Useful-Skew Clock Routing With Gate Sizing for Low Power Design

Joe G. Xi* Wayne W.-M. Dai
Computer Engineering
University of California, Santa Cruz

Abstract

Instead of zero-skew or assuming a fixed skew bound, we seek to produce useful skews in clock routing. This is motivated by the fact that *negative skew* may allow a larger timing budget for gate sizing. We construct a *useful-skew tree* (UST) such that the total clock and logic power(measured as a cost function) is minimized. Given a required clock period and feasible gate sizes, a set of negative and positive skew bounds are generated. The allowable skews within these bounds and feasible gate sizes form the feasible solution space of our problem. We use a *merging segment perturbation* procedure and a simulated annealing approach to explore various tree configurations. This is complemented by a bi-partitioning heuristic to generate appropriate connection topology and take advantage of useful skews. Experimental results have shown 11% to 22% total power reduction over previous methods of clock routing with zero-skew or single fixed skew bound and separately sizing logic gates.

1 Introduction

Carrying the heaviest load and switching at high frequency, clock distribution is a major source of power dissipation in synchronous digital systems. The switching logic gates contribute to the rest of the power dissipation. On the other hand, control of clock skew and critical path timing are also critical issues for high performance circuits.

There have been active research in the area of high-performance and low-power clock routing. First, *zero-skew tree* (ZST) clock routing was proposed[10, 1, 5]. Recently, it has been pointed out that it is almost impossible to achieve exact zero-skew in real designs[11]. In fact, it is neither necessary nor desirable to achieve zero-skew[6, 12]. For low power designs, *bounded-skew tree* (BST), rather than ZST has been proposed to reduce clock power[7, 3].

The BST algorithms assume a fixed non-zero skew bound. No indication was given as to how the bound is derived and what appropriate value should be assigned to the bound. Moreover, if we study clock skew more closely, we find (i) Because the logic delay varies from one block to another, the allowable skew for correct clock operation varies from one pair of clock sinks to another[6]. (ii) Skew could be either negative or positive with respect to the logic path direction. The allowable negative skew can be used to increase the effective clock period and therefore can be considered useful skew; (iii) The allowable skew bounds can be adjusted by adjusting the logic path delays, e.g. by gate sizing.

In the related area, *clock skew optimization* was proposed which uses allowable negative skews to improve synchronous circuit performance or reliability[6]. Later, *clock skew optimization* was incorporated in gate sizing[2] to take advantage of the larger timing budget. However, either arbitrary or bounded skew values are assumed and the cost and power of clock routing are overlooked. To produce negative skews, a common approach is to insert buffers as delay elements[2]. But this results in increased buffer power and process variation induced skew uncertainties[11].

In this paper, we formulate and solve the **Useful-Skew Clock Routing with Gate Sizing for Power Minimization** (UST) problem. Clock routing can take advantage of the allowable skew bounds while the *useful skews* can be used to allow a larger timing budget for gate sizing. We will show that this approach mitigates the unfavorable tradeoff of circuit speed and power. Savings on both logic power and clock routing cost can be achieved.

The rest of this paper is organized as follows. In section 2, we discuss the motivation of this work and give the formulation of the UST problem. In section 3, we present the routing algorithm and a topology generation heuristic. Experimental results are given in section 4.

2 Problem Formulation

Consider a simple synchronous circuit as shown in Figure 1. We assume positive edge-triggered flip-flops are used in this example and throughout this paper. Due to interconnect delays, skew may result between clock terminals such as C_{01} and C_{02} of flip-flops, FF_{01} and FF_{02}. Figure 2 illustrates the clock operations in two cases of skews. In both cases, skews are considered allowable if correct data are produced under the

*Affiliated with National Semiconductor Corp, CA.

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

given clock frequency. With excessive skew in either cases, incorrect operations may occur when data are produced either too early (known as *double-clocking*) or too late (known as *zero-clocking*) from FF_{01} to FF_{02}[6].

