Static Statistical Timing Analysis for Latch-based Pipeline Designs

Mango C.-T. Chao[*], Li-C. Wang[*], Kwang-Ting Cheng[*], Sandip Kundu[**]

[*]Department of ECE, UC-Santa Barbara
[**] Intel Corporation, Austin, Texas

Abstract

A latch-based timing analyzer is an essential tool for developing high-speed pipeline designs. As process variations increasingly influence the timing characteristics of DSM designs, a timing analyzer capable of handling process-induced timing variations for latch-based pipeline designs becomes in demand. In this work, we present a static statistical timing analyzer, STAP, for latch-based pipeline designs. Our analyzer propagates statistical worst-case delays as well as critical probabilities across the pipeline stages. We present an efficient method to handle correlations due to reconvergent fanouts. We also demonstrate the impact of not including the analysis of reconvergent fanouts in latch-based pipeline designs. Comparing to a Monte-Carlo based timing analyzer, our experiments show that STAP can accurately evaluate the critical probability that a design violates the timing constraints under a given statistical timing model. The runtime comparison further demonstrates the efficiency of our STAP.

1. Introduction

For designs with level-sensitive latches, a signal may have a delay larger than the clock period and may flush through the latches without causing incorrect data propagation, whereas the delay of a signal in designs with edge-triggered flip-flops must be smaller than the clock period to ensure the correctness of data propagation across flip-flop stages. Due to the characteristics of this clock stealing, using level-sensitive latches is a popular methodology for high-performance pipeline designs. Many researchers, [1, 2, 3], have conducted research in timing analysis for latch-based designs in the early 90's. However, prior works are based on fixed-delay timing models that may no longer be effective to represent the timing behavior of today's deep sub-micron (DSM) designs.

With today's manufacturing technology, the timing of a single device can be significantly influenced by process variations, resulting in timing variations from chip to chip. Therefore, using a statistical timing model in timing analysis becomes a natural extension to the traditional fixed-delay analysis. However, once we come to use a statistical timing model, timing analysis becomes a much more complicated problem than that in fixed-timing domain.

Among many research works for statistical timing analysis (*STA*), the Monte-Carlo based approaches [4, 5] are considered to be most accurate. However, it can be very time-consuming for a Monte-Carlo approach to obtain a convergent result for large designs. Block-based approaches for STA were proposed in [6, 7, 8, 9, 10] as more efficient alternatives to Monte-Carlo based STA. These approaches aim to quickly obtain approximate results, and the accuracy of a block-based STA is often compared to the results from a Monte-Carlo STA.

In [6], the STA approach propagates the worst-case output delay of two discrete random variables based on an enumeration method. It could not handle reconvergent fanouts effectively. The authors in [7] propose a method to compute the upper bound and lower bound of circuit worst-case delay and yet, the approach also relies on enumeration to handle reconvergent fanouts. The approach in [8] utilizes a piece-wise linear model for representing cumulative density functions and can compute the worst-case output delay efficiently. The authors in [8] also propose a heuristic for handling reconvergent fanouts, but not spatial correlations among delay random variables. The works in [9, 10] apply the method in [11] to compute the worst-case output delay by assuming that delay random variables are all Gaussian. Both works propose an approach for handling spatial correlations.

The timing analysis for latch-based pipeline designs with level-sensitive latches is different from the timing analysis for combinational designs or for flip-flop-based design because of the following two characteristics. First, for level-sensitive latches, a delay in one pipeline stage depends on the delays in the previous pipeline stage. Second, on a pipeline design, we need to propagate not only the longest/shortest delays from a primary input (PI) to a primary output (PO) through the pipeline stages, but also the *critical probabilities* that the delays on latches violate setup-time/hold-time constraints. This high dependency across the pipeline stages exacerbates the impact of correlations among delay random variables, especially the correlations resulting from reconvergent fanouts. Section 2.1 and 2.2 will detail these two properties in our timing analysis.

In this paper, we propose a static statistical timing analyzer, named *STAP*, to evaluate the probability that a given latch-based pipeline design violates the timing constraints under process-induced timing variations. In STAP, we utilize the method in [11] to propagate the worst-case delay random variables. The method can effectively consider correlations among random variables. We also propose a separate method for propagating the critical probabilities across pipeline stages. Moreover, we present a technique for efficient manipulation of the correlation data during the delay propagation analysis. We compare STAP with a Monte-Carlo based approach sampling 10000 circuit instances. The experimental results demonstrate the accuracy and the efficiency of STAP in terms of critical probability propagation and handling reconvergent fanouts.

