Minimal Period Retiming Under Process Variations *

Jia Wang and Hai Zhou
Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208

ABSTRACT

With aggressive scaling down of feature sizes in VLSI fabrication, process variations have become a critical issue in designs. With process variations, timing optimization should consider the randomness introduced in delays. This paper considers how to retime a circuit under process variations. A statistical retiming problem is defined on the concept of a disutility function. Based on a new minimal period retiming algorithm, two algorithms are presented for the statistical retiming problem. Both theoretical and experimental results are given.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS-Design Aids

General Terms

Algorithms, Design

Keywords

retiming, process variations, statistical timing analysis

1. INTRODUCTION

Retiming is an effective optimization technique in synchronous circuit design. It was first proposed by Leiserson and Saxe in [1]. Retiming optimizes the circuit by relocating the flip-flops. Minimal period retiming targets at minimizing the clock period of a circuit which is equal to the longest path delay between two consecutive flip-flops. Under the assumption that the delays have fixed values, this problem has been solved by a binary search over feasible periods through fixed period checking[2, 3].

As the geometries in deep sub-micron technology keep descending, process variations become significant and could make the timing of fabricated circuits far from what is designed. Many works focus on the timing analysis of combinational circuits under process variations. Retiming becomes more complicated when process variations are considered because we need to optimize a circuit under uncertainty. Variations could make a less critical path dominant, which could make the result of retiming unreliable. Therefore, how to incorporate process variations into retiming emerges as a challenging problem.

In this paper, we study the problem of minimal period retiming under process variations. We propose a disutility function as a criteria for measuring the distribution of clock period under process variations. After we present the statistical retiming problem, two algorithms are given to solve the problem and tested with the ISCAS89 benchmarks. Both algorithms are based on a new pushing down minimal period retiming algorithm discovered by us. Experimental results show that our approaches are effective.

The rest of the paper is organized as following. In Section 2, we introduce the disutility function and then define the statistical retiming problem. In Section 3, we briefly discuss the methods for statistical timing analysis. In Section 4, we present our algorithms to solve the statistical retiming problem. Experimental results are given in Section 5 and conclusions are drawn at the end.

2. PROBLEM DEFINITION

2.1 Model Circuit As Graph

A sequential circuit could be modeled as a directed graph $G = (V, E)$. Each vertex $v \in V$ represents a gate with delay $d(v)$ and each edge $(u, v) \in E$ represents a signal from u to v. We define the weight of edge $w : E \rightarrow \mathcal{Z}^*$, where $\mathcal{Z}^*$ is the set of non-negative integers, as the number of the flip-flops on the edges.

Following Leiserson and Saxe [2], a retiming is a labeling of the vertices $r : V \rightarrow \mathcal{Z}$. The edge weights after retiming, say w', are given as

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

As $w'(u, v)$ should be greater than or equal to 0, r should satisfy

$$r(u) \leq w(u, v) + r(v), \forall (u, v) \in E \qquad (2)$$

In our pushing down retiming algorithm, we always move flip-flops from the fan-outs of a gate toward its fan-ins. So $r(v)$ will be non-negative and represent the number of flip-flops moved.

*This work is supported by the NSF under CCR-0238484.

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

When we remove all the edges e satisfying $w(e) > 0$, we get a directed acyclic graph (DAG) with the connected components as the combinational parts of the sequential circuit. Because this DAG depends on w, we write it as $G_w = (V, E_w)$. Define

$$FI_w(v) = \{u : (u, v) \in E_w\} \tag{3}$$

as fan-ins of v in G_w and

$$FO_w(v) = \{w : (v, w) \in E_w\} \tag{4}$$

as fan-outs of v similarly.

Denoting the gate arrival time by $t : V \to \mathcal{R}$, we have

$$t(v) = (\max_{u \in FI_w(v)} t(u)) + d(v) \tag{5}$$

where $t(v) = d(v)$ if $FI_w(v) = \phi$. As G_w is a DAG, equation (5) uniquely determines t. The clock period of the circuit must be larger than or equal to $t(v), \forall v \in V$. Thus the clock period of the circuit should be

$$T_w = \max_{v \in V} t(v) \tag{6}$$

Obviously, T_w could be expressed alternatively as below

$$T_w = \max_{l \in \mathcal{L}_w} \sum_{v \in l} d(v) \tag{7}$$

where $\mathcal{L}_w$ is the set of the paths in G_w.

