Process-Variation-Tolerant Clock Skew Minimization

Shen Lin and C. K. Wong
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

In this paper, we propose a novel hierarchical multiple-merge zero skew clock routing algorithm. The routing results produced by our approach will have zero skew in the nominal case and minimal skew increase in the presence of worst process variations. In order to construct such a clock routing, we formulate the linear placement with maximum spread problem and provide an $O(n \min\{n, P\} \log n \log P)$ algorithm for optimally solving this problem, where n is the number of cells to be placed and P is the maximum spread. Experimental results show that our algorithm can indeed reduce the skew in various manufacturing variations effectively.

1 Introduction.

Skew is defined as the maximum difference among the delays from the clock source to clock pins (destinations). In synchronous circuit design, skew sets the lower bound of clock cycle time; therefore, minimizing clock skew is a very important problem in the design of high performance VLSI systems. The zero skew clock routing problem is to construct a routing to connect all clock pins with the same delay from the clock source to each clock pin. Besides the clock pin locations, the capacitive loading of each clock pin, and the per unit wire length resistance (R) and capacitance (C) parameters are also given as inputs for estimating delays.

Many heuristics for clock routing have been proposed in the past. H-tree structures [7] [6] are the most widely used. A generalization of an H-tree that hierarchically connects the median points is proposed in [8]. These H-tree approaches will have difficulty handling the design with clock pins unevenly distributed. To cope with this difficulty, a bottom-up pairwise merge approach [5] and a top-down max-min approach [4] have been proposed. However, these heuristics focus only on wire length balancing, rather than the real objective of balancing clock delay.

Ren-Song Tsay first proposed a bottom-up binary-merge zero skew routing algorithm [1, 2]. His approach recursively merges two zero skew subtrees to form a bigger zero skew subtree until the resulting zero skew subtree contains all clock pins as its leaves. The root of this tree also determines the location of the clock source. Fig. 1 shows an example. A, B, C are three clock pins to be routed. At the first step, A and B are merged. Based on the Elmore RC delay model, we can determine a tapping point D giving the same delay from D to A and from D to B. At the next step, the subtree containing the single pin C and the subtree A-D-B are merged as shown in the figure. The tapping point E giving the same delay from E to C and from E to D then A (or B) is the location of the clock source. Later, the work of [3] was proposed to improve the merging sequence in order to minimize the total wire length.

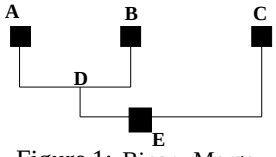

Figure 1: Binary Merge.

However, there are two drawbacks in this type of approach:

1. It is vulnerable to process variation. The routing result produced by [1] is guided by estimated delays, which depend on the R C parameters and the capacitive loading of each clock pin. During the chip fabrication, any change of these parameters will cause severe skew increase. Take Fig. 1 as an example. Assume the actual R is larger than expected. Since the wire length from E to C is longer than that from E to A, the increase in delay at pin C will be larger, resulting in nonzero skew. However, this kind of process variation frequently occurs because those parameters are based solely on designers' prediction.
2. It lacks in design flexibility. If designers want to modify the location of any clock pin, the entire clock routing needs to be redone to avoid skew. The effect of modification is not local and will change the delays to most of the clock pins. However, modifications commonly occur because designers cannot know the exact clock pin locations until the very last stage of physical design process, yet clock routing cannot wait until then.

As digital VLSI technology and design techniques have advanced in recent years, the system clock cycle has been reduced to the range of a couple of nano-seconds. A process-variation clock skew of a few hundred pico-seconds will make these state-of-the-art designs fail. The process-variation tolerant clock design, therefore, becomes more and more important. Empirically in our advanced system design, the process-variation skew can be as large as 10% of the clock delay. In order to minimize the worst case skew,

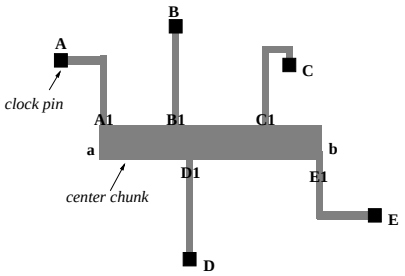

Figure 2: Center Tree.