2. Problem Formulation

2.1. Timing model for level-sensitive latches

In a pipeline design with level-sensitive latches, a delay on a pipeline stage may affect the delay on the next stage in the next clock cycle. Figure 1 illustrates this property. In Figure 1, a signal propa-

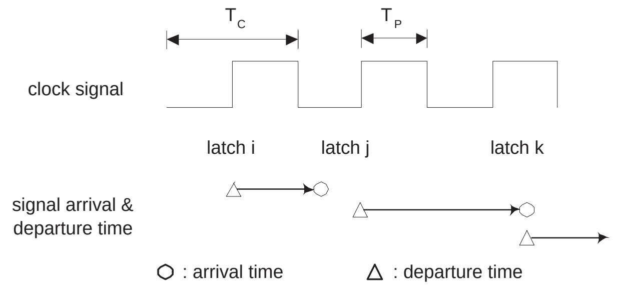

Figure 1: Relation between departure time and arrival time for a level-sensitive latch.

gates from latch i to latch j and then to latch k, and all three latches use the same clock phase. We use triangles and circles to represent the signal departure time and arrival time, respectively. T_C and T_{P_i} represent the clock period and the width of the active interval of latch phase P_i, respectively. If the signal arrives at a latch before the latch becomes active, the departure of the signal for the next latch has to wait until the current latch become active, e.g., the signal from latch i to latch j in Figure 1. If the signal arrives at a latch during the active interval, the signal can directly pass through the latch without any waiting, e.g., the signal from latch j to latch k in Figure 1.

The relations among arrival times, departure times, T_C, T_{P_i}, and the worst-case delays follow the following SMO model [1]:

$$
\begin{aligned}
D_i &= max(A_i, T_C - T_{P_i}) && (1)\\
d_i &= max(a_i, T_C - T_{P_i}) && (2)\\
A_i &= max(D_j + \Delta_{ji} - E_{P_jP_i}) && (3)\\
a_i &= min(d_j + \delta_{ji} - E_{P_jP_i}) && (4)\\
A_i &\leq T_C - T_{P_i} - S_i && (5)\\
a_i &\geq H_i && (6)
\end{aligned}
$$

where D_i (d_i) represents the latest (earliest) signal departure time, A_i (a_i) represents the latest (earliest) signal arrival time, Δ_{ji} (δ_{ji}) represents the maximum (minimum) delay from latch i to latch j, $E_{P_jP_i}$ represents the *forward phase shift* from phase P_i to phase P_j, and S_i (H_i) represents the setup (hold) time of latch i.

Due to the above latch-based timing constraints, in a latch-based pipeline design, the worst-case delay on one pipeline stage depend on its departure time in the previous stage. This property prevents us from applying a traditional timing analysis approach for combinational designs or for flip-flop-based designs on latch-based pipeline designs. A traditional timing analysis tool treats flip-flops as pseudo primary inputs (PPIs) and pseudo primary outputs (PPOs). The analysis is only for finding out the worst-case delays between flip-flops. It works fine because the departure time of an edge-triggered flip-flop is always the same during each clock cycle. For example, in Figure 1, the departure time for the edge-triggered flip-flop would be always like that case depicted on latch j. For the signal on latch k in Figure 1, the flip-flop would have latched the wrong data. Therefore, for latch-based pipeline designs, we need to compute the worst-case delay by propagating delays across multiple latch stages.

In the remaining of this paper, we primarily discuss the analysis by considering the late-mode timing constraints, that is, D_i, A_i, and setup time constraints. The early-mode timing analysis can be achieved with a similar approach.

