Skew Sensitivity Minimization of Buffered Clock Tree

Jae Chung and Chung-Kuan Cheng

Department of Computer Science and Engineering
University of California at San Diego; La Jolla, CA 92093-0114

Abstract

Given a topology of clock tree and a library of buffers, we propose an efficient skew sensitivity minimization algorithm using dynamic programming approach. Our algorithm finds the optimum buffer sizes, its insertion levels in the clock tree, and optimum wire widths to minimize the skew sensitivity under manufacturing variations. Careful fine tuning by shifting buffer locations at the last stage preserves the minimum skew sensitivity property and reduce the interconnect length. For a given clock tree of n points and a library of s different buffer sizes, the run time of the presented algorithm is $O(log^3 n \cdot s^2)$.

Experimental results show a significant reduction of clock skews ranging from 87 times to 144 times compared to the clock skews before applying the proposed algorithm. We also observe a further reduction of the propagation delay of clock signals as a result of applying the proposed skew sensitivity algorithm.

1 Introduction

The increase in complexity of synchronous ASICs has made the clock signal distribution a considerable limiting factor of overall performance. Clock skew contributes about 10% of clock rate in recent ASICs[1]. This number will increase as ASIC technology advances. Controlling clock signal becomes harder as circuits get faster, chips become larger, and minimum feature size is scaled down. Because variations in the parameters of clock buffers, interconnection parameter variations, and capacitive loading variations contribute to clock skew, these parameters must be considered in calculating the minimum effective clock skew. If these manufacturing variations are not considered, zero skew in theory results in a considerable skew in practice and thus affects the cycle time of the system.

A number of research has been done to minimize clock skew and delay. H-tree[1], MMM[6], and KCR[8] reduce and balance wire length.

Tsay[11] devised a zero-skew merging scheme using Elmore delay model. Chao et al.[3] proposed a two-phase algorithm which enhanced Tsay's work. DME[2] builds the clock tree in a bottom up fashion.

Recently Pullela[8] proposed to insert buffers by using exhaustive search. We propose an optimal buffering mechanism using dynamic programming approach which drastically reduce the complexity of the operation. The method can reduce the total propagation delay significantly while preserving the zero-skew property. For a given clock tree of n points and a library of s different buffer sizes, the run time of the presented algorithm is $O(log^3 n \cdot s^2)$. The remainder of the paper is organized as follows. We formulate the zero-skew buffering problem in the next section. Then we describe the optimal buffer synthesis algorithm. The last section contains our experimental results and comparison with previous works.

2 Problem Formulation

There are a number of circuit parameters that have effects on the clock skew. Some can be controlled by designers and some are under little designer control. Parameters such as process, voltage, and temperature vary due to the fluctuations of the manufacturing process. We assume that wire width and buffer size are parameters designers can control.

Given wire width, w, and buffer size, v, as control parameters, we define the skew sensitivity, SS, as the maximum difference between skews under varying values of w and v due to process variations. Hence the goal of skew sensitivity minimization is to find the optimum w and v that achieve minimum SS. Intuitively, simply increasing w and v seem to reduce the skew sensitivity. It is true to a certain point and the skew sensitivity decreases with larger w and b. However as the absolute value of propagation delay increases with larger w and v, their differences also increase and result in larger skews.

We assume the same size buffer on the same level of a tree. Also proposed algorithm searches finite number of sizes of buffers available in the given library. Thus we fix parameters of a minimum size buffer and assume an integer multiples of minimum buffer sizes. We start the algorithm from the given topology. This topology is generated by SASPO[4] algorithm, which is known to generate an efficient topology.

Based on above assumptions, we can formulate the skew sensitivity minimization (SSM) problem formally as follows.

Problem SSM: Given a clock tree *T(E,V)* with *n* leaf nodes and a library of buffers with *s* different size buffers, find an optimum level of buffers with proper sizes and wire widths that minimizes skew sensitivity.