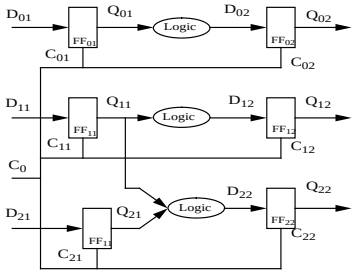


Figure 1: A synchronous circuit example.

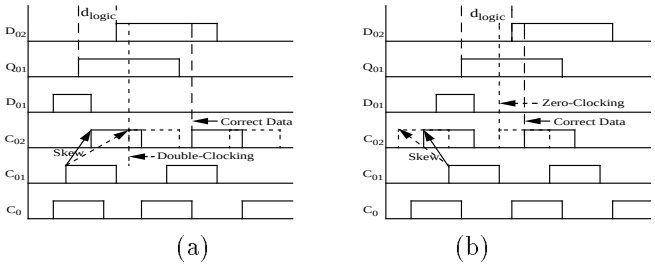


Figure 2: (a) negative skew; (b) positive skew.

In general, to ensure correct clock operation under a required clock period, P, the allowable clock skews between two adjacent flip-flops, FF_i and FF_j, are: To avoid *double-clocking* with *negative skew*, $d_i \leq d_j$:

$$d_j - d_i \leq MIN(d_{logic}) + d_{ff} - d_{hold} \qquad (2.1)$$

To avoid *zero-clocking* with *positive skew*, $d_i \geq d_j$:

$$d_i - d_j \leq P - (d_{ff} + MAX(d_{logic}) + d_{setup}) \qquad (2.2)$$

where d_i and d_j are the clock arrival times, $MAX(d_{logic})$ and $MIN(d_{logic})$ denote the longest and shortest path delays of the combinational block between FF_i and FF_j.

We notice the following properties of clock skew. First, both the negative and positive skew bounds vary from one pair of clock terminals to another since combinational logic path delays vary from one to another. To use a single fixed skew bound, one has to choose the smallest skew bound of all sink pairs, both negative and positive. Secondly, *negative skew* can be considered *useful skew* since it can allow circuits to run at a clock period less than the critical path delay[6]. Lastly, the skew bounds can be adjusted by sizing the logic gates. The *positive skew* bound can be enlarged by sizing the gates to reduce $MAX(d_{logic})$. The *negative skew* bound can be enlarged by sizing the gates to increase $MIN(d_{logic})$. Increasing the logic path delays can generally be done by reducing gate sizes which generally also reduce the dynamic power. This can be

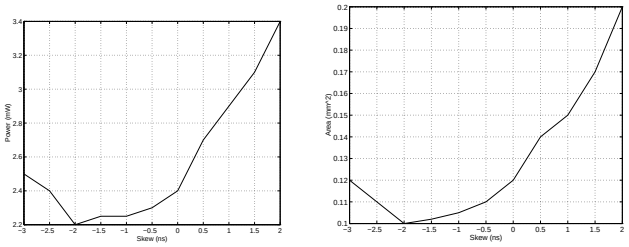


Figure 3: The minimum power and area vs. allowable skews within the negative and positive skew bounds for a combinational block between two flip-flops.

seen from Figure 3 which shows the relationship between minimum power, area of a combinational block and the allowable skew values.

These properties motivate us to consider the clock routing problem together with gate sizing. We realize that if *negative* skew and *positive* skew are treated the same, then a bounded-skew clock tree may produce skews that impose even tighter timing budget for gate sizing. Therefore, *negative* and *positive* skew bounds rather than a fixed unsigned skew bound should be considered in clock tree construction. With *negative skews* between certain clock sinks, e.g. the sinks that entail critical logic paths, a larger timing budget can allow gate sizing to further reduce logic power. We now define the **Useful-Skew Clock Routing with Gate Sizing for Power Minimization** (UST) problem: *Given a standard-cell based design and a library which consists of a set of gates and a set of feasible sizes(templates) for each gate, a required clock period, P, and a set of clock sink locations $S = \{s_1, s_2, \cdots, s_n\}$, the clock source s_0, we seek a tree topology G (a rooted binary tree with n leaves corresponding to the sinks in S), a clock tree T which embeds G, i.e. each internal node $v \in G$ is mapped to a location in the Manhattan plane, and a set of sizes $X^* = \{x_1^*, x_2^*, \cdots, x_m^*\}$ for all gates in the design. The objective is to minimize a cost function C given by:*