2.2. Critical probabilities for latches

There are two events that a latch i may capture the wrong data. The first event is that the worst-case arrival time A_i violates the setup time constraint in equation (5) due to an overly long signal delay in the current pipeline stage. We define this event as $CECS_i$, the critical event (failure) induced by the current pipeline stage. Then we can define $CPCS(i)$ as the probability that $CECS_i$ occurs.

$$
\begin{aligned}
CPCS(i) &= Prob\{CECS_i\}\\
&= Prob\{A_i \geq T_C - T_{P_i} - S_i\} && (7)
\end{aligned}
$$

The other event when the latch i may store the wrong data is when any latch j in the previous stage stores the wrong data and then propagates the wrong data to latch i. We denote this event as $CEPS_i$, the critical event induced by the previous pipeline stage. With the $CECS_i$ and $CEPS_i$, we can define the critical event of a latch i as ICE_i, the combined critical event that $CECS_i$, $CEPS_i$, or both occur. Then we define $CPPS(i)$ and $ICP(i)$ as the probabilities that $CEPS_i$ occurs and ICE_i occurs, respectively.

$$
\begin{aligned}
ICP(i) &= Prob\{ICE_i\}\\
&= Prob\{CECS_i \cup CEPS_i\}, && (8)\\
CPPS(i) &= Prob\{CEPS_i\}\\
&= Prob\{\bigcup_{j \sim i} ICE_j\}, && (9)
\end{aligned}
$$

where $j \sim i$ means there is a path from latch j to latch i.

We further define the circuit critical probability, CCP, of a pipeline design as

$$
CCP = Prob\{\bigcup_{j \in PO} ICE_j\}, \qquad (10)
$$

The output of our STAP reports the circuit critical probability CCP, and the $ICP(i)$ for each latch i based on a given statistical delay library with the parameters for each clock phase (T_{P_i}, $E_{P_jP_i}$) and the parameters for each latch (S_i, H_i).

Therefore, in our STAP, we need not only to handle the worst-case delay propagation, but also to handle the critical probability propagation by an ”OR” operation over critical events. This is different from the tradition statistical timing analysis which only considers the worst-case delay propagation.

In Section 3, we will discuss how to propagate worst-case delays and critical probabilities in our timing analysis tool.

2.3. Input/Output of STAP

The inputs of STAP are:

- a target pipeline circuit under analysis,
- statistical delay libraries,
- T_{P_i}, $E_{P_jP_i}$, S_i, H_i for each phase and latch, and
- the clock period, T_C.

The outputs of STAP are:

- the circuit critical probability CCP, and
- $ICP(i)$ for each latch i.

3. Statistical Timing Analysis Approach

3.1. Propagating worst-case delay

In our STAP, we build the timing graph by converting gates and interconnects into nodes and edges. Each delay random variable of a gate or an interconnect is modeled as a Gaussian distribution. For two cascaded random variables, we compute its output delay random variable by convolution. For two random variables converging at a node, we compute its resulting output delay random variable by a maximum operation (in late-mode analysis) or minimum operation (in early-mode analysis).

We denote a random variable X by (μ_x, σ_x^2), where μ_x and σ_x^2 are the mean and variance of X. Given two random variables, $X(\mu_x, \sigma_x^2)$, $Y(\mu_y, \sigma_y^2)$, and their correlation coefficient ρ_{xy}, the convolution $Z(\mu_z, \sigma_z^2) = X + Y$ can be computed by $\mu_z = \mu_x + \mu_y$ and $\sigma_z^2 = \sigma_x^2 + \sigma_y^2$. We assume that the output distribution Z is still Gaussian. The correlation coefficient between the output random variable Z and any other random variable W can be computed by the equation,

$$\rho_{zw} = \rho_{xw} + \rho_{yw} \tag{11}$$

The above computation of convolution of two random variables can be found in any probability textbook.

As for computing the maximum, $Z = Max(X,Y)$, we follow the methods proposed in [11]:

$$\begin{aligned}
\varphi(x) &= (2\pi)^{-1} exp(-x^2/2), && (12)\\
\Phi(x) &= \int_{-\infty}^{x} \varphi(t)dt, && (13)\\
a^2 &= \sigma_x^2 + \sigma_y^2 - 2\sigma_x\sigma_y\rho_{xy}, && (14)\\
\alpha &= (\mu_x - \mu_y)/a. && (15)
\end{aligned}$$