DEC's Alpha processor would rather dedicate almost a whole metal plane for the clock network and one-fifth of the chip area for the clock driver [10]. However, this approach makes the Alpha chip consume too much power (about one half the power is actually consumed by the clock network).

The zero skew algorithm introduced in this paper is, therefore, aiming at overcoming the two drawbacks mentioned above, while avoiding overkilling the problem. We propose an hierarchical two-stage multiple-merge approach, which will be discussed in the next section. Our algorithm may use more wiring resources than the binary merge approach mentioned above but the increase is insignificant, unlike the Alpha design.

2 Our Approach.

Unlike the binary merge of [1], we merge multiple pins (e.g. 15 or 16 pins) at one time to form a zero skew subrouting (which is not a tree and will be discussed later). The routing process is hierarchical. At the bottom level, we identify the group of clock pins to be merged together and merge them using our two-stage multiple merge algorithm. For example, the clock pins on a macro boundary or a set of clock pins which are close to each other can be merged together. Recursively, at a higher level of the hierarchy, we merge those closer roots of the subroutings constructed one level below until the resulting subrouting covers all the clock pins.

Each multiple merge is composed of two stages: at the first stage, each clock pin is connected to a center chunk separately with the same delay as shown in Fig. 2, and at the second stage, N points of the chunk are routed using a balanced length binary tree (to be defined in section 4), where N is a power of two. Fig. 3 shows a balanced length binary tree with N being four. Fig. 4 shows the result combining these two stages. Separate layers are used for x and y routing. We call the first stage routing the *center tree routing*.

The center chunk is a fat wire. The determination of its length, width, and location will be discussed in the next section and so is that of the number N. We assume this center chunk has zero skew everywhere and will discuss how to accomplish this later. Therefore, if we let the interconnect from each clock pin to the center chunk have the same RC delay based on the Elmore model, the clock signal will arrive at each clock pin at the same time after the signal goes to the center chunk. Fig. 2 shows five clock pins A, B, C, D and E, connected to the center chunk a-b. The delays of A-A1, B-B1, C-C1, D-D1, and E-E1 are the same. A1, B1, C1, D1, and E1 are called the *branching points* of the center chunk for each clock pin. The routing of this center tree will be introduced in the next section. This first stage routing shields the effect of any local modification from affecting the rest of the zero skew routing. If later, designers want to move the clock pin A to the right, they can just shift A-A1 to the right. Or, later, if the capacitive loading of B is different from expected, only the delay of B-B1 will be different but not the rest. Furthermore, since the connection from each pin to the center is short, only minimal skew will be introduced even when the R C parameters vary.

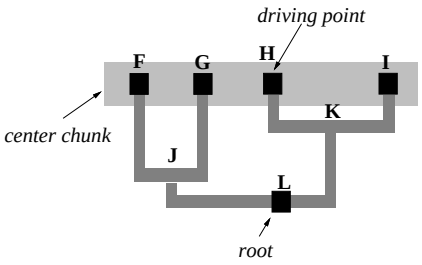

Figure 3: Balanced Length Binary Tree.

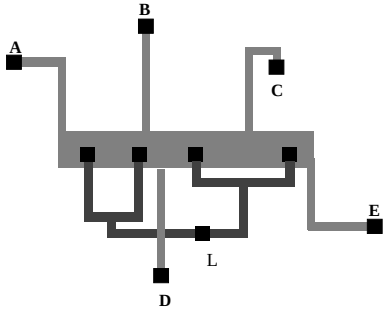

Figure 4: Multiple Merge.