2.2 Disutility Function

The clock period T_w becomes a random variable under process variations. Suppose the price of a produced circuit with clock period T is $g(T)$, where $g(T)$ is non-increasing with T. Then $E(g(T_w))$ is the expectation of what we could obtain by selling one circuit with the random clock period T_w. Different retimings give different distributions of T_w. We always prefer the distribution with larger $E(g(T_w))$. As traditional retiming algorithm is looking for minimum, we introduce a disutility function $disu(T_w)$ as the measurement of the distribution. The smaller the $disu(T_w)$, the better the distribution. It could be derived from g as

$$disu(T_w) = 1/E(g(T_w)) \tag{8}$$

Or use $disu(T_w) = -E(g(T_w))$ if $E(g(T_w))$ could be 0.

Another way of defining the disutility function is to derive it directly from the mean and the variance of T_w

$$disu(T_w) = E(T_w) + \omega * \sqrt{E(T_w - E(T_w))^2} \tag{9}$$

where ω is the weight of the variance.

2.3 Statistical Retiming

Based on the discussion above, we define the statistical retiming problem as following,

PROBLEM 1 (STATISTICAL RETIMING). *In a directed graph G representing a circuit, gate delays are given as random variables due to process variations. Suppose the minimal clock period on which the circuit could function correctly is T_w. We are required to retime G to achieve minimal $disu(T_w)$ for a given disutility function disu that measures the distribution of the random clock period due to process variations.*

3. STATISTICAL TIMING ANALYSIS

From Section 2.1, the clock period T_w of a sequential circuit is determined by the combinational parts separated by consecutive flip-flops. Given gate delays as random variables, we could compute the distribution of T_w by statistical timing analysis techniques on combinational circuits.

One way to perform such computation is using the Monte Carlo method to collect the distribution information by simulation. However, as the number of gates could be tremendous, it requires many iterations to get reliable results.

Another way is to compute the distribution of T_w directly from the distributions of gate delays. The hardest part is to handle the correlations. Even if the correlation between gate delays could be omitted, the correlation due to reconvergent paths is known to be a problem. The method proposed by Chang and Sapatnekar [4] based on Clark [5] solves the problem consistently. We prefer to use this method because it could compute the distribution of the arrival time of each gate, which could be reused between iterations in our algorithm to achieve efficiency and accuracy.

This method begins with approximating the gate delays $d(v)$ as multivariate normal distribution. Then by using *principal component analysis*, which can reduce the number of variables to make the algorithm efficient, we find m uncorrelated, or equivalently, independent random variables $p_i, 1 \le i \le m$ with standard normal distribution so that after constructing the set

$$\mathcal{P} = \{a_0 + \sum_{k=1}^{m} a_k * p_k : a_i \in \mathcal{R}, 0 \le i \le m\} \tag{10}$$

we have $d(v) \in \mathcal{P}$. Obviously when $X, Y \in \mathcal{P}$, we have $X + Y \in \mathcal{P}$. We could also approximate $\max\{X, Y\}$ with some $Z \in \mathcal{P}$. Then, according to (5) and (6), the clock period of the circuit, T_w, could be approximated with some element in $\mathcal{P}$.

Although this approach is only an approximation, [4] shows that the result is acceptable compared to the result obtained by the Monte Carlo method.

4. STATISTICAL RETIMING ALGORITHM

4.1 Pushing Down Retiming

Compared with the traditional retiming algorithms [2] [3], the pushing down retiming algorithm discovered by us focuses on iteratively adjusting the longest combinational path in the circuit instead of checking whether a given clock period is feasible. At the end of each iteration, it gives a retimed circuit with smaller clock period or proves the current retiming is optimum. Below is the simplified pushing down retiming algorithm. We omit some details in the condition checking at line 9 for simplicity but they will be explained following the algorithm.

