Clock Tree Resynthesis for Multi-corner Multi-mode Timing Closure

Subhendu Roy[‡], Pavlos M. Mattheakis[†], Laurent Masse-Navette[†], David Z. Pan[‡]
[‡]Department of Electrical and Computer Engineering, University of Texas at Austin, USA
[†]Mentor Graphics, Grenoble, France
subhendu@utexas.edu, {pavlos_matthaiakis,Laurent_Masse-Navette}@mentor.com,
dpan@ece.utexas.edu

ABSTRACT

With aggressive technology scaling and complex design scenarios, timing closure has become a challenging and tedious job for the designers. Timing violations persist for multi-corner, multi-mode designs in the deep-routing stage although careful optimization has been applied at every step after synthesis. Useful clock skew optimization has been suggested as an effective way to achieve design convergence and timing closure. Existing approaches on useful skew optimization (i) calculate clock skew at sequential elements before the actual tree is synthesized, and (ii) do not account for the implementability of the calculated schedules at the later stages of design cycle. Our approach is based on a skew scheduling engine which works on an already built clock tree. The output of the engine is a set of positive and negative offsets which translate to the delay and accelerations respectively in clock arrival at the clock tree pins. A novel algorithm is presented to accurately realize these offsets in the clock tree. Experimental results on large-scale industrial designs demonstrate that our approach achieves respectively 57%, 12% and 42% average improvement in total negative slack (TNS), worst negative slack (WNS) and failure-end-point (FEP) with an average overhead of 26% in clock tree area.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuits**]: Design Aids;

Keywords

Useful skew, clock skew scheduling, ECO, MCMM, CTS

1. INTRODUCTION

Clock skew is the difference in clock arrival times at different sequential elements in the clock-distribution network. A lot of work has been done in the past to minimize clock-skew [1][2][3]. Targeting global zero skew not only costs in area and power, but also limits the achievable operating frequency to the maximum data path delay in the circuit. This has led to a paradigm shift from skew minimization to useful skew optimization as the latter has the potential to significantly improve design performance [4][5][6][7][8][9].

As technology scales aggressively in the nanometer regime, interconnects play a determining role in timing and uncertainty due to process variations [10][11] and the multi-corner analysis becomes more and more tedious. [12] has proposed an algorithm for chip-level clock tree synthesis (CTS) to tackle clock divergence issue in different corners. However, it does not take into account the timing information on data path for CTS. Also, a chip has to operate in several modes to reduce power dissipation. For instance, a design can be in active and sleep modes when performance and power are the main concerns respectively. Consequently, timing closure has posed a challenging job for designers to meet stringent silicon delivery targets [13], especially with multi-corner, multi-mode (MCMM) designs. In [14][15] clock tree aware placements are performed with the objective of reducing total wire-length and/or switching power, but they do not account for any timing improvements. Several works have focused on timing optimization during placement and routing as well [16][17][18]. But in spite of all these efforts, timing violations still exist after detail routing in MCMM designs. So the designers have to intervene manually to analyze and fix the timing violations considering every mode and process variation altogether in an iterative and non-convergent way, where as the verification engineers need to run timing analysis for each scenario[1].

Engineering change order (ECO) is always used after detail routing in order to fix existing timing violations by incremental adjustment of pertaining cells and nets [19][20]. These ECO adjustments, focused mainly on data path optimization, are not sufficient to handle all timing violations. So data path aware clock scheduling becomes an important step for timing closure, as it allows modifications in the clock tree which is towards timing closure. Several works study the clock scheduling problem. In [7] clock skew scheduling is formulated as a constrained quadratic problem, minimizing the least square error between the computed clock schedule, consistent to the interconnection between the registers, and the target clock schedule. [21] presents a fast primal-dual based approach for minimal clock period, improving over Burn's algorithm [22] in run-time complexity. [9] even tackles the clock scheduling problem in presence of process variations by ILP formulation. But the issues with these approaches are (i) actual implementation of that clock schedul-

[1]any mode/corner combination

ISPD'14, March 30 - April 02 2014, Petaluma, CA, USA.

http://dx.doi.org/10.1145/2560519.2560524

ing is difficult to achieve in real designs, especially at later design stages (ii) they are unaware of MCMM scenarios.

[23] formulates an LP problem to optimize clock period in the post-CTS stage by bounded delay buffering at the leaves of the clock tree. But since this work only considers inserting delay but not speeding up clock arrival at the leaves, the scope of the optimization is limited and buffering at the leaf level introduces a high area overhead in clock tree. Furthermore, [23] does not tackle MCMM scenarios. A recent work [24] focuses on the realization of the useful skew on industrial-scale designs at post-routing stage. It also performs local transformations at the leaf-level by inserting/removing buffers to minimize negative D-slack/Q-slack[2] violations. For instance, if $Dslack < 0$, it means the data arrives too late or clock arrives too early. Fig. 1 shows an example to mitigate D-slack violation by delaying the clock arrival. But it might cause Q-slack violation if there is no enough positive Q-slack available. The main issues of this work are (i) it does not have the global view of the clock tree, instead performs timing optimization greedily. So this approach can not handle negative slacks at both sides (D and Q) or negative slack at one side with very less available positive slack at the other side, which is a common situation in today's high-performance time-constrained real designs, (ii) area-overhead in clock tree is high as it works only at leaf-level, (iii) Speeding up clock arrival to fix Q-slack violations by only removing buffer is hardly realizable in practice to be discussed in Section 2.3 (Fig. 5).

