PAPER UNDER REVIEW, PLS DO NOT DISTRIBUTE
Retiming for High Performance FPGAs Considering Flip-flop Constraints and Process Variations

Yu Hu, Yan Lin and Lei He
Electrical Engineering Department
University of California, Los Angeles

ABSTRACT

Retiming is effective to improve performance of sequential circuits. The existing FPGA retiming techniques implicitly assume that an LUT can only drive a flip-flop (FF) within the same basic logic block (BLE). Leveraging all FFs within a same cluster of BLEs, we propose an efficient FF constraint driven retiming algorithm (CDR), and further extend to statistical retiming ($sCDR$) for process variations. Experiments show that CDR improves performance by 5.54% on average (up to 11.6%) compared to the BLE-based retiming. In addition, sCDR reduces up to 6.93% of the mean delay and 17.49% of delay deviation with 10x more runtime compared to CDR. To justify the runtime overhead of sCDR, we propose an effective way to perform sCDR only when it is beneficial. To the best of our knowledge, this is the first in-depth study on the FF constraint driven retiming problem considering process variations for FPGAs.

1. INTRODUCTION

Retiming [1] has been studied extensively to optimize performance in sequential circuits. For ASIC designs, [2] employed retiming in the global placement stage to minimize the clock period by binary search, and [3] presented optimal retiming algorithms without binary search. The available FF slots are pre-fabricated and their locations are fixed for FPGAs. Therefore, FF constraints should be considered explicitly for FPGA retiming. Such constraints are not considered in [2] and [3], however.

There are the following two types of FF constraints in FPGAs. (1) *BLE-based FF Constraint*, where BLE is basic logic element and a lookup table (LUT) can drive a flip-flop (FF) within the same BLE. (2) *CLB-based FF Constraint*, where CLB is a cluster of BLEs and an LUT can drive any FF within the same CLB. [4] and [5] performed FPGA retiming considering BLE-based FF constraint in incremental placement. [6, 23] developed FPGA retiming considering both FF and signal integrity constraints. However, CLB-based FF constraint, which offers more design flexibility, has not been considered in these papers. The first contribution of this paper is to *develop an efficient CLB-based FF constraint driven retiming algorithm*, namely CDR. CDR reduces critical path delay by 5.54% on average (up to 11.16%) compared to retiming with BLE based constraints.

Recently, process variations have drawn growing attention because of their significant impact on delay and power. [7, 8] conducted preliminary studies on the statistical retiming problem for ASIC designs. However, there is no existing statistical retiming applicable to FPGAs with FF constraints. The second contribution of this paper is to *extend our CDR algorithm to statistical CDR (sCDR) to consider process variations*. Experiments show that, compared to CDR, sCDR reduces the mean value and standard deviation of the critical path delay by up to 6.93% and 17.4%, respectively, with 10x more runtime. Furthermore, we reveal the correlation between the potential gain from sCDR and topology of the timing network, which enables us to perform sCDR only when it is beneficial.

The rest of the paper is organized as follows. Section 2 presents FF constraint driven retiming (CDR), and Section 3 extends CDR to consider process variations. Section 4 concludes the paper. To the best of our knowledge, this is the first in-depth study of retiming algorithm considering both FF constraints and process variations for FPGAs.

2. FF CONSTRAINTS DRIVEN RETIMING

2.1 FF Constraints for Modern FPGAs

Our study assumes island style FPGA architecture [9]. Each CLB contains K BLEs, and each BLE contains an LUT and an FF. In order to retime an FPGA design in the post-layout stage, we need to consider placement and FF binding constraints. In older FPGA devices, a BLE has only one output. If an LUT drives the FF within the same BLE, the combinational output of LUT is NOT allowed to drive other FFs since it cannot be seen by other FFs. This is called BLE-based constraint and it is assumed by the existing FPGA retiming work [4] and [5].

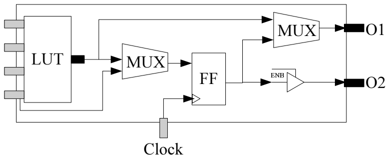

Figure 1: Modeling for the BLE in modern FPGAs

On the other hand, in modern devices such as Virtex 4 [10] and Stratix II [11], an empty FF can be driven by any LUT within the same CLB through local routing. Figure 1 shows the simplified model of the 4-input BLE structure in modern FPGAs. The two outputs correspond to the normal output and sequential output, respectively. The sequential output is enabled when I_4 is occupied to feed FF directly, which enables the independent access of LUT and FF within one BLE. This feature introduces more freedom for FF placement after retiming a circuit. However, if an LUT is allowed to drive an FF outside of its CLB, the overhead introduced by placement disturbance and extra global interconnect delay[1], may negate

[1] In this paper, the local interconnects are those within one CLB, and global interconnects are those between CLBs.

the gain of retiming. To reduce the overhead of the post-retiming FF placement, we introduce CLB-based FF constraint where an LUT can drive all FFs within its CLB, but not FFs outside its CLB.