$$C(T, X) = \lambda L(T) + \gamma \Phi(X) \qquad (2.3)$$

where $L(T)$ is the total wire length of the T, $\Phi(X)$ is the total power of logic gates and λ and γ are weight coefficients. Meanwhile, the skew between any sink pair, s_i and s_j satisfies their corresponding negative skew bound, NSB_{ij} and positive skew bound, PSB_{ij}.

3 The UST Solution

3.1 Overview

Our useful-skew clock routing (UST) algorithm involves four tasks: (i) generating an appropriate clock tree topology; (ii) finding the locations of internal nodes of the tree; (iii) preserving the negative and positive skew bounds for correct clock operations under a given frequency and (iv) selecting the sizes of

logic gates for minimum power. Figure 4 gives a high level description of our algorithm. The main idea is as follows. First, in generating a topology, we try to maximize the the feasible regions for possible internal node locations. Then, in the process of embedding the topology, we explore feasible placements of internal clock tree nodes. This is equivalent to exploring allowable skews between various clock sinks. The resulting skews are used in gate sizing to determine the minimum logic power. The search of the internal node locations is inspired by the *Deferred-Merge Embedding* (DME) based algorithms[1, 7]. A ZST constructed by the DME algorithm[1] is used as the initial starting point. A simulated annealing approach is used to iteratively search for better placements of internal nodes and produce useful skews[8]. Because it is time prohibitive to perform gate sizing on the fly, we predetermine the gate sizing result of each combinational block for a known skew value. The cost function is updated in constant time at each iteration.

```
Input :
    S = set of clock sinks, n = |S|,
    P = the required clock period, P = 1/f,
    X^0 = the initial sizes of logic gates.
Output :
    a UST, T*, sizes of logic gates, X*.
PROCEDURE BuildUSTwithGateSizing (S, P, X^0) {
    X = X^0;
    G = GenerateTopology(S);
    T = BuildInitialZST(G,S); /* according to [1] */
    t = t_0;
    while (not Frozen) {
        while (not Equilibrium) {
            Pick a node, v;
            T' = PerformMSP(v, G, T);
            X' = GateSize(T', X, P); /* table look-up */
            ΔC = C(T', X') - C(T, X);
            If (ΔC ≤ 0 or e^{-ΔC/(k_B t)} ≥ random(0,1))
                T = T'; X = X';
        }
        t = δ(t) × t;
    }
    T* = T; X* = X;
}
```

Figure 4: High-level Description of UST Algorithm

3.2 Negative and Positive Skew Bounds

The negative and positive skew bounds define the feasible solution space. We say the clock sinks, s_i of flip-flop FF_i and s_j of flip-flop FF_j are *adjacent* if there exists a combinational logic path from FF_i to FF_j. Let d_i and d_j be the delays from clock source to sinks s_i and s_j, the skew between s_i and s_j is *negative skew* if $d_i \leq d_j$, and is *positive skew* if $d_i \geq d_j$. We define the *negative skew bound* (NSB) between s_i and s_j as the maximum value of *negative skew* between s_i and s_j with which the clock operates correctly under a required clock frequency. Similarly, the *positive skew bound* (PSB) is the maximum value of *positive skew* with which the clock operates correctly under a given frequency.