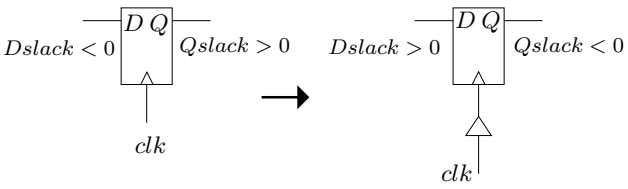


Figure 1: Buffer insertion to mitigate D-slack violation can cause Q-slack violation

To tackle these issues, a novel clock tree resynthesis methodology is presented in this paper. Instead of estimating clock schedule at the leaf level registers, our approach considers offsets in clock arrival at the clock tree driver pins of any placed design with already synthesized and routed clock tree. To consider MCMM scenarios, we develop an LP solver based on [25], for calculating these offsets. We illustrate in Section 2.2 that it is easier to realize the positive offsets by inserting buffer chains, but at the cost of clock tree area. On the other hand, the negative offset realization is disruptive and can have catastrophic effects on the timing profile of the design unless handled properly. As a result, realization of an arbitrarily large negative offset is not feasible. We run experiments with the LP solver for industrial designs and come to the conclusion that a significant gain in timing metrics is possible by realizing positive offsets and bounded negative offsets. We develop a slack manager infrastructure which keeps track of the available slacks for clock arrival at the clock pins of the clock tree network. By utilizing the positive slack at the fan-out cone of the clock tree elements as a safe margin, our algorithm realizes the negative offsets incrementally through clock tree restructuring or sizing.

[2]The slack at the input/output pin of a register is defined as D-slack/Q-slack

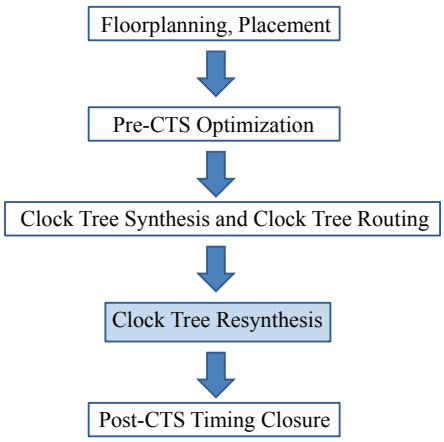


Figure 2: Our methodology in a conventional back-end flow

Fig. 2 illustrates the steps of a conventional back-end flow and where our methodology for clock tree resynthesis fits into this. The benefits of our methodology are two fold. Firstly, it helps to lead to the timing closure. Secondly, post-CTS timing closure involves ECO adjustments, such as data path optimization etc., which generally cost a significant area/power penalty. So more we advance towards the timing closure by clock tree resynthesis, better is the savings in terms of area/power. The key contributions of our paper are as follows.

- To the best of our knowledge, this is the first work to consider offsets at output pins of clock tree cells for improving timing metrics in a placed design with already routed clock tree instead of estimating clock schedule at the leaf level registers. Moreover, the offset calculation is tightly coupled with feasibility in realizing those offsets.
- A novel algorithm which is non-intrusive and area-efficient is presented that realizes negative offsets.
- A methodology for clock tree resynthesis is presented which has significantly improved timing metrics of large scale industrial designs (after placement and clock tree routing) under MCMM scenarios.

The rest of the paper is organized as follows. Section 2 illustrates the concept of feasibility aware clock scheduling in presence of MCMM scenarios using an LP solver. Section 3 presents our novel algorithm to realize the negative offsets predicted by the LP solver and the overall methodology for clock tree resynthesis. Section 4 presents the experimental results of our approach for industry-strength designs. Section 5 discusses about the applicability of our methodology and future work with a conclusion in Section 6.

2. FEASIBILITY AWARE CLOCK SCHEDULING

In this section we first present an LP solver based on [25] to calculate the offsets in clock arrival at the clock driver pins under MCMM scenarios. Although this LP solver is

not our main contribution in this work, it is imperative to address concisely how the LP solver tackles various modes and corners in the design. Then we illustrate the approach for positive offset realization and the issues in realizing negative offsets. Finally, the notion of feasibility aware clock scheduling is introduced.

2.1 LP Solver

In [25] an LP engine is presented, which estimates the clock-scheduling for a design under MCMM scenarios targeting the minimization of timing metrics, such as the total negative slack (TNS) and total hold slack (THS)[3]. To include the various corners in the design, scaling factors (c_i) for each corner i are calculated having as reference the constraint corner *i.e.*, $c_i = 1$ for the constraint corner and $c_i < 1$ for any other corner. These scaling factors are used in the set-up/hold time analysis for different corners. With respect to multiple mode handling, the functional timing paths across all active modes are analyzed. Additionally, on-chip-variation (OCV) derates [26] calculated on the already built tree are introduced in the LP solver as means to reduce the variability effects on the resultant timing profile.

We develop an LP solver based on [25], to calculate the positive and negative offsets at the output pins of the leaf-level gates/buffers (driving sequential leaf cells) in terms of clock tree level which corresponds to intrinsic buffer delay (minimum buffer delay in the design), denoted by D^{buf}_{min}. Positive (negative) offset of d_{off} at any pin signifies that the clock-arrival at that pin is to be delayed (fastened) by d_{off}. Any offset d_{off} in the constraint corner is equivalent to an offset of $c_i \times d_{off}$ in the i^{th} corner. We can specify the range of these offsets by constraining minimum level (L^{off}_{min}) and maximum level (L^{off}_{max}). For instance, suppose the D^{buf}_{min} of a design is 60 pico seconds (ps) and we specify $L^{off}_{min} = -2$ and $L^{off}_{max} = 3$, then the LP solver will estimate the offsets of values -120ps, -60ps, 60ps, 120ps, 180ps along with a prediction of timing improvement. The calculation and realization of the offsets are tightly coupled in this work. Additionally, the realization maintains the timing profile of the parts of the design which should not be affected.