2.2 Problem Formulation

Retiming [1] minimizes the critical path delay and clock period of a circuit by inserting and deleting FFs, but without affecting the circuit functionality. For retiming, a directed cyclic graph, $\mathcal{G} = (V, E)$, is often constructed to model the circuit for sequential timing analysis. In this graph, vertices represent the inputs/outputs of basic circuit elements such as LUTs and FFs. Edges are added between the inputs of combinational logic elements (e.g. LUTs) and their outputs, and between the connected pins specified by the circuit netlist. Each edge is annotated with the delay $d(e)$ of the circuit element or routing, and the weight $w(e)$ that is the number of FFs inserted in it. Formally, retiming is defined as follows [1].

DEFINITION 1. *Given the retiming graph $\mathcal{G} = (V, E)$, a retiming is an integer-valued vertex-labeling $r : V \rightarrow \mathcal{Z}$.*

The edge weights, w', after retiming are expressed as

$$w'(u,v) = w(u,v) + r(v) - r(u),\ \forall (u,v) \in E \qquad (1)$$

Following [3], let T denote a given clock period. The sequential arrival time, $a(v)$, of node v is

$$a(v) = \max(0, \max_{\forall (u,v) \in E}(a(u) + d(u,v) - w'(u,v) \cdot T)), \forall v \in V$$

A valid retiming solution must satisfy that the edge weights (1) are non-negative and sequential arrival time is less or equal to the target clock period T.

Given an FPGA architecture with size-K CLB and a target clock period T, the retiming constraints are summarized as follows.

$$\sum_{\forall e(u,v) \in C} w'(u,v) \leq K, \qquad \forall C \in \mathcal{CLB} \qquad (2)$$

$$r(u) = r(v), \quad \forall e(u,v) \in E_{con} \quad (3)$$

$$w'(u,v) = w(u,v) + r(v) - r(u) \geq 0, \qquad \forall e(u,v) \in E \quad (4)$$

$$a(v) \leq T, \qquad \forall v \in V \qquad (5)$$

where $\mathcal{CLB}$ is the set of CLBs in the FPGA design, C is the edge set within a CLB and E_{con} contains all edges that cannot insert FFs due to the FPGA architecture constraints. To reduce the overhead of FF placement after retiming and keep minimal layout disturbance, we do not allow FFs being inserted in global interconnects. Also, we do not allow FFs being inserted in timing edges from LUT inputs to LUT output. Therefore E_{con} contains all edges corresponding to global interconnect and LUT inputs to output edges. Constraints (2), (3), (4) and (5) correspond to the CLB-based FF constraint, FPGA architecture constraint and the basic retiming constraints, respectively.

DEFINITION 2. ***Constraint Driven Retiming (CDR)** Problem is to retime a circuit to achieve minimal clock period such that constraints (2), (3), (4) and (5) are satisfied.*

2.3 CDR Algorithm

We extend the push down retiming algorithm [3] proposed in for ASIC to consider FF constraints and interconnect delay for FPGAs. The basic idea of push down retiming is to iteratively move the FF backward before the critical node until the termination condition is satisfied. There are two main differences comparing CDR to [3]. Firstly, we do not allow FFs being inserted into global interconnects. In addition, the local interconnects are pre-fabricated, and FFs cannot be inserted in the arbitrary positions of the local interconnects, but only at the ends of timing edges. The second difference is the CLB-based FF constraint. In the retiming graph, we pre-record the attribution of each node u, which describes the ID of the CLB and BLE that node u belongs to. In order to find and relocate FFs which violate the CLB-based FF constraint, we use these attributions to update the occupied FF number within a CLB efficiently in retiming. Given a timing node u, the number of occupied FFs within CLB i is N_i, and it should be updated as follows when we perform push down operation on node u.

$$N_i = \begin{cases} N_i - 1, & \text{if } u \text{ is the input of CLB } i \\ N_i + 1, & \text{if } u \text{ is the output of CLB } i \end{cases} \qquad (6)$$

i.e., one FF is pushed into or outside of the CLB.

The CDR algorithm (shown in Algorithm 1) starts from retiming graph initialization (lines 1-4), $\mathbf{UpdateTiming}$ (line 2) performs the sequential timing analysis and results in arrival time of each node. The pseudo code of this routine is shown in Algorithm 2, where Q_r is a queue containing all the nodes requiring retimed in the current iteration, T_{opt} is the current optimal clock period, Q_a is the queue that contains all the nodes with retiming constraint violations. The CDR algorithm enters a loop (lines 5-34) with the termination condition in lines 8-9. [3] presented several efficient termination conditions which can be extended to CDR in a straightforward fashion. After obtaining the arrival time of each node, we push critical nodes into a queue (lines 10-12). δ is a constant to compensate the numerical error in the calculation. Line 13-32 performs backward FF movement iteratively. Whenever a label $r(v)$ changes, we update the occupied FF number inside a CLB by **UpdateFFInCLB** (line 19) based on (6). After each movement, we check the retiming constraint (based on (2), (3), (4) and (5)) violations (line 24). For violations of retiming constraints, a BFS (lines 25-28) is performed to relocate FFs to validate the current retiming solution. The arrival time for each node should be calculated again after FF relocation and those critical nodes are pushed into the queue (lines 29-32). When queue Q_r is empty, the clock period is updated based on the current retiming solution (lines 33-34) and the next round of the iteration begins.