Algorithm `Pushing Down Retiming`
`INPUTS:` G, w, $d(v)$
`OUTPUTS: optimal` $maxT$, $r(v)$

1. `Set all` $r(v)$ `to` 0
2. `Compute` $t(v)$ `using (5)`
3. `Set` $maxT$ `to` $\max_v t(v)$

`LOOP:`

4. `Save current` $maxT$ `and` $r(v)$ `as optimal`

```
5.  Initialize stack and push all v
    satisfy t(v) = maxT into it
6.  While stack is not empty
7.  Pop v from stack
8.  If t(v) ≥ maxT
9.      If there is a cycle in G_m
10.         report optimal maxT and r(v)
11.         algorithm termination
12.     Increase r(v) by 1
13.     Set t(v) to d(v)
14. For each fan-out u of v
15.     If r(v) > w(v,u) + r(u)
16.         Increase r(u) by r(v) − w(v,u)
17.         Set t(u) to t(v) + d(u)
18.     Else if r(v) = w(v,u) + r(u)
    and t(u) < t(v) + d(u)
19.         Set t(u) to t(v) + d(u)
        Else
20.         Continue the for loop on line 14
21.     If u is not in stack
22.         Push u to stack
23.  Set maxT to max_v t(v)
24.  Goto LOOP
```

The algorithm starts with the original circuit as the initial feasible solution. The initial arrive time $t(v)$ is computed one by one according to the order determined by a topological sort [6] on G_w. Then the initial $maxT$ is calculated, which is the clock period of the current circuit.

In each iteration started at line 4, we find all the nodes with the arrival time $maxT$ and putting them into *stack* for updating in the following While loop. Those nodes are the end-points of the critical paths. In each loop, we adjust a path with delay not less than $maxT$ by moving a flip-flop from the fan-outs toward the fan-ins. However, this may change the arrival time of the fan-outs. So we update the arrival time of such nodes and put them into the *stack* for later updating if that is needed.

If we can retime the circuit to yield a smaller clock period, we will leave the While loop with an empty *stack* and start a new iteration. If not, we will stop the loop and report the optimal solution. We perform the checking at line 9 as follows. Let $p(v)$ be the starting gate of the longest combinational path to v. Let $m(v)$ be one of the gates which satisfy $\delta_r(m(v)) < \delta_r(v)$, where $\delta_r(v)$ stands for the increase of $r(v)$. When no such gate exists, we set $m(v)$ to *nil*. The graph G_m is constructed with the vertices as V and the edges as $v \rightarrow m(v)$ when $m(v)$ is not *nil*. The $p(v)$s are initialized when we initialize $t(v)$ on line 2. The $m(v)$s are set to *nil* at the beginning. Then $p(v)$ will be updated whenever we change $t(v)$ and $m(v)$ will be updated whenever we change $r(v)$.

We proved the following result.

THEOREM 1. *When the pushing down retiming algorithm terminates, we achieve the minimal clock period.*

Furthermore, this algorithm is more efficient than the traditional ones based on binary search.

As this algorithm only needs local information when making decisions on how to move the flip-flops and keeps valid arrival time of each gate across iterations, we could combine the method in [4] into such steps. That is the following statistical retiming algorithm.

4.2 Statistical Retiming Algorithm

As stated in Section 2.3, we are required to minimize $disu(T_w)$. However, as we could not identify the longest combinational paths when considering random delays, we need to find the combinational paths so that after adjusting them, $disu(T_w)$ could be smaller. Because $T_w = \max_v t(v)$ according to (6), we could adjust the gates with the maximum $disu(t(v))$. As $\max_v disu(t(v))$ is not $disu(T_w)$, the chance we could get any improvement on $disu(T_w)$ actually depends on whether the decreasing of $\max_v disu(t(v))$ means the decreasing of $disu(\max_v t(v)) = disu(T_w)$. When the algorithm terminates, we need to compute the distribution of T_w and then $disu(T_w)$ so that we could check if we get any improvement.