2.2 Positive Offset Realization

Positive offset realization is accomplished by inserting route aware delay elements. Fig. 3 illustrates the realization of a positive offset at the output pin of repeater B_1. Initially, the LP solver predicts that a positive offset (d_{off}) should be realized at the output pin (*op*) of the buffer B_1, *i.e.* the clock arrival of the buffers/leaf-cells driven by B_1 should be delayed by d_{off}. We can implement this positive offset by incorporating a delay element D (merely a buffer chain) of d_{off} at *op*. While doing this, we consider various corners and insert/size/place the delay block accordingly to realize this positive offset as accurate as possible across all corners. Additionally it should be guaranteed that the offset realization does not degrade the quality of the clock tree *e.g.* design rule check (DRC) violations are not increased. It should be stressed that the positive offset realization is not intrusive as the parts of the clock tree which are irrelevant to the inserted offset are not affected. For instance in the example shown in Fig. 3 there is no impact of D on B_2 and B_3, the siblings of B_1, as B_1 effectively acts as a shield buffer. Consequently, there is no side-effect on the clock tree.

[3]Here THS signifies total negative hold slack

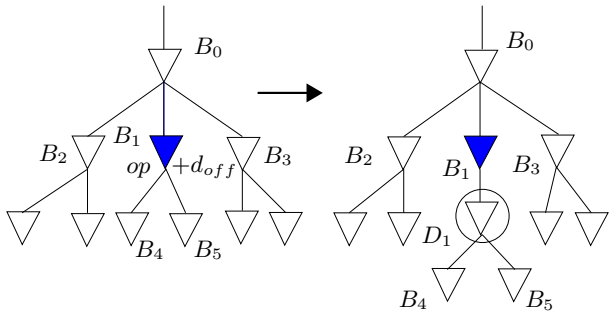
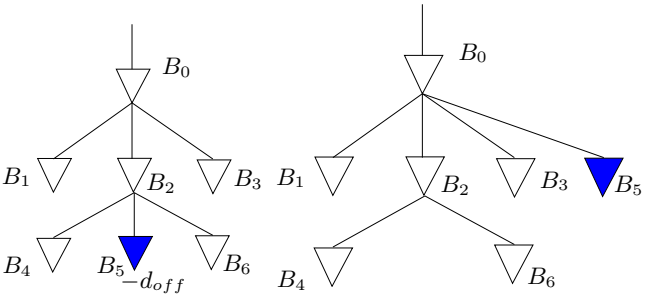


Figure 3: Positive offset realization



(a) B_5 output pin should be accelerated by d_{off}

(b) B_5 is shifted one level upwards the tree

Figure 4: Negative offset realization

2.3 Issues in Negative Offset Realization

The negative offset realization poses more challenges. A representative example is the following. Let us assume that the LP engine predicts a negative offset (d_{off}) for the output pin of buffer B_5 as shown in Fig. 4. This offset can be realized by placing, sizing, or changing the clock tree structure. Each one of the aforementioned approaches has its own drawbacks. For instance, placing B_5 at another location will force its parent (B_2) to drive a different amount of load than before, altering thus the arrival time of all clock tree nodes at B_2's transitive fanout (TFO). Sizing has similar effects on B_5's siblings as B_2 will again have to drive a different amount of load defined by the gate sizing result. Another option is to restructure the clock tree, moving upwards cell B_5. In this case, the arrival time to FFs at the TFO of B_5 is reduced but multiple side effects alter the arrival times to the old and the new siblings of B_5. This is due to the load decrease and increase at the nets driven by B_2 and B_0 respectively and that affect all the FFs at the TFO of B_0. [24] has illustrated that clock arrivals could be accelerated by removing the corresponding buffer B_1 (Fig. 5). But this can be useful only when it does not have any sibling, which is not common in practice. Furthermore, this technique might not be effective in that case as well as (i) B_0 is now driving 3 buffers instead of 1, viz. B_2, B_3 and B_4, (ii) B_0 has to drive more wire-load. When B_1 is far away from B_0, then the wire-load increase is even more. As a result the clock arrival is delayed at the TFO cone of B_0.

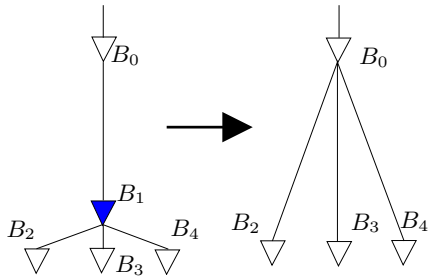


Figure 5: Buffer removal might not be effective in realizing negative offset

From the above it can be concluded that realizing negative offsets in the clock tree imposes side effects which may significantly change the timing profile of the design and possibly cancel the expected timing gains. Additionally, it should be noted that the more the negative an offset is, the more the pin should be moved upwards the tree. As a consequence, more FFs downwards the tree will be affected increasing the probability of degrading the timing instead of optimizing it.

2.4 Offset Bounds

Any positive offset can be realized by injecting a delay element with delay equal/close to the offset. Negative offsets, on the other side, can not always be realized. For instance, if a pin has a negative offset with delay greater than the arrival time from the clock root to this pin, then it can be deduced that this offset is infeasible for this pin. Hence, the pins which can carry offsets should be bounded to guarantee that the calculated negative offset can be realized. A per-pin negative offset bound would be cumbersome as the side effects of each negative offset realization should be modeled into the LP solver, thus a global bound was selected for all pins. An experiment was performed to calculate a negative bound which should deliver as much timing gain as possible and at the same time be as less disruptive as possible, *i.e.*, closer to zero.