Regarding to the complexity, we have the following lemma.

LEMMA 1. *Lines 20-28 can solve CLB-based FF constraint violation with a constant time complexity.*

PROOF. Suppose an FF is moved into a CLB from $e(p, u)$ to $e(x, p)$ and the total number of occupied FFs becomes $K+1$, which indicates a CLB-based FF constraint violation. As node p is a CLB output node, it must have and only have one ancestor (parent) node x. On the other hand, there always exists a CLB input v that path $v \rightsquigarrow x$ exists unless the root of node x is the output of a constant generator (an LUT that has no inputs). In this case, x can never be on the critical path since we always assume that the arrival time of the constant generator input is $-\infty$. Based on our push down operation, $r(v) \leftarrow r(v) + 1$, only one FF is moved backward across one node. If we assume that the total number of timing nodes within a CLB is M, which is a constant determined by CLB size K, we need at most M times of FF movements to move at least one FF out of the CLB. This solves the CLB-based constraint violation. □

According to [3], the worst case time complexity of push down retiming algorithm is $O(|V|^2|E|)$ for retiming graph $\mathcal{G} = (V, E)$, where $|V|$ and $|E|$ are numbers of nodes and edges in the retiming graph, respectively, and the optimal solution exists if a node v exists such that $r(v) > N$, where N is the total number of FFs in the original circuits. Therefore, the worst case time complexity of CDR is $O(|V|^2|E| \cdot N)$ considering Lemma 1. Our experimental

Algorithm 1 CDR

```
Load retiming graph, $\mathcal{G} = (E, V)$, based on circuit model;
$T \leftarrow$ **UpdateTiming**($\mathcal{G}$);
$Q_r \leftarrow \emptyset$; $r(v) \leftarrow 0, \forall v \in V$; $T_{opt} \leftarrow \infty$;
$N_i \leftarrow$ occupied FF# in $C_i$, $\forall C_i \in \mathcal{CLB}$;
**while** True **do**
  **if** $T_{opt} > T$ **then**
    $T_{opt} \leftarrow T$; $r_{opt}(v) \leftarrow r(v), \forall v \in V$;
  **if** critical edges form a cycle **then**
    Report $T_{opt}, r_{opt}$ and exit;
  {Push all critical nodes to queue}
  **for** each $v \in V$ **do**
    **if** $a(v) > T - \delta$ **then**
      $Q_r \leftarrow Q_r \cup \{v\}$;
  {Iteratively move FFs to optimize critical path delay}
  **while** $Q_r \neq \emptyset$ **do**
    $u \leftarrow$ dequeue($Q_r$);
    **if** $a(u) > T - \delta$ **then**
      Exit if certain termination condition is satisfied;
      $r(u) \leftarrow r(u) + 1$;
      $a(u) \leftarrow \max(a(u), \max_{\forall e(u,v) \in E} d(u, v))$;
      $N_i \leftarrow$ **UpdateFFInCLB**($u$);
      {Validate the retiming solution}
      $Q_a \leftarrow u$;
      **while** $Q_a \neq \emptyset$ **do**
        $p \leftarrow$ dequeue($Q_a$);
        **for** each $e(p, x) \in E$ or $e(x, p) \in E$ **do**
          **if** $N_i > K \wedge e(x, p) \in E || e \in E_{con} \wedge r(x) \neq r(p) || w'(e) \leq 0$ **then**
            $r(x) \leftarrow r(x) + 1$;
            $Q_a \leftarrow Q_a \cup \{x\}$;
            **if** $e(x, p) \in E$ **then**
              $N_i \leftarrow$ **UpdateFFInCLB**($x$);
      {update arrival time}
      $T \leftarrow$ **UpdateTiming**($\mathcal{G}$);
      **for** each $v \in V$ **do**
        **if** $a(v) > T - \delta$ **then**
          $Q_r \leftarrow v$;
  **for** each $v \in V$ **do**
    $T \leftarrow \max(T, a(v))$;
```

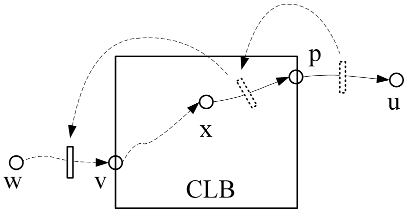

Figure 2: Backward movements for solving CLB-based FF constraints

results show that the algorithm is much more efficient than the worst case complexity. Note that our approach will not introduce area overhead since we only utilize the unused FFs already in FPGAs.

2.4 Experimental Results

We conduct experiments on sequential circuits from MCNC and QUIP benchmarks, which are mapped into FPGAs with LUT size of 4 and cluster size of 10. We use the Berkeley predictive device model [12] for $65nm$ technology node. We use the min-ED (energy-delay product) device setting (V_{dd} = 0.9v and V_{th} = 0.3v), as suggested in [13] for higher yield. All experimental data are collected on a Linux workstation with a $1.9GHz$ Xeon CPU and $2GB$ memory. In our synthesis flow, we first perform logic optimization and technology mapping by SIS [14] and TurboMap [15], respectively. To obtain more flexibility in retiming, we duplicate FFs with high degree fanouts (>10). The circuits are then packed and placed by T-VPack and VPR [9], respectively. Retiming followed by routing (by VPR) is then performed.