If s_i, s_j are *adjacent*, then,

$$NSB_{ij} = \max(MIN(d_{logic})) + d_{ff} - d_{hold} \quad (3.4)$$

$$PSB_{ij} = P - \min(MAX(d_{logic})) - d_{setup} - d_{ff} \quad (3.5)$$

where $\max(MIN(d_{logic}))$ is the maximum delay of the shortest combinational logic path achievable with feasible gate sizes while satisfying the long path constraint. $\min(MAX(d_{logic}))$ is the minimum delay of the longest combinational logic path achievable with feasible gate sizes while satisfying the short path constraint.

If s_i and s_j are not *adjacent*, then,

$$NSB_{ij} = \infty, \qquad PSB_{ij} = \infty \quad (3.6)$$

In addition, we also define $NSB(v)$ and $PSB(v)$ associated with each node v of a binary tree as the maximum allowable delay difference from v to its two children, a and b.
(I) If the two children nodes of v are sinks, i.e. s_i, s_j, then $NSB(v) = NSB_{ij}$ and $PSB(v) = PSB_{ij}$.
(II) If one or more of the children nodes of v are not sinks, i.e. a and b with subtrees, TS_a and TS_b, then,

$$NSB(v) = \min(d_i(a) - d_j(b) + NSB_{ij}, d_l(a) - d_k(b) + PSB_{kl}) \quad (3.7)$$

$$PSB(v) = \min(d_j(a) - d_i(b) + PSB_{ij}, d_k(b) - d_l(a) + NSB_{kl}) \quad (3.8)$$

for all sink pairs, $s_i, s_l \in TS_a, s_j, s_k \in TS_b$.

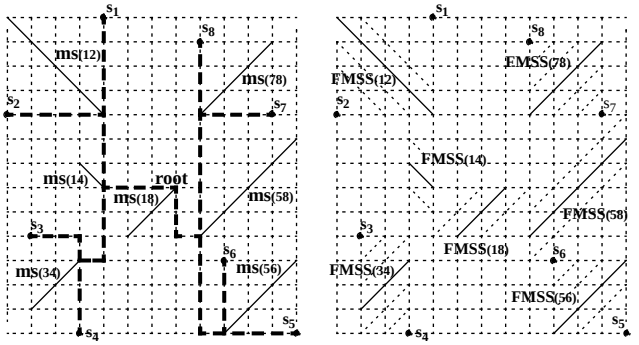
Obviously, a feasible placement of v has to satisfy $NSB(v)$ and $PSB(v)$. As we will discuss later, the existence of this feasible region depends on the tree topology and the placements of v's descendent nodes.

3.3 Merging Segment Perturbation

A *merging segment* (MS), as defined in the DME based algorithms[1, 7, 3], is a line segment associated with an internal node in a clock tree and represents the loci of possible locations of this node. In the Manhattan plane, a *merging segment* is a *Manhattan Arc* which is a line segment (possibly a single point) with a slope of $+1$ or -1. Let $ms(v)$ be the merging segment of a node v, a and b be the children nodes of v. To construct a ZST, the DME algorithm constructs $ms(v)$ from $ms(a)$ and $ms(b)$ in a bottom-up process[1]. Any point on $ms(v)$ satisfies the zero-skew property and at the same time, its two children are merged with minimum added wire. A ZST example is shown in Figure 5(a).

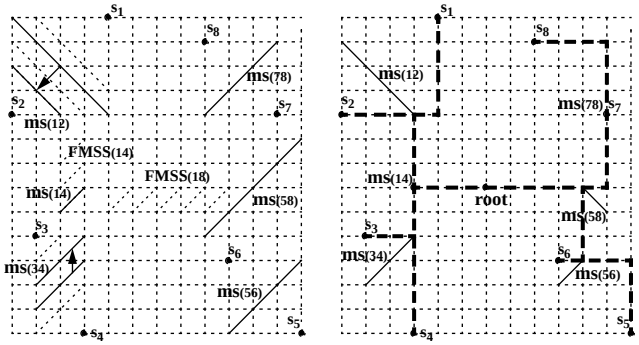
For a BST with non-zero skew bound, a *merging region* is associated with a node which contains all the feasible merging points for a given skew bound[7]. To construct $mr(v)$, the *shortest-distance region* (SDR) between v's children, $mr(a)$ and $mr(b)$ is first found. $mr(v)$ is formed by the set of points within $SDR(v)$ that have minimum merging cost while satisfying a fixed skew bound.