Three LP runs were performed with real industry-strength benchmarks. The first run corresponds to LP solutions with only positive offsets, whereas the second and the third allow for one and three levels of negative offset respectively. For all three runs the positive offset bound was set to three. The results are shown in Fig. 6, where TNS predicted by the solver for each one of the three aforementioned experiments are normalized w.r.t. the original TNS of each design, which is the TNS after placement, clock tree synthesis and routing by an industrial tool. We observe that there is a significant improvement from original TNS to the TNS predicted in first run and from predicted TNS in first run to the second run, but the same trend does not continue as the bound further decreases. From the above it can be concluded that most of the potential TNS gain can be acquired by pairing a single level of negative offset with many levels of positive offset. This finding will be used throughout this work as the solver will be bounded to produce solutions with a single level of negative offset.

3. CLOCK TREE RESYNTHESIS

Section 2.4 showed that significant TNS gains can be enjoyed if pins which can carry offsets are bounded to -1 level ($L_{min}^{off} = -1$). In this section we present a methodology for clock tree resynthesis to improve timing in a routed clock

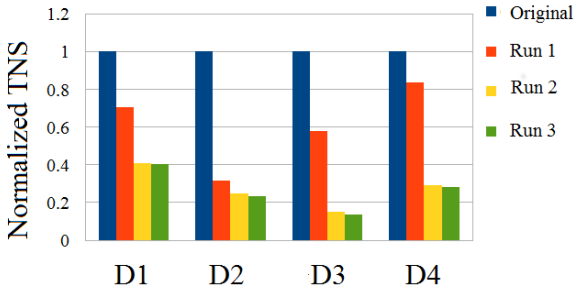


Figure 6: Normalized TNS prediction by LP solver for industrial designs

tree. A novel algorithm is presented which realizes accurately one level of negative offset, so that the predicted TNS gain is maintained after offset realization. The two basic operations used are sizing and restructuring. It should be stressed that the restructuring is always performed within the scope of a hyper-net to guarantee that the clock gating function will be preserved by the clock tree restructuring. A hyper-net is a set of logically equivalent or opposite polarity nets separated by buffers/inverters in the same physical partition as the root driver of the top net, and thus this set is necessarily connected in a tree topology. The root of this tree (hyper-root) is either the driver pin of a clock gate or a clock root. The elements of any hyper-net are comprised of all the nets traversed until another hyper-root is visited. Fig. 7 demonstrates a clock tree comprised of 3 hyper-nets. The datapath logic and the enable signals at the clock tree clock gates are omitted in the figure.

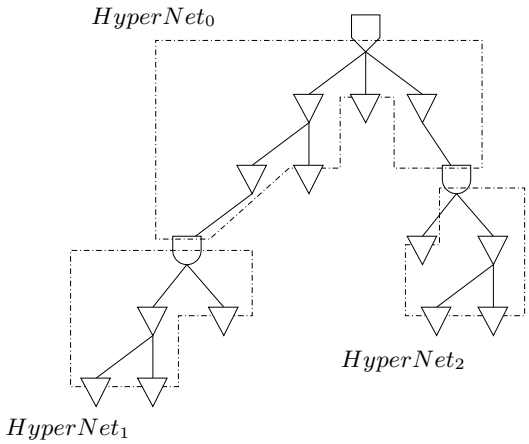


Figure 7: Clock tree decomposition to hyper-nets

The key to accurately realizing negative offsets is the utilization of the positive slack. If a clock tree driver pin has only sequential cells with positive slack (more specifically Q-slack) at its transitive fanout it is annotated as a potential acceptor of pins with negative offset. In this way, negative offsets are realized accurately without degrading the total negative slack. We develop an engine, called slack manager, which helps to extract the potential acceptors for pins with negative offset.

3.1 Slack Manager

The slack manager is an engine, that keeps track of certain parameters at any pin corresponding to the D-slack

and Q-slack of the leaf-cells in the TFO cone of that pin ($leafCells_{fo(pin)}$). We define the following parameters.

- $Qslack_{sum}(pin)/Dslack_{sum}(pin)$ = sum of the negative Q/D-slacks at $leafCells_{fo(pin)}$
- $Qslack_{cnt}(pin)/Dslack_{cnt}(pin)$ = count of $leafCells_{fo(pin)}$ having negative Q/D-slack

We store these parameters per scenario, *i.e.* corner and mode combination. These parameters are calculated recursively in a bottom-up fashion. Algorithm 1 presents the procedure 'BUSlackParamCalculate(pin, $mode$)', which stores the slack-parameters in any pin for all the scenarios active in that *mode*. Lines 3-7 first initialize the parameter values at each scenario. Then at Line 8 it is checked whether the pin is a leaf, and in this case it gets the Q-slack value from the timer (Line 5). If the Q-slack is less than a threshold, then (Lines 13-14) we set $Qslack_{cnt}$ to be 1 and $Qslack_{sum}$ to be the Q-slack value. In the other case, *i.e.*, for non-leaf pins, Line 22 calls the procedure recursively for all of its children pins (Note children of a pin depends on *mode*) and then it accumulates the values of its children (Lines 23-24). In our implementation, we have set this threshold to be 0, and thus these parameters respectively estimates the count of $leafCells_{fo(pin)}$ with negative Q-slack and sum of negative Q-slacks of $leafCells_{fo(pin)}$.