Algorithm 2 UpdateTiming($\mathcal{G} = (E, V)$)

```
$a(v) \leftarrow 0, \forall v \in V; T \leftarrow 0; Q \leftarrow V$;
**while** $Q \neq \emptyset$ **do**
  $u \leftarrow$ dequeue($Q$);
  **for** each $e(u, v) \in E$ **do**
    **if** $w'(u, v) = 0$ **then**
      $a(v) \leftarrow \max(a(v), a(u) + d(u, v))$;
      $T \leftarrow \max(T, a(u) + d(u, v))$;
      **if** $v \notin Q$ **then**
        $Q \leftarrow Q \cup v$;
    **else**
      $a(v) \leftarrow \max(a(v), d(u, v))$;
      $T \leftarrow \max(T, a(v))$;
**Return** $T$;
```

To justify the effectiveness of retiming by relaxing BLE based constraints to CLB based constraints, we first extend [3] to consider BLE based constraints by allowing only FF insertion in timing edges from an LUT output to its corresponding FF input, and then compare the algorithm to CDR. Table 1 shows that retiming with CLB based constraints (column *CLB-con*) improves performance by over 5% on average compared to retiming with BLE based constrained (column *[3]-ext*).

Furthermore, we show the effectiveness of constraint-aware retiming by comparing CDR to constraint unaware retiming [3] followed by FF legalization (i.e. locally adjusting FFs until constraints (2), (3), (4) and (5) are satisfied). Table 1 shows that CDR (column *CLB-con*) achieves over 6% performance improvement on average compared to the constraint-unaware retiming with legalization (column *UCR+legalization*). For most of the circuits, no improvement can be obtained since the legalization phase moves FFs back to their original (non-retimed) locations in order to satisfy FF constraints. On the other side, CDR solves FF constraints within optimization procedure and therefore achieves a much better retiming solution.

	BLE-con	CLB-con	
Circuit	[3]-ext	UCR+legalization	CDR
bigkey	9.7	9.6 (-1.03%)	9.4 (-2.08%)
clma	40.9	41.2 (0.73%)	39.5 (-4.13%)
diffeq	22.1	22.5 (1.81%)	20.4 (-9.33%)
dsip	8.4	8.9 (5.95%)	8.1 (-8.99%)
elliptic	28.1	28.1 (0.00%)	27.8 (-1.07%)
frisc	28.1	28.1 (0.00%)	26.3 (-6.41%)
s298	38.8	39.3 (1.29%)	38.4 (-2.29%)
s38417	24.4	24.4 (0.00%)	24.1 (-1.23%)
s38584	18.1	18.1 (0.00%)	16.1 (-11.05%)
tseng	22.8	22.8 (0.00%)	20.9 (-8.33%)
ave	24.14	24.3 (0.66%)	23.1 (-5.49%)
barrel64	14.33	14.92 (4.09%)	14.20 (-5.03%)
mux64_16bit	9.33	8.86 (-5.03%)	8.64 (-2.55%)
mux8_128bit	7.34	7.38 (0.62%)	7.02 (-5.20%)
mux32_16bit	7.62	8.07 (5.99%)	7.63 (-5.78%)
mux8_64bit	6.56	6.52 (-0.68%)	6.31 (-3.33%)
oc_cordic_r2p	30.39	32.47 (6.85%)	30.30 (-7.17%)
oc_wb_dma	17.74	18.69 (5.37%)	17.50 (-6.83%)
oc_cordic_p2r	24.75	25.00 (0.99%)	23.08 (-8.30%)
oc_correlator	107.50	107.78 (0.26%)	104.59 (-3.05%)
ave	25.06	25.52 (1.83%)	24.36 (-5.25%)

Table 1: Clock period (ns) comparisons with different deterministic retiming algorithms

3. STATISTICAL RETIMING

3.1 Statistical Sequential Criticality

Following the statistical timing model in [16], the delay of a timing edge is modeled as a normal distributed random variable [17], and the spatially correlated variation is extracted by [18] and orthogonalized by principle component analysis (PCA) [19].

Under the presence of process variations, we employ the first order statistical timing analysis [17] into our CDR algorithm framework. The critical path delay is then approximated as a normal random variable. To perform retiming under process variations, we need to reconsider the following two problems. *How to judge the performance of a retiming solution represented by a random variable? How to select the critical nodes under the statistical context?*

[20] suggested comparing two random variables a and b by checking the probability of $P(a < b)$ and accepting $a < b$ if $P(a < b) > P_T$, where $0.5 \leq P_T \leq 1.0$. [7] presented a concept of "disutility function", $disu(x)$, which is defined as

$$disu(x) = E(x) + \omega \cdot \sqrt{E(x - E(x))^2} \qquad (7)$$

where $E(x)$ is the mean of random variable x. The probability based comparison in [20] relaxes the deterministic ordering relation with probability P_T and will potentially increase the number of "critical" nodes that are needed to be retimed, which increases the runtime and therefore is not applied. On the other hand, the disutility function targets the worst case performance of the system and may lead to pessimism on statistical delay. To alleviate the pessimism of disutility function, we define the statistical sequential criticality, $seq\_crit(u)$ of node u as

$$seq\_crit(u) = crit(u)^{\alpha} \cdot disu(d(u)) \qquad (8)$$

where $crit(u)$ is the statistical criticality [17] of node u, which provides the probability that node u lies on the critical path, and α is an empiricial constant (1.0 in this paper). Obviously, sequential criticality seeks those timing nodes that have large arrival time and high probability lying on the critical path. This criticality is used to select the critical nodes in the retiming procedure.