3 Skew sensitivity minimization

3.1 Wire width variance

We can derive an equation of propagation delay as a function of wire width. In Figure 1 we calculate the propagation delay between two points *S* and *L*. The clock driver is driven by a source resistance of R_s and the load has capacitance C_L. The wire between two points have resistance value of r_w and c_w and modeled by π equivalent circuit.

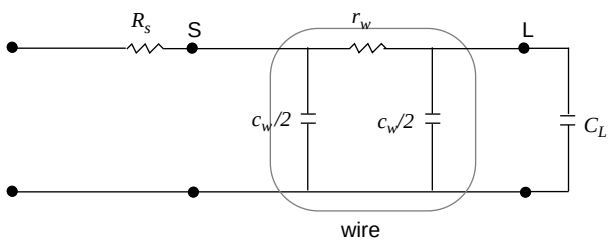

Figure 1. Wire width optimization

Let *w* be the width and *l* be the length of the wire section between *S* and *L*. Also let k_r, k_c, and k_f be the resistance coefficient, capacitance coefficient, and fringe capacitance coefficient respectively. Then the wiring capacitance and resistance is given as $c_w=(k_c \cdot w + k_f) \cdot l$ and $r_w = k_r \cdot l/w$ respectively. Equation (1) is the delay formula as a function of width and length of the interconnect wire section *S* and *L*. By setting $\partial t/\partial w$ equal to zero we get equation (2), which gives us the optimum wire width for delay sensitivity under wire width variation.

$$t = R_s (k_c \cdot w \cdot l + k_f \cdot l + C_L) + \frac{k_r}{w} \cdot l \left(\frac{k_c \cdot w \cdot l + k_f \cdot l}{2} + C_L\right) \quad (1)$$

$$w = \sqrt{\frac{k_r \cdot (C_L + k_f \cdot l/2)}{k_c \cdot R_s}} \quad (2)$$

Observation of equation (2) indicates that the optimum wire width is a function of input resistance and load capacitance along with k_r and k_c, and does not depend on the length of the wire section between the source and load if k_f=0. Also according to equation (2), because load capacitance is large at the higher level of the clock tree, wire widths at the top level is wider and becomes narrower as the level goes down the clock tree. This is desirable since clock skew is greatly affected by the width variation at the top level.

3.2 Buffer size variance

Wire width is not the only parameters in manufacturing variations. Buffer variance must also be considered to reflect true manufacturing variations. In general, the value of buffer length is the minimum achievable and the absolute value of CMOS buffer size can vary by 10% to 20% of minimum width. Thus, we add 15% to each buffer width in one path from the root and subtract 15% to each buffer width in another path from the root. This gives us the worst case simulation of the buffer width variation during manufacturing process.

To analyze the effect of the buffer size on delay, the input capacitance and output resistance are represented in terms of buffer size v_1 for *buffer 1* and v_2 for *buffer 2*.

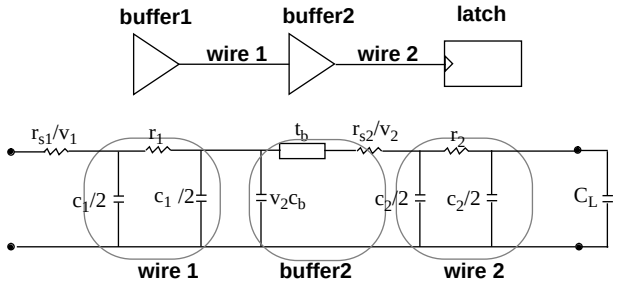

Figure 2. Buffered clock model

The propagation delay from buffer 1 to latch in Figure 2 is given by equation (3). Total delay *t* is divided into $delay(l_1,v_1,l_2,v_2)$ in (4) and $delay(l_2,v_2)$ in (5). $delay(l_1,v_1,l_2,v_2)$ represents a delay from *buffer 1* to *buffer 2* and $delay(l_2,v_2)$ represents a delay below *buffer 2*, which is shown to be independent of the size of *buffer 1* (v_1) in (5).