Algorithm 1 Procedure to calculate slack parameters

```
1: Procedure BUSlackParamCalculate(pin, mode);
2: activeCorners ← corners active in mode;
3: for all cor ∈ activeCorners do
4:    scn ← combination(mode, cor);
5:    Qslack_sum(pin, scn) ← 0;
6:    Qslack_cnt(pin, scn) ← 0;
7: end for
8: if isLeaf(pin) then
9:    for all cor ∈ activeCorners do
10:      scn ← combination(mode, cor);
11:      Qslack ← getQslack(pin, scn);
12:      if Qslack < slackThreshold then
13:         Qslack_cnt(pin, scn) ← 1;
14:         Qslack_sum(pin, scn) ← Qslack;
15:         return
16:      end if
17:   end for
18: end if
19: for all childPin ∈ childList(pin, mode) do
20:   for all cor ∈ activeCorners do
21:      scn ← combination(mode, cor);
22:      BUSlackParamCalculate(childPin, scn);
23:      Qslack_cnt(pin, scn) ← Qslack_cnt(pin, scn) +
         Qslack_cnt(childPin, scn);
24:      Qslack_sum(pin, scn) ← Qslack_sum(pin, scn) +
         Qslack_sum(childPin, scn);
25:   end for
26: end for
27: return
28: end Procedure
```

The execution of the algorithm is demonstrated with a representative example in Fig. 8. Cells B_2 and B_3's output pins have $Qslack_{cnt}$ equal to 1 due to cells $ff3$ and $ff5$ respectively. B_1's output pin has $Qslack_{cnt}$ equal to 2 which

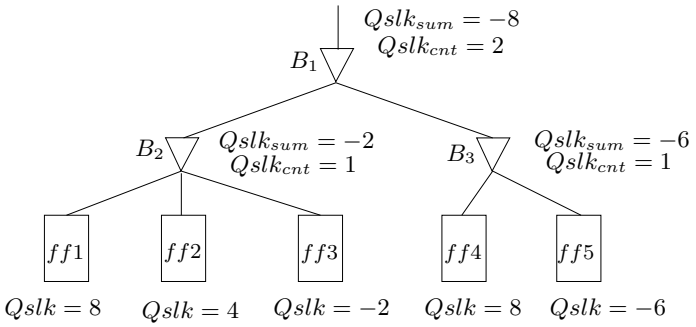


Figure 8: Q-slack parameter calculation

results from the addition of its children's corresponding values. The $Qslack_{sum}$ values are calculated accordingly.

Similar calculation is done for D-slack parameters and it has not been shown in Algorithm 1 or Fig. 8 for brevity.

3.2 Negative Offset Realization Algorithm

The slack manager exposes the space that is available for negative offset realization in terms of slack. The Negative Offset Realization Algorithm (NORA) utilizes this space to (i) accurately realize all negative offsets and (ii) gain the improvement in total negative slack calculated by the LP solver.

Algorithm 2 captures the functionality of NORA for a single pin, p (output pin of a cell c), with negative offset. Initially, a reference (constraint) scenario is chosen along with p's parent, p_{par}. Then, (Line 5) it is decided whether the negative offset will be realized by restructuring the clock tree or by sizing. This decision is made after the slack parameters calculated by the slack manager for p_{par} are modified to compensate for the case when p is detached from p_{par}. These new values are named $Qslack^{eff}_{sum}$ and $Qslack^{eff}_{cnt}$ and they are calculated according to the following formulas:

$$Qslack^{eff}_{sum}(p_{par}, scn) = Qslack_{sum}(p_{par}, scn) - Qslack_{sum}(p, scn) \quad (1)$$

$$Qslack^{eff}_{cnt}(p_{par}, scn) = Qslack_{cnt}(p_{par}, scn) - Qslack_{cnt}(p, scn) \quad (2)$$

The effective $Dslack$ values are calculated accordingly. If $Qslack^{eff}_{count}$ is greater than $Dslack^{eff}_{count}$ for p_{par}, then it is preferable to reduce the load at p_{par} fanout as in this way the clock will arrive faster to the sequential cells and the negative slack at the Q side will be reduced. Thus, it is chosen to detach p from p_{par} and connect it to another node higher in the tree, as in this way not only the negative offset will be realized, but also the negative slack at the Q-side of the sequential cells at p_{par}'s TFO will be reduced. The above will have a negative impact on the D-side of the sequential cells at p_{par}'s TFO, but it is better to optimize in favor of the Q-side, as the latter affects multiple endpoints with negative slack.

In order to realize the negative offset at p, a driver pin is found higher in the clock tree, so that if p is connected to it, the difference in Arrival Time (AT) will effectively realize the offset. However, these driver pins, called from now on acceptors, should reside at the same scope of hyper-net as p to guarantee the same functionality as mentioned earlier. In addition, the polarity is also matched to take care of

Algorithm 2 Procedure to realize a negative offset

```
1: Procedure NORA(p, offset);
2: scn ← getConstratintSchenario;
3: p_par ← parent(p);
4: bestSol ← currentSol;
5: if Qslack^eff_cnt(p_par, scn) ≥ Dslack^eff_cnt(p_par, scn) then
6:    a_cand ← driver pins in p's hyper-root;
7:    prune a_cand based on level;
8:    remove a_cand elements if their AT is ≥ AT(p)−2 ∗ offset;
9:    for all a ∈ a_cand do
10:      if Qslack_cnt(inPin(a), scn) > 0 then
11:         remove a from a_cand;
12:      end if
13:   end for
14:   sort a_cand according to geometric distance from p;
15:   for all a ∈ a_cand do
16:      connect p with a;
17:      buffer(p);
18:      if cost(currentSol) < cost(bestSol) then
19:         bestSol ← currentSol;
20:      end if
21:   end for
22: else
23:   size(p);
24:   if cost(currentSol) < cost(bestSol) then
25:      bestSol ← currentSol;
26:   end if
27: end if
28: return bestSol;
```

inverters in the clock tree.