We tested several combination parameters ω and α for circuit *bigkey* (the other circuits show the similar trend for these two parameters.). The mean and standard deviation of the critical path delay under different settings are summerized in Table 2, which shows that ω=3 (in (7)) and α=1 (in (8)) is the best combination to improve the timing yield in our algorithm. Note that a small ω make the algorithm focus on reducing mean values of critical path delay, and may produce larger standard deviations, which reduce timing yield. In addition, a medium α reduces the pessimism of disutility function and therefore improves the overall performance, however a big α will over-emphasize statistical criticality and enforce sCDR to consider more timing nodes and therefore reduce the effectiveness of sCDR within 50 iterations.

	α=0 (i.e. [7])	α=1	α=3
ω=3	11.02+1.65	10.99+1.34	11.04+1.39
ω=1	11.01+1.69	10.97+1.54	11.13+1.52
ω=0	11.01+1.70	10.98+1.47	11.16+1.51

Table 2: Mean+standard deviation for *bigkey*

3.2 Statistical Constraints Driven Retiming

We extend our CDR algorithm to sCDR to consider process variations in retiming by performing sequential statistical static timing analysis (S-SSTA). The necessary updates of CDR are addressed as follows.

- The deterministic timer, **UpdateTiming**, is replaced by **UpdateStatTiming**, which is shown in Algorithm 3. In the statistical sequential timer, a forward BFS is first performed to calculate the statistical arrival time of each timing node and arrival tightness probability (ATP) [17]. A backward BFS is then performed to calculate the criticality of each timing node. After that, the statistical critical path delay and the maximum of sequential criticality by (8) are obtained.
- The judgment of performance improvement (line 6 in Algorithm 1) is replaced by comparing the disutility function value (7) of the distribution of the current critical path delay T and that of the current optimal critical path delay T_{opt}.
- The sequential criticality of the current node and the maximum sequential criticality are compared to determine the critical nodes (line 11 and line 31 in Algorithm 1).
- All *max* and *plus* operations are replaced with probabilistic equivalents.
- Termination condition in Algorithm 1 (line 8-9) is no longer valid since it is impossible to identify the set of critical edges deterministically in the context of process variation. Alternatively, we set an upper bound to control the iteration number. Figure 3 shows the performance improvement (mean+3·std) converges in the first 50 iterations for circuit *bigkey*. The same trends are observed for other test cases and therefore we set the 50 as the iteration upper bound in sCDR.

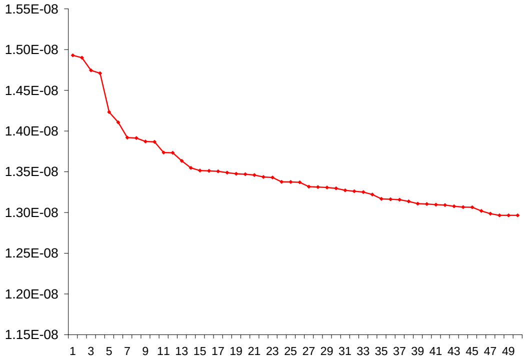

Figure 3: Performance improvement in first 50 iterations (for *bigkey*)

3.3 Experimental Results

In our experiments, we first generate spatial correlation matrix and PCA presented in Section 3.1, then perform sCDR algorithm after placement. Following the setting in [16], we assume a variation in each of L_{eff} and V_{th} of 10%, 10% and 6% at $3\sigma^2$, respectively, to model global, spatial and local variations. To model spatial correlation, each FPGA chip is partitioned into grids such that each grid contains five tiles in one dimension (around $0.5mm$ in $65nm$ technology). The correlation covariance coefficient decreases to 0.1 at $2mm$ distance.

Table 3 compares the effectiveness of sCDR to CDR under process variations. Compared to CDR, the sCDR algorithm reduces the mean and standard deviation of the critical path delay distribution by 0.34% and 0.45% for MCNC testset and by 1.82% and 3.66% for QUIP testset on average, respectively. Note that sCDR obtains up to 16.10% mean reduction and 25.76% standard deviation reduction compared to CDR, which shows the effectiveness and importance of integrating statistical timing analysis in retiming. The runtime for both CDR and sCDR are shown in Table 3 (column *time*). sCDR further optimizes circuits under process variations with about 20x runtime overhead compared to CDR. For two circuits, *clma* and $s38417$, sCDR gives solutions inferior to CDR due to our heuristic metric (8) for selecting the critical nodes. This will be addressed in our future research.