At the second stage routing, we let the clock signal arrive at N different points (*the driving points*) of the center chunk simultaneously to ensure that the center chunk has zero skew everywhere. As shown in Fig. 3, F, G, H and I are the four driving points of the center chunk. Since we route them using a balanced length binary tree, the wire length of F-J is equal to that of G-J, H-K, or, I-K; and the wire length of J-L is equal to that of K-L. Therefore, the clock signal will arrive at F, G, H and I at the same time. Moreover, the clock signal will arrive at F, G, H and I at the same time even when the R C parameters vary, because the same amount of delay change will be introduced to F-J, G-J, H-K, and I-K; and the same amount of delay change will be introduced to J-L and K-L. Hence, our approach is much less vulnerable to process variation. The determination of the number N and the locations of those N driving points will be given in the next section.

3 Center Tree Construction.

In this section, the electrical backgrounds making our zero-skew-chunk assumption valid and the center tree routing

algorithm will be presented. It is the wire resistance that causes skew on a wire segment. If the wire is a perfect conductor, voltage will rise at the same speed everywhere on the wire. Since the chunk is a fat metal wire, the resistance is small. The width of the wire is determined to give the RC delay between adjacent two driving points less than a predetermined threshold value. Secondly, the clock signal comes to this fat wire from N different places simultaneously, hence shortening the signal propagation latency. Besides these, there are another two supporting factors. They are (1) the locations of the N driving points are determined so that each driving point needs to drive the same amount of capacitive load; (2) the branching points (e.g. A1, B1, C1, D1 and E1 of Fig. 2) are located on the center chunk as evenly distributed as possible. For the first factor, we assume each driving point needs to charge up the capacitive load of half the distance to its neighbors. For example, in Fig 3, F is supposed to charge up all the capacitive loads from the left end of the chunk to the middle point of F-G. G is supposed to charge up the loads between the middle point of F-G and the middle point of G-H.

In order to let each driving point drive the same amount of capacitive load, we add up all the capacitive load at the chunk, which includes the load of the chunk itself, that of each connection from a pin to the chunk, and that of each pin. Let us denote the total amount by C_L. Then, starting from one end of the chunk to the other, we scan through every position of the chunk. The N driving points will be located at the positions where the capacitive load sums up to $\frac{1}{2N}C_L$, $\frac{3}{2N}C_L$, $\frac{5}{2N}C_L$, $\cdots$ and $\frac{2N-1}{2N}C_L$. Therefore, each driving point drives one Nth of C_L.

The center tree is constructed in the following steps.

1. To keep the interconnect from each clock pin to its branching point short, the center chunk is routed along the larger dimension of the enclosing box of all the clock pins. For example, let the enclosing box have the x dimension L and the y dimension W. If L is larger, then the center chunk is routed in x direction. To illustrate our discussion, in the steps below we assume that the chunk is routed in x direction as shown in Fig. 2. Only axis swap is needed for the discussion if the chunk is routed in y direction.
2. Pick the clock pin with the largest y coordinate (e.g. B of Fig. 2) and the one with the smallest y coordinate (e.g. D of Fig. 2). The center chunk will be located in between these two pins. The location is determined to give the same RC delays from the center chunk to the two pins. Let us denote this delay by Δ.
3. For the remaining pins to have this delay Δ to the center chunk, snakings are necessary (e.g. A, C and E of Fig. 2). The amount of snaking for pin i is given by

$$S_i = \sqrt{\frac{cl_i}{C} * \frac{cl_i}{C} + 2 * \frac{\Delta}{RC}} - \frac{cl_i}{C} - |y_i - y_c|, \quad (1)$$

where cl_i is pin i's capacitance and y_i and y_c are the y coordinates of pin i and the center chunk, respectively. Therefore, pin i's branching point on the center chunk can have the x coordinate anywhere in the interval $[l_i, u_i]$, where $l_i = x_i - S_i$ and $u_i = x_i + S_i$. $[l_i, u_i]$ is defined as the *permissible interval* of pin i's branching point.

4. In order to minimize the signal skew on the center chunk, its length should be as short as possible. By exploiting branching points' permissible intervals, the left boundary coordinate L_c (e.g. A1 of Fig. 2) of the chunk and the right boundary R_c (e.g. E1 of Fig. 2) can be determined by

$$L_c = \min_i(u_i), \qquad R_c = \max_i(l_i) \quad \forall i. \quad (2)$$

A1 and E1 of Fig. 2 are the boundary points determining L_c and R_c, respectively.

