430.79	12.94	469.61	0.91	129.5	0.62
C499	737.00	11.29	770.87	419.4	617.92	14.55	661.57	0.86	481.6	1.15	617.79	14.65	661.74	0.86	238.8	0.57
C1355	714.82	8.59	740.59	484.8	418.08	11.47	452.49	0.61	578.1	1.2	418.06	11.62	452.92	0.61	291.8	0.60
cordic	669.99	8.60	695.79	657.0	578.18	18.5	633.68	0.91	657.23	1.00	587.74	12.55	625.40	0.90	355.3	0.54
i6	496.16	22.80	564.56	353.0	449.55	19.84	508.52	0.90	609.5	1.73	449.63	19.88	509.27	0.90	293.79	0.83
i7	496.25	21.76	561.53	514.3	449.31	20.60	511.11	0.91	494.9	0.96	449.39	20.65	511.34	0.91	366.6	0.65
rot	781.23	13.75	822.48	571.0	501.65	17.24	552.72	0.67	1343.6	2.35	501.87	17.26	552.78	0.67	810.4	1.42
x1	319.34	10.40	350.54	261.5	269.43	13.70	310.10	0.88	277.14	1.06	277.62	13.37	317.73	0.91	141.6	0.54
AVG								0.79		2.29				0.80		1.51

TABLE III

COMPARISON OF SSTA AND STATSENSE

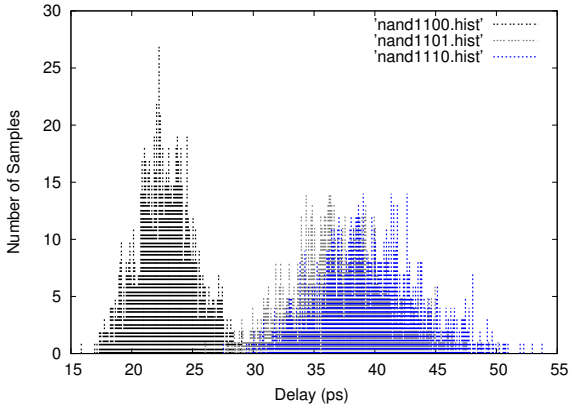

Fig. 4. Characterization of NAND2 Delay for all Input Transitions which Cause a Rising Output

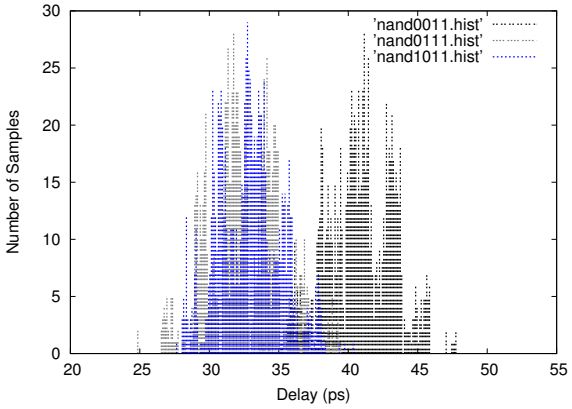

Fig. 5. Characterization of NAND2 Delay for all Input Transitions which Cause a Falling Output

the maximum delay of the circuit). For each of these 1000 iterations, a random value of delay is chosen for each gate. This random value is chosen from a Gaussian distribution with a μ and σ derived from the precharacterized table of values for each gate. Note that the μ and σ used for any gate correspond to the vector transitions that appear at that gate, for the primary input vector transition being simulated. We assume that the variations of process parameters within a gate are correlated (i.e. the threshold voltages and channel lengths of all the devices within the gate vary in the same manner). To enforce this assumption, we must choose the random delay value carefully. For example, in a NAND2 gate we have 3 different input rising transitions that cause an output falling transition (these transitions are shown in the bottom half of Table I). For any iteration of the timing analysis, if we choose the value of delay for one of the 3 transitions (say $00 \rightarrow 11$) to be $\mu_{00\rightarrow 11} + n\sigma_{00\rightarrow 11}$, we must choose the value of the other two transitions ($01 \rightarrow 11$, $10 \rightarrow 11$) to be $\mu_{01\rightarrow 11} + n\sigma_{01\rightarrow 11}$ and $\mu_{10\rightarrow 11} + n\sigma_{10\rightarrow 11}$ as well.

Table III describes the results of experiments conducted to compare StatSense with SSTA. Our major contribution in this work is to make statistical timing analysis more realistic. Hence, we compare our methodology with Monte-Carlo based SSTA. It is well known that block based SSTA sacrifices accuracy for speed due to approximations when propagating PDFs (especially when computing the MAX of 2 or more delay distributions). The SSTA experiments in this table were conducted using 10000 iterations. The StatSense iterations were computed using 1000 iterations per input vector transition. In this table, Column 1 lists the circuit under consideration. Columns 2 through 4 list the μ, σ and $\mu+3\sigma$ delays(in ps) returned by SSTA. Column 5 lists the SSTA runtime. All runtimes in this table are in seconds. Columns 6 through 11 list the results for StatSense, when 50 input vector transitions (which result in the largest sensitizable circuit delay) were simulated. Columns 6 through 8 list the μ, σ and $\mu + 3\sigma$ delays(in ps) returned by StatSense. Column 9 reports the ratio of the $\mu + 3\sigma$ value returned by StatSense, compared to that returned by SSTA. *Note that StatSense, on average, returns a much lower worst case circuit delay (the $\mu + 3\sigma$ delay) than SSTA. This illustrates the pessimism of SSTA, and validates our claim that StatSense reduces this pessimism.* Column 10 and 11 respectively list the runtime for StatSense and the ratio of this runtime with the runtime of SSTA. On average, note that StatSense (run with 50 input vector transitions requires about $2.3\times$ more runtime than SSTA.