$$t = (r_{s1}/v_1)(c_1 + v_2 c_b) + r_1 \cdot (c_1/2 + v_2 c_b) + t_b + (r_{s2}/v_2)(c_2 + C_L) + r_2 \cdot (c_2/2 + C_L) \quad (3)$$

$$delay(l_1, v_1, l_2, v_2) = (r_{s1}/v_1)(c_1 + v_2 c_b) + r_1 (c_1/2 + v_2 c_b) \quad (4)$$

$$delay(l_2, v_2) = t_b + (r_{s2}/v_2)(c_2 + C_L) + r_2 \cdot (c_2/2 + C_L) \quad (5)$$

4 Skew sensitivity minimization algorithm

In the skew sensitivity minimization (SSM) algorithm, we maintain a matrix *B*[*b*,*l*,*s*], which represents the minimum skew with *b* buffer levels, the highest buffer locating at level *l* and buffer size *s*. The algorithm inputs the clock topology from the given circuit. *BufferInsert*() is the main function which inserts buffers of optimum sizes to minimize the clock skew sensitivity. After the completion of *BufferInsert*(), the content of lookup table *B*[*b*,*l*,*s*] will be complete and we can retrieve all the buffer parameters which yield the minimum skew sensitivity. As a last step, fine tuning by *BufferReposition*() repositions buffers (see section 4.3) to minimize wire length while maintaining the minimum skew property.

BufferInsert() calls *ConstructTable*(*b*,*l*,*s*) to build the

lookup table *B*[*b*,*l*,*s*] for all *b, l,* and *s*. For each iteration, the algorithm reads the lookup table and finds a combination of buffer location and buffer size that yields the smallest skews. The searching can be written as:

$$B[b, l, s] = min[MinSkew(l, s, l', s') + B[b-1, l', s']] \quad (6),$$

where *MinSkew*() calculates the skew sensitivity (*SS)* from level *l* to level *l'*. To compute *B*[*b, l, s*] we need to know the values of *MinSkew*(*l,s,l',s'*) and *B*[*b-1,l',s'*]. While calculating the partial *SS*, the algorithm performs the *WireSizing*() for optimum wire size between level *l* and level *l'* as outlined in section 3. *MinSkew*() then selects two paths in the clock tree and returns the calculated partial *SS* from level *l* to level *l'*. Two paths *path 1* and *path 2* are set by varying uniform wire widths by w_δ and by varying fixed buffer size by s_δ on two different paths in the given clock tree.

Algorithm SSM

ClockTree = ReadTopology();
BufferInsert(*ClockTree, BufferLibrary*);
Select the best result;
BufferReposition(*ClockTree*);
print *minimum delay, buffer levels*, and *buffer sizes*;

Procedure BufferInsert(*ClockTree, BufferLibrary*)
 for *b* := 1 to *maxbuffer* **do**
  for *l* := 1 to *maxlevel* do
   for *s* := 1 to *maxsize* do
    B[*b, l, s*] = ConstructTable(*b, l, s*);
 return B[*b, l, s*];

Procedure ConstructTable(*b, l, s*)
for *l'* := 1 to *l*-1 do
 for *s'* := 1 to *s* do
  SS = min[MinSkew(*l, s, l', s'*) + B[*b-1, l', s'*]];
  keep minimum *SS* result;
return *SS*;

Procedure MinSkew(*l, s, l', s'*)
inset buffer at *l* with size *s*;
insert buffers at *l'* with size *s'*;
WireSize(*l, l'*);
call SetPath(*l, l'*) to set *path 1* and *path 2*;
partial SS = CalcSkew(*path1, path2, l, l'*);
return *partial SS*;

4.1 Run time analysis

The run time of our optimum buffer synthesis algorithm is dominated by for loops. Since there are two loops for number of buffers and levels in procedure Buffer, it takes $O(log^2 n)$ time. Two loops for different buffer sizes require $O(s^2)$. And wire sizing operation requires *O(logn)* time. Together, the total run time is $O(log^3 n \cdot s^2)$.