The mean μ_z, variance σ_z^2, and correlation coefficient ρ_{zw} between $Z = Max(X,Y)$ and any other random variable can be obtained by

$$\begin{aligned}
\mu_z &= \mu_x\Phi(\alpha) + \mu_y\Phi(-\alpha) + a\varphi(\alpha), && (16)\\
\sigma_z^2 &= (\mu_x^2 + \sigma_x^2)\Phi(\alpha) + (\mu_y^2 + \sigma_y^2)\Phi(-\alpha) \\
&\quad + (\mu_x + \mu_y)a\varphi(\alpha) - \mu_z^2, && (17)\\
\rho_{zw} &= [\rho_{xw}\sigma_x\Phi(\alpha) + \rho_{yw}\sigma_y\Phi(-\alpha)]/\sigma_z. && (18)
\end{aligned}$$

By a similar derivation as that in [11], we can obtain the results for the minimum operation $Z = Min(X,Y)$ as:

$$\begin{aligned}
\mu_z &= \mu_x\Phi(-\alpha) + \mu_y\Phi(\alpha) - a\varphi(\alpha), && (19)\\
\sigma_z^2 &= (\mu_x^2 + \sigma_x^2)\Phi(-\alpha) + (\mu_y^2 + \sigma_y^2)\Phi(\alpha) \\
&\quad - (\mu_x + \mu_y)a\varphi(\alpha) - \mu_z^2, && (20)\\
\rho_{zw} &= [\rho_{xw}\sigma_x\Phi(-\alpha) + \rho_{yw}\sigma_y\Phi(\alpha)]/\sigma_z. && (21)
\end{aligned}$$

The output distribution of a maximum (or minimum) operation from two Gaussian distributions is not a Gaussian distribution but very close to. To express this quasi-Guessian distribution as a Gaussian distribution, we can keep propagating the delay by above convolution, maximum and minimum operations. However, it is where we may lose accuracy.

The goal of the above worst-case delay propagation is to obtain the latest arrival time on each latch i (A_i in equation (3) and a_i in equation (4)). Instead of computing the worst-case delay between every two connected latches in two separate pipeline stages (Δ_{ji}), we directly compute the A_i by applying the above propagation operations from the inputs of one pipeline stage to its outputs based on the topological ordering, similar to a block-based timing analysis approach. Once a PPO i is reached, we can compute the departure time D_i of the latch by equation (1) for the next pipeline stage. We continue to propagate the worst-case delays from stage to stage until we reach all POs.

3.2. Propagating critical probabilities

For a random variable, the probability that this random variable is larger than a fixed number can be easily calculated by its accumulated density function. So after A_i is obtained by the above delay propagation, we can immediately obtain its $CPCS(i)$ by computing the probability that $A_i > c_i$, where c_i is the value defined in the setup time constraint (equation (5)). This property of quickly computing critical probability for a random variable allows us to represent the $CPCS(i)$ as simply a Gaussian random variable n and a constant c. Hence, we denote a critical probability by a tuple (n,c) in the following discussion.

As for computing $CPPS(i)$ and $ICP(i)$, we have to be able to do the "OR" operation based on two critical probabilities (n_1,c_1) and (n_2,c_2). This is to compute the probability that either "$n_1 > c_1$" or "$n_2 > c_2$" would occur. The output critical probability is denoted as (n_{out}, c_{out}). We first consider the case that $c_1 = c_2 = c$. In this case, we have

$$\begin{aligned}
Prob\{n_1 > c \cup n_2 > c\} &= 1 - Prob\{Max(n_1,n_2) \le c\} \\
&= Prob\{Max(n_1,n_2) > c\} && (22)
\end{aligned}$$

So in this case, $n_{out} = Max(n_1,n_2)$, $c_{out} = c$.

For the general case with different c_1 and c_2, we just shift the mean of n_2 to fit the case in equation (22):

$$Prob\{n_1 > c_1 \cup n_2 > c_2\} = Prob\{Max(n_1, n_2 + c_1 - c_2) > c_1\} \tag{23}$$

In equation (9), if there are more than two $CEPS$s, then we can decompose the "OR" operation of all $CEPS$s into several consecutive "OR" operations of two $CEPS$s.

For any latch i in the first pipeline stage, its $ICP(i)$ is directly equal to $CPCS(i)$ because each $ICP(j)$ is zero when j is a PI. So each $ICP(i)$ in the first pipeline stage can be represented by a tuple (A_i,c_i). By continually propagating all tuples to the latches in the next pipeline stage, we can compute the $ICP(i)$ for all POs in the last stage.