[2] I.e. a 99.73% chance that variation is within +/- 10% or 6% deviated from the nominal value.

	non-retimed		CDR			sCDR		
	mean(ns)	std(ns)	mean (impv)	std (impv)	cpu(s)	mean (impv)	std (impv)	cpu(s)
bigkey	11.32	1.33	11.26 (-0.57%)	1.43 (7.57%)	4	10.99 (-2.99%)	1.34 (1.03%)	27
dsip	10.03	1.98	9.63 (-3.98%)	2.00 (0.84%)	4	9.62 (-4.10%)	2.00 (0.84%)	34
s298	42.73	6.41	40.88 (-4.32%)	6.54 (1.92%)	10	40.88 (-4.33%)	6.51 (1.43%)	95
s38584	19.62	3.69	18.34 (-6.50%)	3.64 (-1.57%)	23	18.42 (-6.08%)	3.40 (-7.91%)	221
s38417	29.37	4.10	29.32 (-0.16%)	3.94 (-3.87%)	31	29.16 (-0.71%)	4.26 (3.96%)	289
clma	44.93	6.04	43.39 (-3.43%)	5.78 (-4.32%)	54	43.85 (-2.42%)	5.80 (-3.89%)	1247
tseng	24.05	4.31	22.40 (-6.88%)	4.03 (-6.69%)	1	22.31 (-7.25%)	4.01 (-7.02%)	13
elliptic	31.03	4.89	30.79 (-0.78%)	4.89 (0.11%)	14	30.02 (-3.24%)	4.87 (-0.27%)	118
frisc	31.03	4.89	29.99 (-3.37%)	4.90 (0.20%)	24	29.81 (-3.92%)	4.84 (-1.03%)	378
diffeq	25.67	5.86	24.53 (-4.43%)	4.96 (-15.44%)	19	24.55 (-4.35%)	4.86 (-17.05%)	145
(ave)	26.98	4.35	26.05 (-3.43%)	4.21 (-3.25%)	18	25.96 (-3.77%)	4.19 (-3.70%)	257
barrel64	17.00	2.84	16.42 (-3.43%)	2.68 (-5.75%)	1	15.84 (-6.86%)	2.40 (-15.57%)	5
mux64_16bit	11.02	1.70	10.36 (-5.94%)	1.67 (-1.78%)	2	10.20 (-7.40%)	1.61 (-5.10%)	14
mux8_128bit	8.46	1.41	8.17 (-3.44%)	1.40 (-0.61%)	1	7.95 (-6.06%)	1.24 (-12.27%)	7
mux32_16bit	9.89	1.79	8.92 (-9.85%)	1.61 (-10.12%)	2	8.30 (-16.10%)	1.33 (-25.76%)	22
mux8_64bit	8.16	1.45	7.58 (-7.10%)	1.29 (-11.39%)	3	7.42 (-9.14%)	1.20 (-17.30%)	34
oc_cordic_r2p	36.06	5.71	35.20 (-2.39%)	5.41 (-5.33%)	4	34.40 (-4.61%)	5.19 (-9.20%)	68
oc_wb_dma	21.49	3.32	20.75 (-3.46%)	3.13 (-5.62%)	18	20.67 (-3.84%)	3.10 (-6.61%)	543
oc_cordic_p2r	29.29	4.62	28.99 (-1.04%)	4.64 (0.33%)	4	27.74 (-5.31%)	4.20 (-9.22%)	73
oc_correlator	119.37	23.64	116.65 (-2.28%)	23.23 (-1.73%)	10	115.78 (-3.00%)	23.09 (-2.33%)	40
(ave)	28.97	5.17	28.12 (-2.96%)	5.01 (-3.08%)	5	27.59 (-4.78%)	4.82 (-6.74%)	90

Table 3: Comparison with CDR and sCDR

Algorithm 3 UpdateStatTiming($\mathcal{G} = (E, V)$)

```
$a(v) \leftarrow 0, \forall v \in V; T \leftarrow 0; Q \leftarrow V;$
{Calculate ATP for each timing edge and statistical arrival time for each node.}
while $Q \neq \emptyset$ do
    $u \leftarrow$ dequeue($Q$);
    for each $e(u, v) \in E$ do
        $atp(e) \leftarrow$ ATP($e$);
        if $w'(u, v) = 0$ then
            $a(v) \leftarrow \max(a(v), a(u) + d(u, v))$;
            $T \leftarrow \max(T, a(u) + d(u, v))$;
            if $v \notin Q$ then
                $Q \leftarrow Q \cup v$;
        else
            $a(v) \leftarrow \max(a(v), d(u, v))$;
            $T \leftarrow \max(T, a(v))$;
{Initialize criticality for sink nodes}
for each $e(u, v) \in E$ do
    if $w'(u, v) > 0$ then
        $Q \leftarrow Q \cup \{u\}$;
        $crit(u) \leftarrow \frac{1.0}{N_{FF}+N_{PO}}$;
    else
        $crit(v) \leftarrow 0$;
{Calculate criticality for each timing node.}
while $Q \neq \emptyset$ do
    $v \leftarrow$ dequeue($Q$);
    for each $e(u, v) \in E$ do
        if $w'(u, v) = 0$ then
            $crit(u) \leftarrow crit(u) + atp(e) \cdot crit(v)$;
            if $v \notin Q$ then
                $Q \leftarrow Q \cup u$;
Return $T$ and $\max_{\forall v \in V} seq\_crit(v)$;
```