5. $R_c - L_c$ gives the length of the center chunk. The width of the center chunk is determined to give the RC delay between any two adjacent driving points less than a predetermined threshold value (3 pico-seconds in our algorithm). If the chunk is very long, there is a trade-off between choosing large N or increasing the width of the chunk to save the wiring resources. Since practically N is 4, 8 or 16, it does not take too much computation to figure out the best choice.
6. The permissible intervals of the branching points need to be updated to

$$l_i = max(L_c, l_i), \quad u_i = min(R_c, u_i) \ \ \forall i. \quad (3)$$

We can exploit the permissible interval to place the branching points on the center chunk as evenly distributed as possible. The situation with many branching points placed in a small interval of the center chunk should be avoided. The distance between adjacent points should be maximized. We introduce the *linear placement with maximum spread* problem to determine the location of each branching point x_i.

Definition: The linear placement with maximum spread problem is defined as

$$MAXIMIZE \quad \min_{ij}(|x_i - x_j|) \ \ 1 \le i \ne j \le n$$
$$\text{s.t.} \quad l_i \le x_i \le u_i \quad i = 1...n, \quad (4)$$

where l_i, x_i, and u_i are integers and the objective function $\min_{ij}(|x_i - x_j|)$ is defined as the $spread$.

We have developed an $O(n \min\{n, P\} \log n \log P)$ algorithm for solving this problem optimally, where n is the number of branching points and P is the maximum spread. This algorithm will be introduced in section 6.

Therefore, we have constructed the center tree, and are now ready to construct the balanced length binary tree.

4 Balanced Length Binary Tree Construction.

We consider a balanced binary tree with $N = 2^M$ leaves and height M. We can assign levels to the nodes in the tree, letting the leaves be the M-th level nodes and the root of the binary tree the 0-th level node. And, let the immediate children of the i-th level nodes be the $(i+1)$-st level nodes. A balanced length binary tree is defined as follows.

Definition: A balanced length binary tree is a balanced binary tree that has the edges from the i-th level nodes to the $(i+1)$-st level nodes of the same length for $i = 0...M-1$.

For example, in Fig. 3 the length of J-L is equal to that of K-L and the lengths of F-J, G-J, H-K and I-K are the same.

In the previous section, we have determined N and the locations of the N driving points, which are now the N leaves of the balanced length binary tree to be constructed. There are many solutions for a balanced length binary tree connecting these leaves. The desired one is the one with the minimal wire length. When we make connections from the $(i+1)$-st level nodes to their i-th level parents, one half of the distance between the two farthest-apart children of the same parent sets the lower bound for each edge's length. For example, in Fig. 3 the points H and I are farther apart from each other than the points F and G, then one half of the distance between H and I sets the lower bound for the edge F-J and G-J. However, H-K is larger than this lower bound because a connection between H and K of the length equal to this lower bound will overlap with the center chunk. Therefore, the actual edge length is the lower bound added by a small number, e.g. a safe pitch. After determining the edge length, we can uniquely determine the i-th level nodes' locations. This process is continued recursively to the root.

5 Moving Up the Hierarchy.

A buffer is placed at the root of the balanced length binary tree to repower the resulting zero skew subrouting of Fig. 4 and to block the capacitive loads from affecting the upper levels of the hierarchy. In the merge of one level up, we will merge those buffers together as if they were the clock pins at the bottom level. At the bottom level, the clock pins do not have any delay but now at this level, there is delay associated with each buffer, namely, the delay from the buffer input to any clock pin in its own subrouting (the delay to any clock pin is the same since the skew is zero). This delay can be determined through circuit simulation. Therefore, in the merge of one level up, not only the capacitive loading of each pin is different but also its delay. Based on the Elmore delay model, we can use different wire length and wire width to give the same delay from the center chunk of this level to the clock pins of the bottom level. Then, the balanced length binary tree routing and the center tree routing are performed in the same way as described above. This recursion can be continued until all the clock pins in the chip have been covered. In practice, two levels will be sufficient. The bottom level connects the clock pins within a macro and the top level connects all the bottom level subroutings.