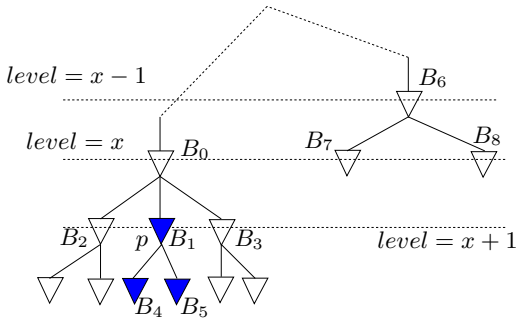
We use the level of any clock-element within the scope of the hyper-net as a coarse knob to identify these acceptor pins a_{cand} (Line 7), *i.e.*, any driver pin which is at higher level than p in the hyper-net would be considered for a potential candidate acceptor. Out of all the candidate driver pins, a finer tuning is done on the basis of AT. The candidates which have AT greater than AT(p) - 2 × $offset$ are disregarded (Line 8) as connecting p to them would not result to the desired arrival time AT(p) - $offset$, considering a best case delay of $offset$ (which is also equal to the intrinsic buffer delay in the design) from the input pin to the output pin p of the corresponding cell c. Finally we prune a_{cand} on the basis of available slack in the TFO of the acceptor pin a (Line 9-13). If there is no available slack, then we remove the element from a_{cand}. This is to ensure that although a would drive more load in case c is connected to a and might worsen Qslack at TFO of a, the available slack is sufficient to account for that (not shown in Algorithm 2).

Then the candidate acceptor pins are sorted according to their proximity to the pin p as it is assumed that the acceptors which are closer will be directly connected realizing the desired offset without incurring extra buffering which would increase the total area (Line 14).

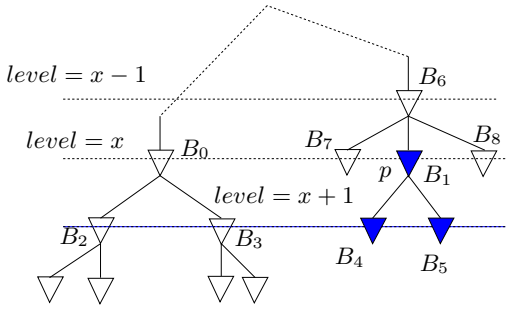
Afterwards, the sorted candidate acceptor pins are examined. Initially, p is connected to the candidate acceptor pin a and buffering is applied on the net between them. Then the cost of the current solution is estimated. The solution with the minimum cost is committed by backtracking mechanism. This cost estimation depends on the accuracy of realizing the offset. The closer the AT difference seen at p approaches the desired negative offset value, lesser is the cost. In addition, if it introduces any new DRC violation, then the cost is set to infinity making the solution infeasible. If there are lot of candidate acceptors, the first 10 acceptors are explored. This reduces run time, and at the same time helps to achieve area-efficient restructuring due to the proximity of the acceptors to the pin p. If there is no potential acceptor with available slack, the acceptor with maximum $Qslack_{sum}$ across all scenarios is chosen.

In the case where buffering was chosen instead of clock tree restructuring (Line 5), p is sized and the solution is committed. Interestingly, sizing can approximately realize the offset as the amount of negative offset is only 1 level of intrinsic buffer delay or D^{buf}_{min}.

The execution of the above algorithm is illustrated with a representative example shown in Fig. 9(a). In this example, pin p of clock tree buffer B_1 is annotated with a negative offset which is equal to one clock tree level. Assuming that restructuring is selected instead of sizing, the candidate acceptors are initially extracted and suppose B_6 driver pin is the best acceptor for p that can realize the offset most accurately. Then, the restructuring is applied by detaching B_1 from B_0's fanout and connecting it at B_6. The resultant clock tree is shown in Fig. 9(b).



(a) Clock tree hyper-net where p has negative offset of 1 clock tree level.



(b) Resultant clock tree hyper-net where the negative offset at p is realized by restructuring.

Figure 9: Negative offset realization example

3.3 Our Methodology

Algorithm 3 shows the steps of our methodology for clock tree resynthesis. Initially, the LP solver calculates the offsets in the clock tree. In the case that the offset at a pin is positive, a buffer chain is inserted according to the method-

ology presented in section 2.2 (Line 5). Otherwise, if the offset is negative, the slack manager is updated (Line 7) and then 'NORA' is used to realize the offset (Line 8).

Algorithm 3 Clock Tree Resynthesis

```
1: Calculate clock tree offsets, S_offset by LP solver;
2: Execute 'BUSlackParamCalculate' for all clock tree
   roots and operating modes;
3: for all (p, offset) ∈ S_offset do
4:   if offset > 0 then
5:     Insert route-aware buffer(s) at p;
6:   else
7:     Update slack manager;
8:     NORA(p, offset);
9:   end if
10: end for
```

4. EXPERIMENTAL RESULTS

We have implemented the algorithms presented in this work in C++ and ran it on a Linux machine with 16-Core 3GHz CPU and 256GB RAM. Table 1 presents the characteristics of 7 industrial designs using cutting-edge technology nodes (20-32nm), in terms of total number of cells (Column 2), number of scenarios (Column 3) and initial timing metrics after placement, clock tree synthesis and routing by an industrial tool. Columns 4, 5 and 6 respectively specify the TNS, worst negative slack (WNS) and failure-end-point (FEP) across all scenarios.