We now assume the Manhattan plane is gridded and there are discrete number of points. Given two *Manhattan Arcs*, l_1 and l_2, the *shortest distance region* between l_1 and l_2, denoted $SDR(l_1, l_2)$ is the set of points



(a) (b)

Figure 5: (a) A zero-skew tree constructed with *merging segments* using the DME algorithm. (b) The *feasible merging segment set* of each node when the lower-level merging segments are fixed.



(a) (b)

Figure 6: (a) Examples of MSP. The arrow indicate the selection of a new FMS within the $FMSS(34)$ or $FMSS(12)$. The new FMSS of the parent node, $FMSS(14)$, is formed and $SDMS(14)$ is chosen as $ms(14)$ which is the closest to its sibling, $ms(58)$. (b) The final UST which minimizes cost function after a sequence of MSP's.

that have minimum sum of Manhattan distance to l_1 and l_2. $SDR(l_1, l_2)$ thus contains a discrete number of Manhattan Arcs. For a given topology, G, we construct a tree of *feasible merging segment sets* (FMSS). Each node $v \in G$, is associated with a $FMSS(v)$. If v is a sink, s_i, then $FMSS(v) = \{s_i\}$. If v is an internal node with children a and b and the merging segment $ms(a)$ and $ms(b)$ are chosen, then a *feasible merging segment* (FMS) of v is a Manhattan Arc which contains possible locations of v such that (i) the negative and positive skew bounds, $NSB(v)$ and $PSB(v)$ given by (3.7) and (3.8) are satisfied; (ii) the merging cost(or added wire) is minimized. Therefore, $FMSS(v)$ is defined by its children, $ms(a)$ and $ms(b)$ in a bottom-up process. For any two FMSS's, $FMSS(a)$ and $FMSS(b)$, the *shortest distance merging segments*, denoted as $SDMS(a)$ and $SDMS(b)$, are a pair of Manhattan Arcs in $FMSS(a)$ and $FMSS(b)$ which are closest to each other. Figure 5(b) shows the FMSS for each node when the lower-level merging segments are fixed.

Lemma 1: *If every node $v \in T$ is chosen within $FMSS(v)$, then skew between any two sinks in T satisfies either their negative skew bound or their positive skew bound.*

Under the linear and Elmore delay, we have the following lemmas regarding $FMSS(v)$.

Lemma 2: *Under both the linear and Elmore delay, the $FMSS(v)$ for any node $v \in G$ exists, i.e. there is at least one FMS, $ms(v)$, if and only if $NSB(v) + PSB(v) \geq 0$.*

Lemma 3: *Under both the linear and Elmore delay models, for any FMS within $SDR(ms(a), ms(b))$, the difference in delay from v to its two children, a and b is a linear function of the position of the FMS. If $FMSS(v)$ exists, it can be constructed in constant time.*

Due to space limitation, we relegate the detailed computation of $FMSS(v)$ to [12].

A *merging segment perturbation* associated with a node v, denoted as $MSP(v)$ is a move that selects another FMS within $FMSS(v)$. Figure 6(a) shows two MSPs as examples. When selecting another merging segment of v, the FMSS's of v's parent and ancestor nodes are updated. This results in a new configuration of the clock tree, T, and hence a new set of skews. They are also allowable skews according to Lemma 1. Let p denote the parent node of v and u be the sibling of p. During an MSP, $FMSS(p)$ is redefined by the new $ms(v)$ and v's sibling. Then $SDMS(p)$ which is the closest to $ms(u)$ is chosen as the new $ms(p)$. As shown in Figure 6(a), $SDMS(14)$ is chosen as $ms(14)$, since it is the closest to $ms(58)$. The new $ms(p)$ and $ms(u)$ form the FMSS of their parent node, i.e. the grandparent of v. This process is iterated in a bottom-up process until the root node of T is updated. Lemma 2 points out that the FMSS of a node may not always be found. An $MSP(v)$ is acceptable only if all the FMSS's of v's ancestors are found. A top-down process connects the merging segments by shortest distance, analogous to DME[7]. Note that the variable bounds of NSB and PSB are used at each node and only v's ancestor nodes are updated. With a binary tree topology, one MSP takes $O(n)$ in the worst case and $O(\log n)$ in the average case. Figure 6(b) shows a final tree after a sequence of MSPs and the cost function is minimized.