Algorithm Statistical Retiming
INPUTS: G, w, $d(v)$
OUTPUTS: optimal max_disu and $r(v)$

```
1.  Set all r(v) to 0 and all inc(v) to 0
2.  Compute t(v) using (5)
3.  Set max_disu to max_v disu(t(v))
LOOP:
4.  Save current max_disu and r(v) as optimal
5.  Initialize stack and push all v
    satisfy disu(t(v)) = max_disu into it
6.  Set n to the number of element in stack
7.  While stack is not empty
8.  Pop v from stack
9.  If disu(t(v)) ≥ max_disu
10.     If inc(v) > 2 * n
11.         report optimal maxT and r(v)
12.         algorithm termination
13.     Increase r(v) and inc(v) both by 1
14.     Set t(v) to d(v)
15. For each fan-out u of v
16.     If r(v) > w(v,u) + r(u)
17.         Increase r(u) by r(v) − w(v,u)
                and inc(u) by 1
18.         Set t(u) to t(v) + d(u)
19.     Else if r(v) = w(v,u) + r(u)
20.         Compute t(u) by (5)
        Else
21.         Continue the for loop on line 15
22.     If u is not in stack
23.         Push u to stack
24.  Set max_disu to max_v disu(t(v))
25.  Goto LOOP
```

In this algorithm, the sums and maximums of random variables are computed by using the method in [4]. On the line 19 and 20 of the algorithm, we need to compute $t(u)$ using (5) but not simply update it as in the counterpart of the pushing down retiming algorithm.

This algorithm uses a different method from the pushing down retiming algorithm as shown on line 10 to check whether max_disu could be smaller. The method is based on the observation that if max_disu could not be smaller, *stack* will never be empty so that $r(v)$ will be increased to infinity. Thus we use $inc(v)$ to record the increase of $r(v)$ and $2*n$ as the upper bound of such increasing. This method may terminate the algorithm too early so that we could not reach at global minimum. However, the experimental results in Section 5 show we could gain much with it.

4.3 Alternative Algorithm

If the disutility function $disu$ satisfies

$$disu(X+Y) = disu(X)+disu(Y) \quad (11)$$
$$disu(\max\{X,Y\}) = \max\{disu(X), disu(Y)\} \quad (12)$$

where X and Y are random variables, according to (7) we have

$$\begin{aligned} disu(T_w) &= disu(\max_{l\in\mathcal{L}_w}\sum_{v\in l} d(v)) \\ &= \max_{l\in\mathcal{L}_w}\sum_{v\in l} disu(d(v)) \end{aligned}$$

Thus, we could compute $disu(d(v))$ at the beginning as gate delays and use pushing down retiming algorithm to achieve global minimum. We use this approach as our alternative algorithm.

As (11) and (12) do not hold for most disutility functions, the alternative algorithm may not give a global optimum for them. However, when we use the disutility function that makes (11) and (12) hold approximately, such as the functions defined in (9), the alternative algorithm could still optimize the circuit. But if we use the disutility function like those defined in (8), the alternative algorithm will not work. The experimental results shown in Section 5 confirm our judgment.

4.4 Lowerbound on Disutility Functions

With the alternative algorithm above, we could give a lowerbound of the $disu(T_w)$ as Theorem 2.

THEOREM 2. *When the disutility function disu satisfies*

$$disu(X+Y) \geq disu(X)+disu(Y) \quad (13)$$
$$disu(Z) \geq 0 \quad (14)$$

where X, Y, Z are random variables and Z is always larger than 0, the lowerbound of $disu(T_w)$ could be given as

$$\min_{w\in\mathcal{W}} disu(T_w) > minT \quad (15)$$

where $\mathcal{W}$ is the set of the locations of flip-flops after different retimings and $minT$ is the optimum obtained by the pushing down algorithm with gate delays as $disu(d(v))$.