One important reason that we propagate a critical probability as a tuple (n,c) instead of as a constant value is that critical events may have correlations to one another, even for those critical events not in the same pipeline stages. If every critical event is independent, then we can simply calculate the critical probability by multiplications and subtractions over constant values. However, this simple approach would not be accurate. The accuracy of our STAP relies on an effective method for handling correlations among random variables. This is why we selected the technique in [11] for implementing STAP.

3.3. Handling correlation

The most important advantage of [11]'s method for propagating worst-case delays is its ability to handle correlations between two random variables. With equations (11), (18), and (21), the correlation coefficient between the output random variable and any other random variable can be updated. Hence, the correlations can be preserved for the subsequent delay propagation.

The correlations among delay random variables may come from the spatial correlations or from reconvergent fanouts. The spatial correlations among random variables are determined by the circuit layout and the manufacturing process. They can be specified in a covariance matrix in advance. The authors in [9] use the principle component analysis to extract a set of independent random variables to represent the correlated random variables, and can relieve the burden of holding a huge covariance matrix. The authors in [10] also propose a canonical form of the delay model to represent the spatial correlations.

However, for the correlations from reconvergent fanouts, the entire covariance matrix cannot be built until the timing analyzer starts propagating delays. So once we generate a new output random variable, we need to update its correlation coefficients with all other random variables. This will result in large memory consumption and require the timing analyzer to dynamically expand the covariance matrix.

In STAP, we use a *correlation list* for each random variable to record the non-zero correlation coefficient with other variables. After every convoluation or maximum operation, the output random variable Z will inherit the correlation coefficients from its input random variables X and Y, and then update its correlation coefficients with equation (11) or equation (18). For each correlated random variable W in the correlation list of Z, we add the same correlation coefficient into the correlation list of W.

Two rules in STAP are applied to minimize the size of the correlation list. First, after updating the correlation coefficients with equation (11) or with equation (18), we remove the correlation coefficients smaller than a given threshold (this threshold is set as 0.001 in our experiments). Especially after the maximum operation, some correlation coefficients from one input random variable may all become very small due to the factor $\Phi(\alpha)$ or $\Phi(-\alpha)$ in equation (18). Second, for a random variable W in the correlation list, if no random variable will be produced from W by any convolution or maximum operation, then we remove W.

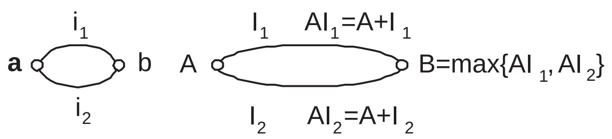

Figure 2: Example for reconvergent fanout.

Figure 2 shows a rather simple example to illustrate how to handle correlations for a reconvergent fanout. Node a and b represent two gates, and i_1 and i_2 represent two interconnects from a to b. A represents the output delay distribution of gate A. I_1 and I_2 represent the delay random variables of i_1 and i_2. Initially, only the random variable itself is in the correlation list of the random variable. After we compute $AI_1 = A + I_1$, AI_1, A and I_1 are in the correlation list of AI_1. By the second rule, I_1 will be remove since no variable will derive from I_1 anymore. AI_1 is then added into the correlation list of A. Next, after we compute $AI_2 = A + I_2$, AI_2, AI_1, A, and I_2 are in the correlation list of AI_2. Then I_2 and A can both be removed according to the second rule. When we compute $B = max\{AI_1, AI_2\}$, we can find out the correlation coefficient from their correlation lists. The final correlation list of B will only contains B itself.

4. Experimental Results

In our experiment, all statistical delay parameters were obtained through pre-characterization of cell libraries using a Monte-Carlo-based SPICE simulator (ELDO) [12] based on a 0.25μm, 2.5V CMOS technology. The pipeline circuits in our experiment are generated by modifying ISCAS benchmark circuits into designs with four pipeline stages based on each circuit's topology. In these designs, we try to balance the maximum number of gates for all paths between two stages and across different stages. All experiments use a single phase clock. We did not consider spatial correlations in these experiments. However, we note that STAP is capable of considering spatial correlations by using the method in [9] or [10].

clock(ps)	3200	3150	3000	2950	2900	2850	2800
MC_STA	0.69	2.42	6.87	21.94	50.10	75.71	94.77
STAP	0.15	1.30	6.77	22.37	48.93	75.99	92.50
difference	-0.54	-1.12	-0.10	0.43	-1.17	0.28	-2.27

Table 1: *CCP* (%) comparisons between MC_STA and STAP on 4-staged benchmark circuit c6288

To validate the accuracy of our STAP, we also implement a Monte-Carlo based timing analyzer, *MC_STA*. In all experiments, the results from *MC_STA* are based on 10000 sampling iterations. Both tools calculate the circuit critical probability *CCP*.