The following Theorem suggests that the entire feasible solution space can be asymptotically explored.

Theorem 1: *For a given tree topology, any configuration of the clock tree that result in allowable skews can be transformed to another by performing a sequence of MSPs.*

3.4 Topology Generation

From the definitions of $NSB(v)$ and $PSB(v)$, it is evident that the skew constraints at higher level nodes(closer to the root) are tighter. If the the high level nodes are given small skew budget, they will have fewer feasible merging segments. If the topology is very asymmetric, then the delay difference of two subtrees

under Elmore model may become so large that feasible merging segments are limited or can not even be found according to Lemma 2,

More importantly, our objective is to produce useful skew – the negative skew. If at an internal node, v, there are two or more pairs of sinks between the two subtrees which have opposite logic path direction, then the NSB of one sink pair is constrained by the PSB of another. The negative skew of one pair of sinks results in the positive skew of another pair of sinks. Good results are unlikely in this case.

These observations indicate that the tree topology is very important to the success of the UST solution. Intuitively, we would like to partition the sinks into groups that have loose skew bounds with each other. Most of the adjacent sinks across two groups should have the same logic path direction(either *forward* or *backward*) such that negative skew can be maximally produced. This suggests that a top-down partitioning rather than a bottom-up clustering approach should be used since the skew bounds between sinks can be evaluated globally. We therefore adopt a partitioning heuristic for the UST problem[1, 12].

We consider recursively cutting the sink set S into two subsets S_1 and S_2 in the Manhattan plane. Each cut would result in an internal node of the tree topology. At each partition, we choose a cut to (i) maximize the skew bounds for the resulting node, and (ii) maximize the number of forward(or backward) sink pairs across the cut. For a bipartition, $S = S_1 \cup S_2$, let FW_{12} and BW_{12} denote the number of sink pairs across the cut that have logic path from S_1 to S_2(forward) and from S_2 to S_1(backward). The total number of adjacent sink pairs across the cut is then, $SP_{12} = FW_{12} + BW_{12}$. We define the *skew bound* between S_1 and S_2 as $SB_{12} = min(NSB_{ij}, PSB_{kl}) + min(PSB_{ij}, NSB_{kl}), \forall s_i, s_l \in S_1, s_j, s_k \in S_2$. We use a weighted function to evaluate a cut,

$$W_{12} = w_1(SB_{12}/SP_{12}) + w_2|FW_{12} - BW_{12}| \quad (3.9)$$

where w_1, w_2 are determined by experiment. For lower level nodes, the partition between the two subsets should also be *balanced* to keep the delay difference small. Let $Cap(S_1)$ and $Cap(S_2)$ be the total capacitance of S_1 and S_2, respectively. $|Cap(S_1) - Cap(S_2)| \leq \epsilon$, where ϵ is gradually reduced with each level of cuts.

An example of using this bi-partitioning heuristic is shown in Figure 7. Figure 7(a) shows the NSB and PSB between the sinks. The clock tree using the topology generated by a clustering based algorithm is shown in Figure 7(b)[5]. It results in an undesirable positive skew between s_3 and s_4. In contrast, the bi-partitioning heuristic generated all negative skews and reduced the routing cost.