6 Linear Placement with Maximum Spread.

We transform the problem, defined in Eq.(4), into a series of *single processor scheduling problems*. The single process scheduling problem is defined as follows.

Definition: We are given a single processor and a set S of n jobs. For each job X there is a release time r_X and a deadline d_X, with r_X and d_X nonnegative integers. A schedule is *feasible* if there is no time at which more than one job is being run and if every job in the schedule is begun no earlier than its release time and is completed by its deadline. The single processor scheduling problem is to find a feasible schedule if one exists in which each job is run for the same amount of time p or to determine that none exists.

Lemma: The single processor scheduling problem can be solved in $O(mn\log n)$ time, where $m = min\{n, p\}$.

The proof of this lemma and the corresponding algorithm are given in [9].

Theorem 1: Given a linear placement with maximum spread problem, we create a set of single processor scheduling problems with n jobs by letting $r_i = l_i$ and $d_i = u_i + p$, where p is a nonnegative integer and is assigned to be each job's execution time. If P is the maximum spread of the linear placement with maximum spread problem, then the subset of single processor scheduling problems with $p \leq P$ are feasible and the subset of single processor scheduling problems with $p > P$ are infeasible.

Proof: It is clear that for each problem of the feasible subset, $l_i \leq x_i \leq u_i$, where x_i is the starting time for job i because $x_i + p \leq d_i = u_i + p$. We are going to first prove that if the single processor scheduling problem thus created with $p = \rho$ is feasible for some integer ρ, then the single processor scheduling problems thus created with $p < \rho$ are also feasible. Let each job of the problem with $p = \rho$ start at x_i. Without loss of generality, we assume $x_1 < x_2 < x_3 < \cdots < x_n$. Then, x_i can also be a feasible schedule for job i for the problems with $p < \rho$ because 1) $x_i + p \leq u_i + p$ and 2) $x_i + p < x_i + \rho \leq x_{i+1}$, for$i = 1...n-1$.

Secondly, we are going to prove that the scheduling problem thus created with $p = P$ is feasible. Let x_i be the position of cell i in the placement. Again, without loss of generality, we assume $x_1 < x_2 < x_3 < \cdots < x_n$. Then, $x_i + P \leq x_{i+1}, i = 1...n-1$. Therefore, x_i can be a feasible schedule for the scheduling problem with $p = P$.

Finally, we are going to prove that the scheduling problem thus created with $p = P + 1$ is not feasible by employing contradiction. Suppose it is feasible and let x_i be the start time of job i. Without loss of generality, we assume $x_1 < x_2 < x_3 < \cdots < x_n$. Hence, $x_i + P + 1 \leq x_{i+1}$, which along with the fact that $l_i \leq x_i \leq u_i$ shows that the maximum spread should be $P + 1$ instead of P. It is a contradiction. We also know that there will be no feasible scheduling with $p > P + 1$ because if yes, then the scheduling with $p = P + 1$ would be feasible. Therefore, we have proved the theorem. □.

By employing theorem 1, we develop the following algorithm to solve for the linear placement with maximum spread problem. Starting from 1, we try different p in the power of 2 until the scheduling is infeasible, say this $p = 2^h$. Then, we can find the maximum spread by answering the scheduling problem and by using binary search of p in-between 2^{h-1} and 2^h. Therefore, we have

Theorem 2: The linear placement with maximum spread problem can be solved in $O(mn \log n \log P)$ time, where P is the maximum spread and $m = \min\{n, P\}$.

7 Experimental Results.

We implemented and tested our algorithm on an industry floating point unit with 18 clock pins. The routing result generated from our algorithm is shown in Fig. 5. Each unfilled square in the figure is a clock pin. The filled square is the root of this zero skew subrouting. This root is driven by a CMOS inverter, not shown in the figure, occupying $550\mu^2$ area . The balanced length binary tree rooted at the filled square has eight leaves, providing eight driving points to the center chunk. The wire of the tree is 2.7μ wide. The center chunk is 6.3μ wide and about $7mm$ long. The connection from the center chunk to each pin is 0.9μ wide. This clock network was simulated on the IBM ASTAP circuit simulator. We observed 3 pico-second (ps) skew out of 370 ps clock delay.