We now show that the complexity of an exhaustive approach is computationally expensive. Since *n* is the number of leaf nodes, *logn* is the number of levels in *T(E,V)*. Trying all different buffer sizes, including no buffer, for each level requires $O((s+1)^{logn})$ time. This can be rewritten as $O(n^{log(s+1)})$, and this is a polynomial time complexity with high exponent. For example when *logn*=12 and the assumed value of *s*=10, the ratio of run time of an exhaustive approach and our proposed dynamic programming based algorithm is $1.8 \cdot 10^7$.

4.2 Optimality of SSM

In general to be able to use dynamic programming algorithm, the problem must satisfy two properties [5].
i) *optimal substructure*, an optimal solution to the problem contains within it optimal solutions to subproblems.
ii)*overlapping subproblems*, recursive or exhaustive algorithm will visit the same problem many times.

Lemma 1 SSM has the *optimal substructure* property.
In Figure 2, let t_{sl} denote the delay from the clock source to latch. Similarly t_{sb} is the delay from the source to the buffer and t_{bl} is the delay from the buffer to latch. We know $t_{sl} = t_{sb} + t_{bl}$. Due to the *isolation property* of the buffer, size and location of the buffer which yields minimum t_{bl} remains same regardless of the selection of the buffer at the higher level. After inserting a buffer between wire1 and wire2, the delay t_{sb} effectively becomes isolated by the buffer input capacitance, C_b.

Lemma 2 SSM has the *overlapping subproblems* property.
As shown in section 4.1, an exhaustive method requires an exponential run time when we consider clock tree level as an input. According to the recurrence relation in equation (6), to compute *B*[*b,l,s*], we only need to know *B*[*b-1, l, s*] and *MinSkew*(*l, s, l', s'*).

As a consequence of *Lemma 1* and *Lemma 2*, we can state the following theorem.
Theorem Dynamic programming algorithm to the SSM problem achieves an optimal solution.

4.3 Buffer repositioning

During the merging stage, often times an excessive detouring is required to maintain minimum skew sensitivity. To avoid an excessive detouring, we adjust the location of buffers along its routing path.

In Figure 3, t_i is the delay of subtree *i* and c_i is the capacitance of subtree *i*. The delay after the buffer is added is calculated based on a formula which takes into account of the buffer input capacitance.

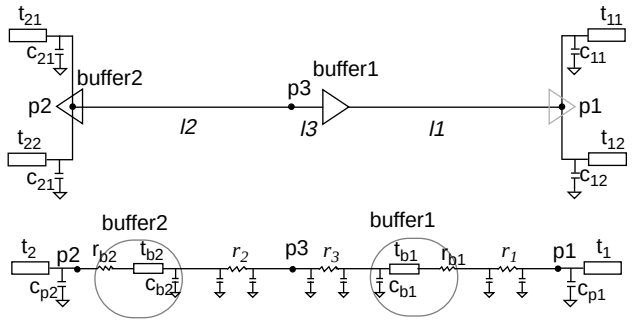

Figure 3. Buffer repositioning

For example, delays seen at point 3 in Figure 3 is given by,

$$t_{left} = r_2(c_2/2 + c_{b2}) + r_{b2} \cdot C_{p2} + (t_{b2} + t_2) \quad (7)$$

$$t_{right} = r_3(c_3/2 + C_{b1}) + r_1(c_1/2 + C_{p1}) + r_{b1}(c_1 + C_{p1}) + (t_{b1} + t_1) \quad (8),$$

where t_{left} is the delay from p3 to p2 and t_{right} is the delay from p3 to p1. In equation (8) we can observe that $r_1 \cdot C_{p1}$ is a new term by shifting buffer1 from its original position p1 to left. By putting equations (7) and (8) equal, we solve the tapping point without introducing excessive detouring of wires.

5 Experimental results

Proposed algorithm, skew sensitivity minimization (SSM), is implemented and tested on five benchmark circuits. The buffer parameters are derived from [10] and based on the 0.5 micron CMOS technology. The output resistance is 3170 Ω, input capacitance is 10fF, and the buffer internal delay is 35.5 ps for 1X buffer. We assume up to 10 different buffers are available in the library (*s*=10).Buffer parameters used in Pullela [8] were not available [9] and we could not compare with SSM. Table 1 compares the skew values under various conditions.