3.5 Gate Sizing

Unlike previous approaches in gate sizing with clock skew optimization[2], our feasible solution space is defined by a clock tree with reasonable cost(wire length)

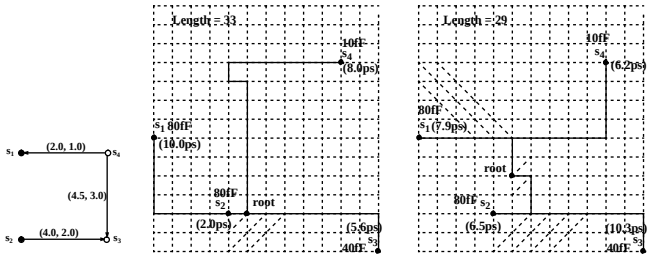


Figure 7: An example showing the effects of topology. (a) (NSB, PSB) between sinks. (b) The tree resulted from the clustering method. s_3 and s_4 have a positive skew of 2.4. (c) The tree resulted from the bipartition heuristic. All sink pairs have negative skew.

and feasible gate sizes. Our approach has two advantages: (i) With the feasible solution region controlled by clock routing, we may take into account both the logic and clock power; (ii) With known skews between each pair of flip-flops, we may decompose the sequential circuit into subcircuits which are individually combinational circuits.

Because gate sizing is a time consuming process, we predetermine the minimum power of each combinational block. The logic power for an allowable skew value and the corresponding gate sizes are stored in a look-up table. At each iteration of our UST routing algorithm, the cost function can be updated in constant time. Finally when the minimum cost function is achieved and the skews between each pair of flip-flops are known, the gate sizes which results in minimum power under the closest skew value are chosen. Due to space limitation, our gate sizing solution is relegated to [12].

4 Experimental Results

Our algorithms have been implemented in C in a Sun Sparcstation10 environment and has been tested on 2 industry circuits and 3 ISCAS89 benchmark circuits as shown in Table 2. The ISCAS89 circuits were first translated with some modifications to a $0.65\mu m$ CMOS standard-cell library[4]. We implemented a previous standard-cell gate sizing algorithm[9] to be used with the DME based ZST and BST clock routing algorithms[1, 3], to compare with our UST solution. Table 1 compares the results of UST with two other approaches: (i) ZST clock routing[1] and gate sizing; (ii) BST clock routing[7, 3] and gate sizing. To ensure correct clock operation, the smallest allowable skew bound(both negative and positive) of all clock sink pairs has to be chosen as the fixed skew bound in the BST/DME algorithm. Note that since BST does not recognize the difference between negative and positive skew, it may even produce skews that result in worse power in gate sizing. Table 3 compares the routing results of ZST, BST algorithms, and the UST routing

Table 1: Power reduction of UST over ZST and BST. The topology in UST-CL is generated by the clustering algorithm. UST-BP uses the bi-partitioning heuristic.

Circuits	Clock Power (mW)				Logic Power (mW)				Reduction	
	ZST	BST	UST-CL	UST-BP	ZST	BST	UST-CL	UST-BP	$\frac{UST}{ZST}$	$\frac{UST}{BST}$
Ckt1	43.53	43.32	43.41	43.22	58.35	55.45	46.08	41.9	16%	14%
Ckt2	20.95	20.66	20.69	20.54	102.66	93.34	85.87	83.36	16%	11%
s1423	5.224	5.161	5.182	5.170	22.48	24.70	18.69	18.17	16%	22%
s5378	11.03	10.82	10.86	10.79	124.4	126.5	114.0	110.2	11%	12%
s15850	32.93	32.44	32.38	32.25	416.5	421.3	356.1	338.9	17%	18%

results with topology generated by both the clustering-based algorithm[5] and the bipartitioning heuristic. BST only achieves small savings in wire length over ZST due to the small value of the fixed skew bound. In contrast, the UST approach reduces wire length in all but one case.

Table 2: Five circuits tested by the UST algorithm.