To model the worst process variation situation, we performed the following three experiments. In the first case, R, C and the pin capacitances all all increased by 25%. The skew is 6 ps out of 520 ps delay. In the second, R, C and the pin capacitances are all decreased by 25%. The skew is 4 ps out of 270 ps delay. In the third, the leftmost driving point (point a in Fig. 5) is disconnected due to possible metal-migration. The skew is 8 ps out of 380 ps delay.

This floating point unit was also routed by using the Zero Skew apporach of [2]. In the nominal case, the skew is 4 ps out of 456 ps delay. We analyzed the first two cases on that routing result. The skew increases to 16 ps (out of 600 ps delay) and 12ps (out of 322 ps delay), respectively. These results show that our algorithm indeed achieves better process-variation tolerance.

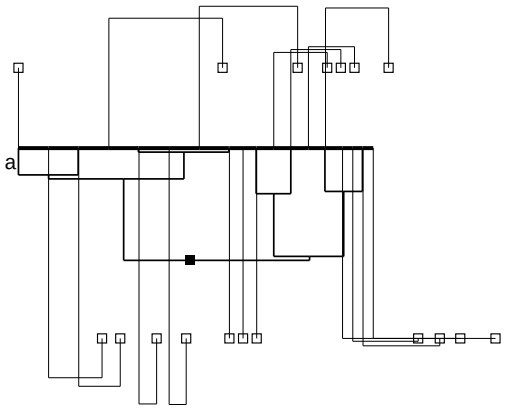

Figure 5: Clock routing for an industry floating point unit. (In reality, the vertical thin and thick lines do not overlap. They appear so due to lack of resolution in the figure.)

8 Conclusions.

As clock skew requirement gets more and more stringent in high performance chip designs, the process-variation tolerance becomes more and more important. In this paper, we have presented a novel process-variation-tolerant zero skew clock routing algorithm. The experimental results show its being very effective. We expect this clock routing algorithm to be widely used to enhance the performance of synchronous VLSI digital systems.

References

[1] Ren-Song Tsay, "Exact Zero Skew," *Proc. ICCAD*, pp. 336-339, Nov. 1991.

[2] Ren-Song Tsay, "An Exact Zero-Skew Clock Routing Algorithm," *IEEE Trans. on Computer Aided Design*, pp. 242-249, Feb. 1993.

[3] T. Chao, Y. Hsu, J. Ho, K. Boose, and A. Kahng, "Zero Skew Clock Routing with Minimum Wirelength," *IEEE Trans. on Circuit and Systems*, pp. 799-814, Nov. 1992.

[4] Qing Zhu and Wayne Dai, "Perfect-balance Planar Clock Routing with Minimal Path-length," *Proc. ICCAD*, pp. 473-476, Nov. 1992.

[5] A. Kahng, J. Cong, and G. Robins, "High-Performance clock routing based on recursive geometric matching," *Proc. DAC*, pp. 322-327, June 1991.

[6] A. L. Fisher and H. T. Kung, "Synchronous large systolic arrays," *Proc. SPIE*, pp. 44-52, 1982.

[7] S. Dhar, M. A. Franklin and D. F. Wann, "Reduction of clock delays in VLSI structures," *Proc. ICCD*, pp. 778-783, 1984.

[8] M. A. B. Jackson, A. Srinivasan and E. S. Kuh, "Clock routing for high-performance IC's," *Proc. DAC*, pp. 573-579, June 1990.

[9] Barbara Simons, "A Fast Algorithm for Single Processor Scheduling," *19-th Annual Symp. on Foundations of Computer Science*, pp. 246-252, Oct. 1978.

[10] Daniel W. Dobberpuhl, et. al. "A 200-MHz 64-b Dual-Issue CMOS Microprocessor", *IEEE J. of Solid-State Circuits*, pp. 1555-1567, Nov. 1992.