Table 1 shows the *CCP* comparison between our STAP and MC_STA on a 4-stage pipeline circuit modified from c6288 by giving different clock periods. The first row in Table 1 shows the given clock periods. The second and third rows show the *CCP* calculated by MC_STA and STAP, respectively, for each corresponding clock period. The last row shows the *CCP* difference between STAP and MC_STA. As the results of the table show, our STAP can match the result of MC_STA with a difference from -2.27% to 0.42% over different clock periods.

circuit	clock	MC_STA		STAP		Comparison	
	(ps)	CCP (%)	runtime (s)	CCP (%)	runtime (s)	CCP dif. (%)	speedup (X)
c880	640	31.53	25.2	38.04	0.2	6.51	126.6
c1355	650	3.64	27.4	4.71	0.4	1.07	68.4
c6288	2850	75.71	113.4	75.99	1.3	0.28	87.2
s1488	850	83.05	38.5	82.64	0.6	-0.41	64.1
s5378	820	28.71	167.6	28.86	0.9	0.15	186.2
s9234	1380	43.45	283.5	48.50	1.6	5.05	177.2
s38417	1070	10.94	1213.6	8.91	15.3	-2.03	79.3
Avg.						2.21	112.7

Table 2: *CCP* (%) and runtime comparisons between MC_STA and STAP on 4-staged pipeline circuits built from ISCAS benchmark.

Table 2 compares *CCP* and runtimes between STAP and MC_STA on more benchmark circuits. For each benchmark, we report its *CCP* and runtime based on one selected clock period. Column 1 and 2 list the benchmarks and its given clock period. Column 3 and 4 list the *CCP* and runtimes for MC_STA. Column 5 and 6 list the *CCP* and runtimes for STAP. Column 7 lists the *CCP* difference between

STAP and MC_STA. Column 8 calculate the speedup factor obtained from STAP over MC_STA.

These experiments demonstrate the accuracy and efficiency of STAP, which can achieve a 112.7 X speedup over MC_STA with only a 2.21% average accuracy loss in *CCP*. The longest runtime is on the pipeline design modified from s38417. For this design, it only took 15.3 seconds. This indicates the capability of STAP for analyzing larger pipeline circuits.

		MC_STA	STA_No_Cor		STA_Stage_Indep	
circuit	clock(ps)	CCP(%)	CCP(%)	CPP dif.(%)	CCP(%)	CCP dif.(%)
c880	640	31.53	38.03	6.50	38.04	6.51
c1355	650	3.64	9.51	5.87	4.69	1.05
c6288	2850	75.71	100.00	24.29	77.82	2.11
s1488	850	83.05	0.00	-83.05	0.00	-83.05
s5378	820	28.71	41.33	12.62	33.77	5.06
s9234	1380	43.45	63.51	20.06	48.50	5.05
s38417	1070	10.94	100.00	61.24	100.00	61.24
avg.				30.52		23.44

Table 3: *CCP* (%) comparisons between MC_STA, STA_No_Cor, STA_Stage_Indep on 4-staged pipeline circuits built from ISCAS benchmark.

In Table 3, we try to observe the effect of considering correlations caused by reconvergent fanouts. In STA_No_Cor, the effects of recovergent fanouts are ignored. The analysis assumes that every delay random variable is independent. In STA_Stage_Indep, only the effects of reconvergent fanouts within one pipeline stage are considered. The correlations due to reconvergent fanouts across two or more pipeline stages are ignored.