PROOF. First, taking two random variables X and Y, from (13) we have

$$\begin{aligned} disu(\max\{X,Y\}) &= disu(X+(\max\{X,Y\}-X)) \\ &\geq disu(X)+disu(\max\{X,Y\}-X) \end{aligned}$$

Then as the random variable $\max\{X,Y\}-X$ is always larger than 0, from (14) we have

$$disu(\max\{X,Y\}) \geq disu(X)$$

Similarly we get $disu(\max\{X,Y\}) \geq disu(Y)$. So,

$$disu(\max\{X,Y\}) \geq \max\{disu(X), disu(Y)\} \quad (16)$$

Thus with (13) and (16) and according to (7), we have

$$\begin{aligned} \min_{w\in\mathcal{W}} disu(T_w) &= \min_{w\in\mathcal{W}} disu(\max_{l\in\mathcal{L}_w}\sum_{v\in l} d(v)) \\ &\geq \min_{w\in\mathcal{W}}\max_{l\in\mathcal{L}_w} disu(\sum_{v\in l} d(v)) \\ &\geq \min_{w\in\mathcal{W}}\max_{l\in\mathcal{L}_w}\sum_{v\in l} disu(d(v)) \\ &= minT \end{aligned}$$

□

When we use the disutility function by setting $\omega = 0$ in (9), conditions (13) and (14) will be satisfied. So the lowerbound $minT$ is the minimum mean value of the T_w. We will compare the results of our statistical retiming algorithm and the alternative algorithm to this lowerbound in Section 5.

Because we only relies on the selection of the disutility function but not the distribution of the delays, this lowerbound holds for distributions other than the multivariate normal distribution. And the lowerbound of the minimum mean value of T_w could be computed according to the discussion above.

5. EXPERIMENTAL RESULTS

We implement the algorithms in standard C++ and run them under RedHat Linux 9.0 with two 933MHz Pentium III processor and 512M memory. The program itself could only uses one cpu at a time. It could handle both the fixed delays as well as random delays. The benchmark is selected from ISCAS89.

The random gate delays are generated as following. m is the square root of the number of the gates. a_0 are uniform distributed on $[1,2]$. Then $a_i, 1 \leq i \leq m$ are uniform distributed such that $\sum_{i=1}^{m} a_i^2 \leq a_0/3$.

For every circuit, we retime it with both the statistical retiming algorithm and the alternative algorithm. The results are divided into two tables by different types of disutility functions. The resulting clock period T_w is computed using the same method as the initialization in section 4.2. The distribution of T_w is shown as $E(T_w)+\sqrt{E(T_w-E(T_w))^2}$. The running time of each test case depends on the size of the circuit and ranges from several seconds to a few minutes.

Table 1 shows the results when we select the disutility function as (9) and set ω to 0,1 and 3. The lowerbound of the mean value is obtained when we set $\omega = 0$ in the alternative algorithm as stated in Section 4.4. From the table we could see both algorithms give almost the same results except for s15858, which may be caused by sticking into a local minimum. And as the results are closed to the lowerbound, the optimization is acceptable.

Table 2 shows the results when we use the disutility function as (8) with the following $g(t)$

$$g(t) = \begin{cases} 4.0, & if\ t < \frac{T_{ref}}{4} \\ 2.0, & if\ \frac{T_{ref}}{4} \leq t < \frac{T_{ref}}{2} \\ 1.0, & if\ \frac{T_{ref}}{2} \leq t < T_{ref} \\ 0.5, & if\ T_{ref} \leq t < T_{ref}*2 \end{cases}$$

where T_{ref} is the reference clock period obtained as the sum on the column* in Table 1. We also compute the T_w before the retiming for comparing the algorithms. The results of