Columns 12 through 17 have the same information as Columns 6 through 11, except that the StatSense simulations for these columns were performed using 25 input vector transitions (which result in the largest sensitizable circuit delay). The purpose of this experiment was to verify if the StatSense runtime can be reduced by simulating fewer input vector transitions. By comparing Columns 8 and 14, we note that there is no appreciable loss of fidelity when 25 input vector transitions are used, instead of 50. The worst case circuit delay (the $\mu + 3\sigma$ delay), averaged over all designs, is almost identical in both cases. The benefit of using 25 input vector transitions is indicated in Column 17, which shows that on average, StatSense (with 25 input vector transitions) requires only about 50% more runtime that SSTA. For most of the circuits used in our experiments, the difference between the nominal delays for the first and fiftieth vector chosen was large. In cases, where this delay difference is small, a larger number of vectors will need to be used.

In spite of the fact that SSTA conducts 10000 STA iterations, and StatSense conducts 50000 (or 25000) iterations, the runtime of StatSense is not $5\times$ (or $2.5\times$) that of SSTA. This is because StatSense performs an event driven delay simulation. Whenever there is no transition at the output of a gate g, delay computations for gates in the fanout of g may be avoided. This pruning is not possible in SSTA.

Figure 6 illustrates the delay histogram obtained by SSTA (with 50000 STA iterations) along with the delay histogram obtained by StatSense (with 50 input vector transitions simulated). These histograms were obtained for the *apex7* example. For each input vector transition in StatSense, 1000 iterations of delay computation are performed. This figure shows how the pessimism of SSTA is alleviated by StatSense.

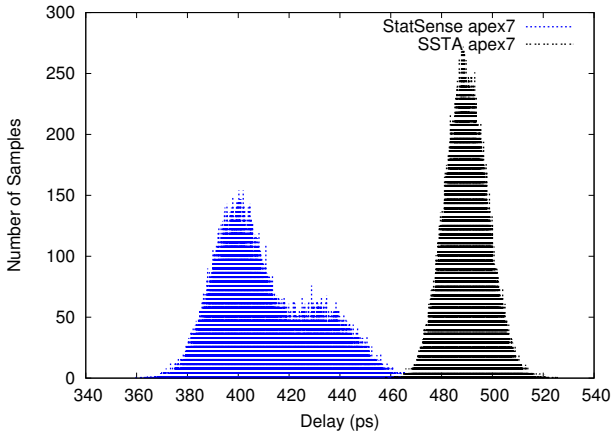

Fig. 6. Delay Histograms for SSTA and StatSense (for $apex7$)

V. Future Work and Conclusions

In response to the growing impact of process variations, there has been much research in extending traditional static timing analysis so that it can be performed statistically. The resulting statistical static timing analysis (SSTA) approaches are, however, quite pessimistic. This pessimism arises from the fact that most static timing analysis tools and their statistical counterparts do not consider false paths. The second major source of pessimism is that most statistical timing analyzers assume delay distributions at all gates in a design to be Gaussian. However, the delay distribution of a gate is not necessarily Gaussian. In fact the delay distribution for a multi-input gate is Gaussian for *each input vector transition* that causes a change on the gate output. In this paper we present a sensitizable statistical timing analysis (which we call StatSense) technique to overcome the pessimism of SSTA. Our StatSense approach implicitly eliminates false paths, and also uses different delay distributions for different input transitions for any gate. These features enable our StatSense approach to perform less conservative timing analysis than the SSTA approach. Our results show that on average, the worst case $(\mu + 3\sigma)$ circuit delay reported by StatSense is about 20% lower than that reported by SSTA. In the future, we plan to work on techniques to reduce the runtime of the statistical timer. This would allow us to use more input vector transitions for the statistical analysis. We also plan to investigate methods to find out the minimum number of vector transitions required to get a realistic statistical timing result.

References

[1] H. Chang and S. S. Sapatnekar, "Statistical timing analysis under spatial correlations," vol. 24, pp. 1467–1482, Sept. 2005.

[2] C. V. C, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in *DAC*, pp. 331–336, 2004.

[3] A. Agarwal, V. Zolotov, and D. T. Blaauw, "Statistical timing analysis using bounds and selective enumeration," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, pp. 1243–1260, Sept 2003.

[4] A. Agarwal, D. Blaauw, and V. Zolotov, "Statistical timing analysis for intra-die process variations with spatial correlations," in *ICCAD*, pp. 900–907, 2003.

[5] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula, "Statistical timing analysis using bounds," in *DATE*, pp. 62–67, 2003.

[6] A. Devgan and C. V. Kashyap, "Block-based static timing analysis with uncertainty.," in *ICCAD*, pp. 607–614, IEEE Computer Society / ACM, 2003.

[7] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation.," in *DAC*, pp. 661–666, ACM, 2001.

[8] J.-J. Liou, A. Krstic, L.-C. Wang, and K.-T. Cheng, "False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation.," in *DAC*, pp. 566–569, ACM, 2002.

[9] P. McGeer, A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli, *Logic Synthesis and Optimization*, ch. Delay Models and Exact Timing Analysis, pp. 167–189. Kluwer Academic Publishers, 1993.

[10] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[11] L. Nagel, "Spice: A computer program to simulate computer circuits," in *University of California, Berkeley UCB/ERL Memo M520*, May 1995.

[12] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," in *Proc. of IEEE Custom Integrated Circuit Conference*, pp. 201–204, Jun 2000. http://www-device.eecs.berkeley.edu/ ptm.