3.4 Prediction for Effectiveness of sCDR

Table 3 shows that sCDR has over 10x runtime overhead compared to CDR while the effectiveness, i.e. timing yield improvement compared to CDR, of sCDR is not significant for certain designs (e.g. clma). Therefore it is necessary to develop an effective way to predict the potential gain (further improvement compared to CDR) of sCDR. In fact, the improvement due to sCDR depends on the topology of the timing network of a design. Figure 4 shows two different topologies for a design with 3 inputs and 2 outputs. The delay values are labeled in timing edges. In topology (a), all input-to-output paths share a long segment (A, B) and an FF is inserted in it. As the deterministic retiming CDR reduces the deterministic timing critical path length, CDR can achieve a good balance of delay among all input-to-output critical paths in topology (a). However, sCDR can hardly further improve timing yield for this case since either forward or backward move of FFs will change the statistical criticality of all the input-to-output paths and it tends to accept the solution produced by CDR. In topology (b), there are two independent input-to-output paths, i.e., $p_1 = (PI, C, D, PO)$ and $p_2 = (PI, E, F, PO)$. CDR achieves the best critical path delay for both paths without considering variations. Suppose node E is more sensitive to process variations, sCDR will move the FF in segment (E, F) backward, reduce the statistical criticality of node E, and increase the tolerance of the process variations. Note that the delay distribution on path p_1 remains the same after sCDR while the timing yield for the overall network increases.

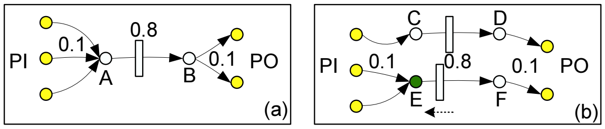

Figure 4: (a) Circuit without independent path (b) Circuit with 2 independent paths

Motivated by the example shown in Figure 4, we find that a larger number of independent critical paths indicates greater effectiveness of sCDR. Note that there usually exist few fully independent paths[3] in a real design and most of them overlap with each other. We propose the concept of *cluster of coupled paths* (CCP) to evaluate the topology of a timing network. Those paths closely coupled with each other are clustered and form a CCP. For an instance, in Figure 4, there is one CCP in (a) and two in (b). Intuitively, CCP reflects the coupling strength of a timing network. More CCPs indicates fewer path sharing and more improvement is expected from sCDR.

To predict the effectiveness of sCDR, we check the number of CCPs for the top-N deterministic timing critical and near-critical paths (CNP). We first enumerate the top-N CNPs, $p_1, \cdots, p_N$, and then calculate, l_{ij}, the total length of all overlapped segments between each two paths p_i and p_j. A normalized coupling strength coefficient matrix (CCM) $L_N = \{l_{ij}/l_{max}\}$ is then generated, where l_{max} is the length of the most critical path, i.e., clock period. Given CCM L_N for the top-N CNPs, the number of CCPs of these paths can be measured based on singular value decomposition (SVD) [22]. The number of large singular values (LSV) can

[3] I.e, a path with no overlap to any other paths.

approximate the number of CCPs in the top-N CNPs. Figure 5 summarizes the derivation of the predictor for the effectiveness of sCDR.

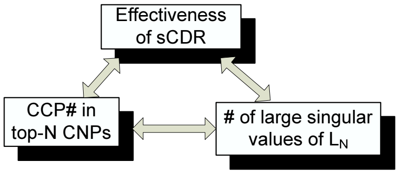

Figure 5: Prediction for the effectiveness of sCDR

As an example for predicting the effectiveness of sCDR, again we consider the two networks shown in Figure 4, the L_2 matrices for (a) and (b) are as follows.

$$L_2(a) = \begin{bmatrix} 1 & 0.8 & 0.8 \\ 0.8 & 1 & 0.8 \\ 0.8 & 0.8 & 1 \end{bmatrix} \tag{9}$$

$$L_2(b) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.8 \\ 0 & 0.8 & 1 \end{bmatrix} \tag{10}$$

The singular values for $L_2(a)$ and $L_2(b)$ are $[2.6, 0.2, 0.2]$ and $[1.8, 1.0, 0.2]$, respectively, and the number of singular values larger than 1.0 are one and two. This matches our observation very well. Therefore, we have the following empirical prediction for the effectiveness of our sCDR algorithm.

Claim 1. *Larger LSV number of $L_N(n)$ matrix for network n indicates more CCPs so that more effective of sCDR, i.e., more timing yield improvement, is expected compared to CDR.*

To verify Claim 1, we extract the top-30 CNPs for all test circuits and calculate their LSV numbers. Figure 6 compares the LSV number and the reduction of the mean clock period by sCDR compared to CDR[4]. The overall trend of the results confirm the above theory. This predictor enables us to perform sCDR only when large gain is expected. *Note that the prediction is "binary" but not "continuous" in the sense that it indicates a good gain by sCDR if the LSV number is larger than a threshold, however, a larger LSV number does not necessarily mean a higher gain.* We find that setting the LSV threshold of 10 for top-30 CNPs leads to good results. The runtime of the prediction is negligible compared to CDR. In general, our SVD based predictor quatitively gives the *sensitivity* of the network toplogy to the process variations and thereby can be used in any other statistical timing optimization procedures besides retiming.