Table 1: Skew sensitivity comparison.

benchmarks (# of pins)	Skew before SSM (sec)	Skew after SSM (sec)	Ratio
r1 (267)	$5.98 \cdot 10^{-11}$	$5.47 \cdot 10^{-13}$	109.32
r2 (598)	$9.71 \cdot 10^{-11}$	$9.91 \cdot 10^{-13}$	97.98
r3 (862)	$1.60 \cdot 10^{-10}$	$1.72 \cdot 10^{-12}$	93.02
r4 (1903)	$2.28 \cdot 10^{-10}$	$2.60 \cdot 10^{-12}$	87.69
r5 (3101)	$1.75 \cdot 10^{-9}$	$1.21 \cdot 10^{-11}$	144.63

The values in the second column, skew before SSM, are measured by varying uniform wire widths by w_δ and by varying fixed buffer size by s_δ on two different paths. We used a value of w_δ to be 15% of the unit wire width and s_δ to be 15% of the unit buffer size to simulate the worst case manufacturing variations. The third column values are measured by the same manufacturing variation simulation after applying an SSM algorithm. Compared to the second column, the skew drops from 87 times (*r4* circuit) to 144 times (*r5* circuit).

Table 2: Propagation delay comparison.

benchmarks (# of pins)	SASPO (ns)	SSM (ns)	ratio over SASPO
r1 (267)	1.119	0.481	2.3
r2 (598)	3.807	0.667	5.7
r3 (862)	3.861	0.685	5.6
r4 (1903)	11.934	1.042	11.5
r5 (3101)	18.457	1.593	11.6

Table 2 lists the propagation delays from SASPO [4] and SSM. Compared to SASPO[4], which does not insert any buffers, we see a dramatic reduction of propagation delay for all test cases. This version of SASPO aims at reducing delays and for a fair comparison, we assumed the same technology parameters. Over 11 times of reduction is reported in *r4* and *r5* test cases. It is obvious that the proposed skew sensitivity minimization algorithm not only helps reduce the skew sensitivity, but also greatly reduces the propagation delay.

References

[1] H. Bakoglu, Circuits, Interconnections and Packaging for VLSI. Addison-Wesley, 1990.

[2] K. D. Boese and A. B. Kahng, “Zero-Skew Clock Routing Trees with Minimum Wire-length,” Proc. 5th IEEE Intl. Conf on ASIC, NY, pp. 17 - 21, 1992.

[3] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, “Zero-Skew Clock Net Routing,” Proc. 29th ACM/IEEE Design Automation Conf., pp. 518-523, 1992.

[4] N.-C. Chou, C.-K. Cheng, “Wire Length and Delay Minimization in General Clock Net Routing,” Proc. IEEE Intl. Conf. on Computer-Aided Design, pp. 552-555, 1993.

[5] T. Cormen, C. Leiserson, and R. Rivest, Introduction to Algorithms. MIT Press, 1990.

[6] M. A. B. Jackson, A. Srinivan, and E. S. Kuh. “Clock Routing for high performance ics.” Proc. Design Automation Conferences, pp. 573-579, 1990

[7] A. Kahng, J. Cong, G. Robins. “High-Performance Clock Routing Based on Recursive Geometric Matching,” Proceedings of Design Automation Conferences, pp. 322-327, 1991

[8] S. Pullela, N. Menezes, J. Omar, L. Pillage, “Skew and Delay Optimization for Reliable Buffered Clock Trees”, Proc. IEEE Intl. Conf. on Computer-Aided Design, pp. 556-562, 1993.

[9] S. Pullela, L. Pillage, Personal communication.

[10] T. Sakurai, “A Unified Theory for Mixed CMOS/ BiCMOS Buffer Optimization,” IEEE Journal of Solid-State Circuits, vol. 27, no. 7, July 1992.

[11] R.-S. Tsay, “Exact Zero Skew,” Proc. IEEE Intl. Conf. on Computer Aided Design, pp. 336-339, 1991.