Table 1: Design Specification

Design	Cells (M)	Scenarios	TNS (ps)	WNS (ps)	FEP
A	0.35	5	-789723	-4433	1907
B	0.62	8	-1586320	-414	12850
C	0.62	8	-82529	-218	1262
D	0.7	8	-1129784	-6433	2408
E	0.85	1	-8032671	-1483	17491
F	1.17	5	-8968128	-6394	43938
G	2.03	6	-4289746	-15418	31946

Table 2 presents the results of our approach. Columns 2-6 exhibit that if the LP solver is constrained to use only one level of negative offset and none of positive ones, then an average improvement of 15.85%, 1.05% and 11.64% is achieved in TNS, WNS and FEP respectively with average clock tree area overhead less than 2%. If three positive offset levels are allowed as well (Columns 7-11), then an average improvement of 56.68%, 12.04% and 41.82% in TNS, WNS and FEP respectively is achieved with an average clock tree area overhead of 26.17%.

Results show that negative offset realization does not increase the clock tree area significantly, as it is only gate upsizing which introduces area in this case and this reinforces our claim of area-efficient negative offset implementation. If positive offsets are allowed as well, the area overhead increases on average to 26.17% as positive offsets are typically realized by introducing delay chains comprised of multiple buffers. *The aforementioned percentage in area increase is in terms of buffers/inverters/combinational elements in clock tree network only and this does not include sequential leaf cells and data path combinational logic*, which dominate the total area of the design. So if we consider the total design area or even include the registers, the percentage increase would be negligible. For instance, for design 'E', the percentage increase in area in clock tree is maximum (55%), but *if we consider the total area of the design, the percentage area increase is less than* 1%.

With respect to the timing optimization, using only negative offsets suffices to reduce TNS for designs D and G by more than 30%. On the contrary, TNS improvement for designs E and F is below 10%. WNS is almost not reduced, as the realized offsets correspond to a single clock tree level which is a relatively small portion of WNS. FEP reduction follows the corresponding reduction of TNS for all the designs but A and B, for which FEP reduction is significantly smaller than the one of TNS.

In the case that three levels of positive offset are allowed as well, TNS reduction reaches 56.68% on overage, with most of the designs exhibiting TNS reduction by more than 62%. WNS is improved more when compared to only using a single level of negative offset as offset values now span from -1 to 3 levels. FEP reduction again follows the TNS reduction, with designs A and D exhibiting significantly less FEP optimization compared to TNS.

It should be stressed that for designs 'B' and 'D', besides TNS, THS is optimized as well, by 88% and 15% respectively with positive and negative offset realization and by 14.5% and 13% respectively with only negative offset realization (not mentioned in Table 2). For rest of the designs, hold corner analysis is not enabled. For design 'D', compared to the case of realizing only negative offsets, TNS/FEP improvement decreases while realizing both positive and negative offsets, but WNS and THS improvement is more.

The biggest design in this benchmark suite contains more than 2M cells and it has 6 scenarios. Our approach achieves 62% improvement in TNS with 11% overhead in clock tree area. Runtime for this benchmark is less than 7 hours, which is quite reasonable. However, it is counter-intuitive that run time is high in few designs ('C' and 'G') for realizing only negative offsets than for realizing both positive and negative offsets. This is due to the behavior of the LP engine, as for those designs the total number of negative offsets to be realized in the case where only negative offsets are allowed is more than the total number of offsets when both positive and negative ones are allowed.

5. DISCUSSION AND FUTURE WORK

In Fig. 2, we place the block of our methodology just before the post-CTS timing closure. Nevertheless it is worth mentioning that this is not the limitation and we can perform the clock tree resynthesis after the post-CTS data-path optimizations as well. But then the post-CTS data path optimizations would cost a significant area/power penalty and the potential of our approach to recover timing with minor area overhead in the design would not have been fully exploited. Furthermore, our approach can be suitably used for reducing design frequency as well by targeting aggressive clock cycle period. It should also be noted that although the realization of bounded negative offsets is feasible for deep clock-trees, the scope to introduce any negative offset might be limited for short-depth trees. In that case, our approach can tackle this situation by only realizing the positive offsets (by running the LP solver accordingly), but that would introduce larger area overhead in the clock tree.

We plan to extend this framework to improve on the area overhead in the clock tree. We can see that the area over-

Table 2: Timing metric improvement in industrial designs by our approach

Design	Only Negative Offset Realization					Positive and Negative Offset Realization				
	% TNS Imprv.	% WNS Imprv.	% FEP Imprv.	% Clock Tree Overhead	Run time (min)	% TNS Imprv.	% WNS Imprv.	% FEP Imprv.	% Clock Tree Overhead	Run Time (min)
A	10.70	-0.13	5.61	2.56	43	77.65	1.20	39.54	20.10	46
B	11.67	0.24	3.61	7.33	175	56.25	0.97	47.32	47.09	189
C	13.35	0.92	9.75	1.05	178	76.62	49.08	57.84	8.63	140
D	32.80	2.64	25.46	1.11	125	31.58	18.51	17.57	11.51	129
E	2.24	2.83	2.20	1.36	98	69.79	10.05	44.43	54.98	306
F	5.91	0.75	7.31	0.17	161	22.80	0.72	35.69	29.78	250
G	34.30	0.08	27.54	0.04	410	62.09	3.80	50.33	11.12	368
Average	15.85	1.05	11.64	1.95	-	56.68	12.04	41.82	26.17	-