Table 3 compares the *CCP* difference between STA_No_Cor and MC_STA in Column 5, where the average difference is 30.52%. The *CPP* difference between STA_Stage_Indep and MC_STA is listed in Column 7, where the average difference is 23.44%. Both STA_No_Cor and STA_Stage_Indep have much larger *CPP* differences than STAP. More importantly, in some cases, STA_No_Cor and STA_Stage_Indep may report similar results and yet, the results can totally contradict the *CPP* reported by MC_STA. For example, in the cases of s1488 and s38417, STA_No_Cor and STA_Stage_Indep agree on their results, but the results are far from those computed by MC_STA. From these experiments, we see the importance of handling reconvergent fanouts in the analysis for pipeline designs.

circuit	c880	c1355	c6288	s1488	s5378	s9234	s38417	avg.
clock	640	650	2850	850	820	1380	1070	
w/o reduce	0.40	1.5	32.50	1.7	4.6	8.1	86.4	
w reduce	0.2	0.4	1.3	0.6	0.9	1.6	15.3	
speedup(X)	2.0	3.8	25.0	2.8	5.1	5.1	5.6	7.1

Table 4: Runtime comparison of STAP with correlation list reduction and without reduction (Section 3.3)

In the last experiment, we try to show the efficiency and effectiveness of the way we maintain the data in the correlation lists (section 3.3). In table 4 we compare the STAP performance using the reduction rules described in section 3.3 and the STAP performance without using those rules. Table 4 shows that on average, we obtain a 7.1X runtime speedup by employing the reduction rules. The *CPP*s obtained with or without those rules are almost the same.

5. Conclusion

This paper presents STAP, a static statistical timing analyzer for latch-based pipeline designs. By performing convolution, maximum, minimum, and "OR" operations on random variables, STAP propagates worst-case delays as well as critical probabilities from one pipeline stage to another. More importantly, STAP can efficiently handle correlations caused by reconvergent fanouts using dynamically-maintained correlation lists associated with delay random variables. By comparing the accuracy and efficiency of STAP to a Monte-Carlo based timing analyzer, we demonstrate the feasibility and superiority of STAP through experiments on various benchmark circuits.

References

[1] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun, Check Tc and min Tc: Timing verification and optimal clocking of synchronous digital circuits, *ACM/IEEE International Conference on Computer Aided Design*, pp.552-555, November 1990.

[2] T. M. Burks, K. A. Sakallah, and T. N. Mudge, Identification of Critical Paths in Circuits with Level-Sensitive Latches, *ACM/IEEE International Conference on Computer Aided Design*, pp.137-141, November 1992.

[3] J. Lee, D. T. Tang, and C. K. Wong, A Timing Analysis Algorithm for Circuits with Level-Sensitive Latches, *ACM/IEEE International Conference on Computer Aided Design*, pp.535-543, November 1994.

[4] J.-J. Liou, K.-T. Cheng, and D. Mukherjee, Path Selection For Delay Testing of Deep Sub-micron Devices Using Statistical Performance Sensitivity Analysis, *IEEE VLSI Test Symposium*, pp. 97-104, April 2000.

[5] J.-J. Liou, A. Krstic, Y.-M. Jiang, and K.-T. Cheng, Path Selection and Pattern Generation for Dynamic Timing Analysis considering power supply noise effects, *ACM/IEEE International Conference on Computer Aided Design*, pp. 493-496, Nov 2000.

[6] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic, Fast Statistical Timing Analysisby by Probabilistic Event Propagation, *ACM/IEEE Design Automation Conference*, pp. 661-666, June 2001.

[7] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula, Statistical Timing Analysis using Bounds, *ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 62-67, March 2003.

[8] A. Devgan, and C. Kashyap, Block-based Static Timing Analysis with Uncertainty, *ACM/IEEE International Conference on Computer Aided Design*, pp.607-614, November 2003.

[9] H. Chang and S. S. Sapatnekar, Statistical Timing Analysis Considering Spatial Correlations Using A Single PERT-like Traversal, *ACM/IEEE International Conference on Computer Aided Design*, pp.621-625, November 2003.

[10] C. Visweswariah, K. Ravindran and K. Kalafala, First-Order Parameterized Block-Based Statistical Timing Analysis, *ACM/IEEE workshop on timing issures in the specification and synthesis of digital systems*, pp.17-24, February 2004.

[11] C. E. Clark, The Greatest of a Finite Set of Random Variables, *Operation Research*, vol.9, pp.85-91, 1961.

[12] Eldo v4.4.x User's Manual. 1996.