4. CONCLUSION

To utilize retiming in modern FPGAs, we have proposed an optimal FF constraint driven retiming algorithm, namely CDR, with polynomial time complexity. Furthermore, we have extended our CDR algorithm to $sCDR$ to handle process variations by considering statistical timing analysis during the retiming procedure.

The experimental results show that CDR achieves on average 5.54% and up to 11.6% performance improvement compared to the optimal retiming algorithm with the BLE based FF constraints. Tested on a rich set of circuits from MCNC and QUIP testset, sCDR reduces up to 6.93% of the mean value and 17.49% of the standard deviation of the distribution of the clock period compared to CDR. To justify the runtime overhead of sCDR, we propose an effective way to predict the potential gain from sCDR, which enables us to perform sCDR only when it is beneficial.

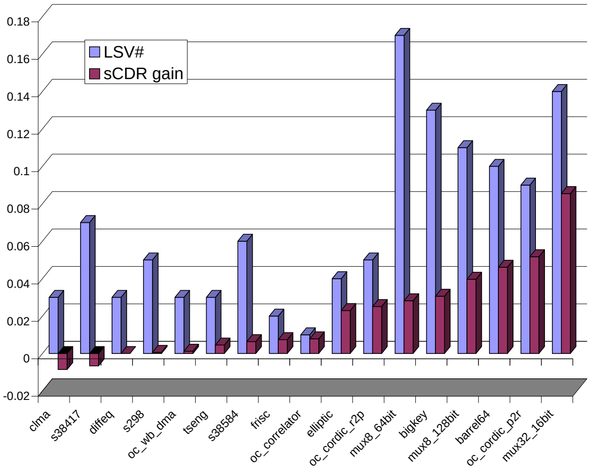

Figure 6: LSV number vs. further statistical delay reduction by sCDR

5. REFERENCES

[1] C.E.Leiserson and J.B.Saxe, "Retiming synchronous circuitry," *Algorithmica*, pp. 5–35, 1991.

[2] J. Cong and S. K. Lim, "Retiming-based timing analysis with an application to mincut-based global placement," *TCAD*, 2004.

[3] C. Lin and H. Zhou, "Optimal wire retiming without binary search," in *ICCAD*, 2004.

[4] D. P. Singh and S. D. Brown, "Integrated retiming and placement for field programmable gate arrays," in FPGA, 2002.

[5] D. P. Singh and S. D. Brown, "Incremental placement for layoutdriven optimizations on fpgas," in ICCAD, 2002.

[6] V. Manohararajah, D. P. Singh, and S. D. Brown, "Incremental retiming for fpga physical synthesis," in *DAC*, 2005.

[7] J. Wang and H. Zhou, "Minimal period retiming under process variations," in *GLSVLSI*, 2004.

[8] M. Ekpanyapong, et al, "Retiming-based timing analysis with statistical bellman-ford algorithm," in *ASPDAC*, 2006.

[9] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[10] "Xilinx product datasheets," in *http://www.xilinx.com/literature*.

[11] D. Lewis and et al, "The stratix ii routing and logic architecture," in *FPGA*, 2005.

[12] U. of Berkeley Device Group, "Predictive technology model," in *http://www.device.eecs.berkeley.edu/ ptm/mosfet.html*, 2002.

[13] H.-Y. Wong, L. Cheng, Y. Lin, and L. He, "FPGA device and architecture evaluation considering process variations," in *ICCAD*, 2005.

[14] E. M. Sentovich et. al., "SIS: A system for sequential circuit systhesis," in *Department of Electrical Engineering and Computer Science, Berkeley, CA 94720*, 1992.

[15] J. Cong and C. Wu, "An efficient algorithm for performance optimal fpga technology mapping with retiming," *TCAD*, 1998.

[16] Y. Lin, M. Hutton, and L. He, "Placement and timing for fpgas considering variations," in *FPL*, 2006.

[17] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in *DAC*, June 2004.

[18] J. Xiong, V. Zolotov, and L. He, "Robust extraction of spatial correlation," in *ISPD*, April 2006.

[19] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," in *ICCAD*, 2003.

[20] J. Xiong and L. He, "Fast buffer insertion considering process variations," in *ISPD*, April 2006.

[21] "Altera: QUIP for Quartus II V5.0," in *http://www.altera.com/education/univ/*.

[22] J. E. Gentle, *Singular Value Factorization*. Berlin:Springer-Verlag, 1998.

[23] U. Seidl, et al,"Performance-Directed Retiming for FPGAs Using Post-Placement Delay Information," in *DATE*, 2004.

[4] Two circuits, mux64_16bit and dsip, are not shown in this chart. They have big LSV numbers while they do not gain much by sCDR due to the heuristic objective function