head in the clock tree is mainly due to the positive offset realization. It should be noted that restructuring might not be helpful in realizing positive offset at any pin as the place-holders for offsets are typically leaf-level gates/buffers and so it is difficult to find an acceptor in the clock tree which can match the desired arrival time of the pin on restructuring. But we can consider the partial realization of the positive offsets, while realizing the negative offsets so that the size of the buffer to be inserted for realizing positive offsets decreases and area overhead improves. For instance, when we choose potential acceptor for realizing negative offset, a priority can be given (by modifying the cost function in Algorithm 2) to the acceptors which have place-holders (driver pins) for positive offsets in its TFO cone as the restructuring would result some delay in clock arrival for those pins, thereby realizing the positive offsets partially. It will be also interesting to study the impact of OCV derates on our result. Incorporating buffers (to realize positive offsets) might have adverse OCV impact, and on the other hand as restructuring (to realize negative offsets) involves moving clock elements upward (as discussed in Section 3.2), the chance of common paths between launch flop and capture flop might increase, leading to improve OCV due to common-path-pessimism-removal (CPPR) [26].

6. CONCLUSION

This work introduces algorithms which significantly improve timing metrics in large-scale industrial designs under MCMM scenarios. To our best knowledge this is the first work to implement a feasibility aware clock scheduling, realized by solving a constrained LP problem globally, and using the clock tree elements as place holders for the resultant offsets. Our approach has achieved an average TNS improvement of 57% in industrial designs with an average overhead of 26% in clock tree area. We have proposed to extend our current framework to improve in clock tree area overhead. In the future we plan to study the impact of OCV derates on our approach and examine the space between solutions with only negative offsets and that with both negative and positive offsets by using area and power bounds.

7. REFERENCES

[1] R. Tsay, "Exact zero skew clock routing algorithm," *Computer Aided Design of Integrated Circuits and Systems*, pp. 242–249, 1993.

[2] K. D. Boese and A. B. Kahng, "Zero skew clock-routing trees with minimum wirelength," *ASIC Conference and Exhibit*, pp. 17–21, 1992.

[3] J. L. Tsai, T. H. Chen, and C. C. Chen, "Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing," *Computer Aided Design of Integrated Circuits and Systems*, pp. 565–572, 2004.

[4] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. on Computers*, pp. 945–51, 1990.

[5] R. Deokar and S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," *ISCAS*, pp. 407–10, 1994.

[6] L. F. Chao and H. M. Sha, "Retiming and clock skew for synchronous systems," *ISCAS*, pp. 283–86, 1994.

[7] I. S. Kourtev and E. G. Friedman, "Clock skew scheduling for improved reliability via quadratic programming," *ICCAD*, pp. 239–43, 1999.

[8] X. Liu, M. C. Papaefthymiou, and E. G. Friedman, "Maximizing performance by retiming and clock skew scheduling," *DAC*, pp. 231–36, 1999.

[9] V. Nawale and T. W. Chen, "Optimal useful clock skew scheduling in the presence of variations using robust ILP formulations," *ICCAD*, pp. 27–32, 2006.

[10] Y. Taur and D. Buchanan, "CMOS scaling in nanometer regime," *Proc. IEEE*, pp. 486–503, 1997.

[11] V. Mehrotra and D. Boning, "Technology scaling impact of variation on clock skew and interconnecet delay," *Interconnect Tech. Conference*, pp. 4–6, 2001.

[12] A. Rajaram and D. Z. Pan, "Robust chip-level clock tree synthesis for SOC designs," *DAC*, pp. 720–723, 2008.

[13] S. Jilla, "Multi-corner multi-mode signal integrity optimization," *EDA Tech Forum*, 2008.

[14] D. Lee and I. L. Markov, "Obstacle-aware clock-tree shaping during placement," *ISPD*, pp. 123–130, 2011.

[15] Y. Wang, Q. Zhou, X. Hong, and Y. Cai, "Clock-tree aware placement based on dynamic clock-tree building," *ISCAS*, pp. 2040–43, 2007.

[16] K. Rajagopal, T. Shaked, Y. Parasuram, T. Cao, A. Chowdhury, and B. Halpin, "Timing driven force directed placement with physical net constraints," *ISPD*, pp. 60–66, 2003.

[17] Y. Liu, R. S. Shelar, and J. Hu, "Delay-optimal simultaneous technology mapping and placement with applications to timing optimization," *ICCAD*, pp. 101–106, 2008.

[18] S. W. Hur, A. Jagannathan, and J. Lillis, "Timing driven maze routing," *TCAD*, pp. 234–241, 2000.

[19] K. Sato, H. E. M. Kawarabayashi, and N. Maeda, "Post-layout optimization for deep sub-micron design," *DAC*, pp. 740–745, 1996.

[20] Y. P. Chen, J. W. Fang, and Y. W. Chang, "ECO timing optimization using spare cells," *ICCAD*, pp. 530–535, 2007.

[21] M. Ni and S. O. Memik, "A revisit to the primal-dual based clock skew scheduling algorithm," *ISQED*, pp. 755–764, 2010.

[22] S. M. Burns, *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1991.

[23] J. Lu and B. Taskin, "Post-CTS clock skew scheduling with limited delay buffering," *International Midwest Symposium on Circuits and Systems*, pp. 224–227, 2009.

[24] W. Shen, Y. Cai, W. Chen, Y. Lu, Q. Zhou, and J. Hu, "Useful clock skew optimization under a multi-corner multi-mode design framework," *ISQED*, pp. 62–68, 2010.

[25] V. Ramachandran, "Functional skew aware clock tree synthesis," *ISPD*, 2012.

[26] J. Bhaskar and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Springer, 2009.