Circuits	Frequency	# of FF's	# of gates	Supply
Ckt1	200	106	389	5.0
Ckt2	100	391	3653	3.3
s1423	33	74	657	3.3
s5378	100	179	2779	3.3
s15850	100	597	9772	3.3

Table 3: Comparison of clock tree wire lengths(μm).

Circuits	ZST	BST (Bound)	UST-CL	UST-BP
Ckt1	3982	2998 (0.1ns)	3051	2755
Ckt2	17863	16002 (0.2ns)	16217	15924
s1423	8823	6651 (1.4ns)	6830	6756
s5278	12967	10645 (0.3ns)	11068	10229
s15850	30579	28348 (0.2ns)	27369	25580

In the implementation of simulated annealing, the outer loop stopping criterion (frozen state) is satisfied when the value of the cost function has no improvement for five consecutive steps. The inner loop stopping criterion (Equilibrium state) is decided by specifying the number of iterations at each temperature. We use $n \times TrialFactor$ in the experiments, where $n = |S|$. For all tested cases, the $TrialFactor$ ranges from 100 to 600. We choose the initial temperature as $t_0 = -\frac{\Delta C}{\ln \chi}$, where ΔC is obtained by generating several transitions at random and computing the average cost increase per generated transition, and χ is the acceptance ratio. In choosing the cooling schedule, we start with $\delta(t) = 0.85$, then gradually increase to $\delta(t) = 0.95$, and stay at this value for the rest of the annealing process.

5 Conclusion

Assuming zero-skew or a fixed skew bound in clock routing could lead to pessimistic results of logic and clock power. On the other hand, *clock skew optimization* with arbitrary skew values could be too costly for clock distribution. We have shown that *useful-skew* clock routing with gate sizing based on allowable negative and positive skew bounds can achieve savings in total power and clock routing cost.

6 Acknowledgement

We are grateful to C. W. Albert Tsao and Prof. Andrew Kahng of UCLA for providing us with the program of BST algorithms for comparisons.

References

[1] T.H. Chao, Y.C. Hsu, J.M.Ho, K. D. Boese, and A. B. Kahng. Zero skew clock net routing. *IEEE Trans. on Circuits and Systems*, 39(11):799–814, Nov. 1992.

[2] W. Chuang, S. S. Sapatnekar, and I. N. Hajj. A unified algorithm for gate sizing and clock skew optimization. In *IEEE Intl. Conf. on CAD*, pages 220–223, Nov. 1993.

[3] J. Cong, A.B. Kahng, C.K. Koh, and C-W. A. Tsao. Bounded-skew clock and steiner routing under elmore delay. In *IEEE Intl. Conf. on CAD*, 1995.

[4] National Semiconductor Corp. *cs65 CMOS Standard Cell Library Data Book*. National Semiconductor Corp, 1993.

[5] M. Edahiro. A clustering-based optimization algorithm in zero-skew routings. In *Proc. of 30th Design Automation Conf.*, pages 612–616, 1993.

[6] J. P. Fishburn. Clock skew optimization. *IEEE Trans. on Computers*, 39(7):945–951, 1990.

[7] D. J.-H. Huang, A. B. Kahng, and C-W. A. Tsao. On the bounded-skew clock and steiner routing problems. In *Proc. of 32nd Design Automation Conf.*, pages 508–513, 1995.

[8] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):458–463, May 1983.

[9] Shen Lin and Malgorzata Marek-Sadowska. Delay and area optimization in standard-cell design. In *Proc. of 27th Design Automation Conf.*, pages 349–352, 1990.

[10] R-S. Tsay. An exact zero-skew clock routing algorithm. *IEEE Trans. on CAD*, 12(3):242–249, 1993.

[11] Joe G. Xi and Wayne W.M. Dai. Buffer insertion and sizing under process variations for low power clock distribution. In *Proc. of 32nd Design Automation Conf.*, June 1995.

[12] Joe G. Xi and Wayne W.M. Dai. Low power design based on useful clock skews. In *Technical Report, UCSC-CRL-95-15, University of California, Santa Cruz.*, 1995.