Table 1: Results when using disutility functions as (9)

circuit	lowerbound	alternative algorithm			statistical retiming algorithm		
		$\omega = 0$	$\omega = 1^*$	$\omega = 3$	$\omega = 0$	$\omega = 1$	$\omega = 3$
s1196	36.89	37.92+1.60	37.92+1.60	37.92+1.60	37.92+1.60	37.92+1.60	37.92+1.60
s13207	81.51	83.17+1.99	83.17+1.99	83.17+1.99	83.17+1.99	83.17+1.99	83.17+1.99
s15850	63.37	67.69+1.28	67.64+1.27	67.67+1.27	102.56+2.17	102.18+2.14	101.35+2.15
s27	8.37	8.52+0.69	8.52+0.69	8.52+0.69	8.52+0.69	8.52+0.69	8.46+0.67
s38417	50.38	55.59+0.42	55.56+0.43	55.60+0.42	55.32+0.44	55.37+0.39	55.20+0.39
s38584	63.82	67.24+1.59	67.30+1.56	67.30+1.56	68.12+1.54	68.16+1.53	67.90+1.54
s400	10.33	11.77+0.55	11.88+0.56	12.17+0.62	11.77+0.55	11.86+0.54	11.82+0.50
s510	17.38	19.06+0.82	19.04+0.85	18.96+0.83	18.52+0.85	18.24+0.73	19.40+0.79
s5378	32.48	35.52+0.77	35.46+0.80	35.37+0.78	36.59+0.87	36.12+0.80	35.31+0.75
s9234	61.20	64.92+1.25	65.17+1.23	65.32+1.23	65.86+1.27	66.00+1.24	64.44+1.21

Table 2: Results when using disutility functions as (8)

circuit	T_{ref}	alternative algorithm		statistical retiming algorithm		before retiming	
		T_w	$disu(T_w)$	T_w	$disu(T_w)$	T_w	$disu(T_w)$
s1196	37.92+1.60	37.92+1.60	1.09	37.92+1.60	1.09	37.92+1.60	1.09
s13207	83.17+1.99	83.99+1.99	1.16	83.17+1.99	1.09	94.70+2.25	2.00
s15850	67.64+1.27	105.36+2.24	2.00	67.36+1.21	1.05	133.56+2.80	2.14
s27	8.52+0.69	8.52+0.69	1.09	8.52+0.69	1.09	8.52+0.69	1.09
s38417	55.56+0.43	56.80+0.79	1.73	55.21+0.41	1.01	70.30+2.49	2.00
s38584	67.30+1.56	79.36+2.32	2.00	67.84+1.55	1.15	89.02+2.41	2.00
s400	11.88+0.56	13.03+0.90	1.59	11.82+0.50	1.06	16.24+1.55	1.99
s510	19.04+0.85	19.30+0.90	1.15	19.06+0.81	1.08	18.08+1.41	1.05
s5378	35.46+0.80	38.27+0.88	1.98	35.74+0.73	1.14	41.22+1.48	2.00
s9234	65.17+1.23	66.54+1.37	1.37	64.58+1.22	1.04	68.07+1.58	1.74

the statistical retiming algorithm are optimized compared to the original circuits. The results of the alternative algorithm are not as good as the statistical retiming. That matches our expectation in Section 4.3.

6. CONCLUSION

In this paper, we present a criteria called disutility function for measuring the distribution of clock period of fabricated circuits and define the statistical retiming problem. We propose a statistical retiming algorithm and an alternative algorithm to solve the minimal period retiming problem under process variations. For some kinds of disutility functions, both of the algorithms give nearly the same optimization results. For other kinds of disutility functions, the statistical retiming algorithm is superior to the alternative algorithm.

7. REFERENCES

[1] C. E. Leiserson and J. B. Saxe. *Optimization Synchronous Systems.* Journal of VLSI and Computer Systems, 1:41–67, 1983

[2] C. E. Leiserson and J. B. Saxe. *Retiming Synchronous Circuitry.* Algorithmica, 6(1), 1991

[3] N. Shenoy and R. Rudell. *Efficient Implementation of Retiming.* Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pages 226-233, 1994.

[4] H. Chang, S. S. Sapatnekar. *Statistical Timing Analysis Considering Spatial Correlations Using A Single PERT-like Traversal.* Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pages 621-625, 2003

[5] C. E. Clark. *The Greatest of a Finite Set of Random Variables.* Operational Research, Vol.9, No.2 (Mar.-Apr., 1961), 145-162

[6] T. H. Cormen, C. E. Leiserson, and R. H. Rivest. *Introduction to Algorithms.* MIT Press, 1989