Clock Tree Resynthesis for Multi-Corner Multi-Mode Timing Closure

Subhendu Roy, *Student Member, IEEE*, Pavlos M. Mattheakis, Laurent Masse-Navette, and David Z. Pan, *Fellow, IEEE*

***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: 1) calculate clock skew at sequential elements before the actual tree is synthesized and 2) do not account for the implementability of the calculated schedules at the later stages of design cycle. In this paper, we propose a novel clock tree resynthesis methodology which 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. We demonstrate the effectiveness of the offsets at the output pins of the leaf-level clock drivers in comparison to the traditional clock scheduling in the clock pins of the flip-flops due to the better implementability and lesser area overhead and present an algorithm to accurately realize these offsets in the clock tree. Experimental results on large-scale industrial designs demonstrate that our clock tree resynthesis methodology achieves respectively 57%, 12%, and 42% average improvement in total negative slack, worst negative slack, and failure-end-point with an average overhead of 26% in clock tree area. We also experimentally study the impact of on-chip-variation-derates on our approach in terms of the timing metric improvement and clock tree overhead.**

***Index Terms*—Clock skew scheduling, clock tree synthesis (CTS), engineering change order (ECO), multi-corner, multi-mode (MCMM), useful skew.**

I. Introduction

CLOCK skew is the difference in clock arrival times (ATs) at different sequential elements in the clock-distribution network. A lot of work has been done in the past to minimize clock-skew [1]–[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]–[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. Rajaram and Pan [12] have proposed an algorithm for chip-level clock tree synthesis (CTS) to tackle the clock divergence issue in different corners. However, it does not take into account the timing information on data path for CTS. Additionally, 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] and [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]–[18]. But in spite of all these efforts, timing violations still exist after detail routing in MCMM designs, especially for the advanced technology nodes. 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 toward 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. Ni and Memik [21] present a fast primal-dual based approach for minimal clock period, improving over Burns' algorithm [22] in run-time complexity. Nawale and Chen [9] even tackle the clock scheduling problem in presence of

Manuscript received July 19, 2014; revised October 8, 2014 and December 10, 2014; accepted January 4, 2015. Date of publication January 20, 2015; date of current version March 17, 2015. This paper was recommended by Associate Editor J. Hu.

S. Roy and D. Z. Pan are with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712 USA (e-mail: subhendu@utexas.edu).

P. M. Mattheakis and L. Masse-Navette are with Mentor Graphics, Grenoble 38100, France.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2394310

[1] Any mode/corner combination.

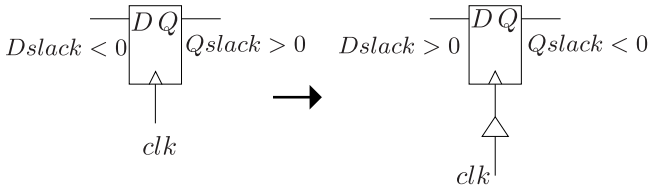

Fig. 1. Buffer insertion to mitigate D-slack violation can cause Q-slack violation.

process variations by Integer Linear Programming formulation. But the issues with these approaches are: 1) actual implementation of that clock scheduling is difficult to achieve in real designs, especially at later design stages and 2) they are unaware of MCMM scenarios.

Lu and Taskin [23] formulate a Linear Programming (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 paper 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 paper are: 1) 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; 2) area-overhead in clock tree is high as it works only at leaf-level; and 3) speeding up clock arrival to fix Q-slack violations by only removing buffer is hardly realizable in practice to be discussed in Section II-D (Fig. 8).

To tackle these issues, a novel clock tree resynthesis methodology is presented in this paper. We develop an LP solver based on [25] to estimate MCMM-aware clock scheduling. The notion of branch level clock scheduling is introduced which, instead of estimating clock schedule at the leaf level registers, considers offsets in clock arrival at the clock tree driver pins of any placed design with already synthesized and routed clock tree. Experimental runs with the LP solver on industrial designs manifest the advantages of this granularity reduction from leaf level to branch level clock scheduling in terms of better implementability and lesser area/power cost without significant degradation in the scope to improve timing.

We illustrate in Section II-C 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 (NOR) 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.

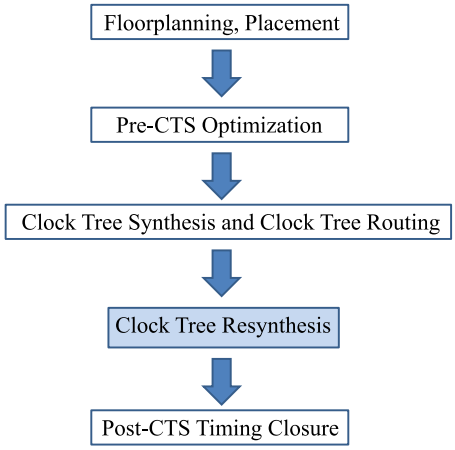

Fig. 2. Our methodology in a conventional back-end flow.

Fig. 2 illustrates the steps of a conventional back-end flow into which our methodology for clock tree resynthesis [26] can be integrated. 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, the more we advance toward timing closure by clock tree resynthesis, the better are the savings in terms of area/power. The key contributions of this paper are as follows.

1) 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.
2) A novel algorithm which is non-intrusive and area-efficient is presented that realizes negative offsets.
3) 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.
4) The impact of on-chip-variation (OCV)-derates [27] on our clock tree resynthesis methodology is experimentally studied in terms of timing improvement as well as clock tree overhead.

The rest of this paper is organized as follows. Section II illustrates the concept of feasibility aware clock scheduling in presence of MCMM scenarios using an LP solver. Section III presents our novel algorithm to realize the negative offsets

predicted by the LP solver and the overall methodology for clock tree resynthesis. Section IV presents the experimental results of our approach for industrial designs. Section V discusses about the applicability of our methodology and future work with the conclusion in Section VI.

II. 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 paper, it is imperative to address concisely how the LP solver tackles various modes and corners in the design. Then, we explain the notion of branch level clock scheduling and why it is beneficial for modern space-constrained industrial designs in comparison to leaf level clock scheduling by running experiments on designs with the LP solver. Next the approach for positive offset realization (POR) is illustrated. Finally, the issues in realizing negative offsets are discussed and offset bounds are introduced to tackle those issues. So our clock scheduling technique not only tries to maximize the gain in terms of timing metric improvements or lesser area overhead, but also predicts clock schedules which are feasible to implement in industrial designs. We call this as feasibility aware clock scheduling.

A. 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, OCV derates [27] 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]. It can calculate the positive and negative offsets at the leaf-clock pins or 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 $D^{\text{buf}}_{\text{min}}$. 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 $c_i \times d_{\text{off}}$ in the i^{th} corner. We can specify the range of these offsets by constraining minimum level ($L^{\text{off}}_{\text{min}}$) and maximum level ($L^{\text{off}}_{\text{max}}$). For instance, suppose the $D^{\text{buf}}_{\text{min}}$ of a design is 60 ps and we specify $L^{\text{off}}_{\text{min}} = -2$ and $L^{\text{off}}_{\text{max}} = 3$, then the LP solver will estimate the offsets of values -120, -60, 60, 120, 180 ps along with a prediction of timing improvement. The calculation and realization of the offsets are tightly coupled in this paper. Additionally, the realization maintains the timing profile of the parts of the design which should not be affected.

[3]Here, THS signifies total negative hold slack.

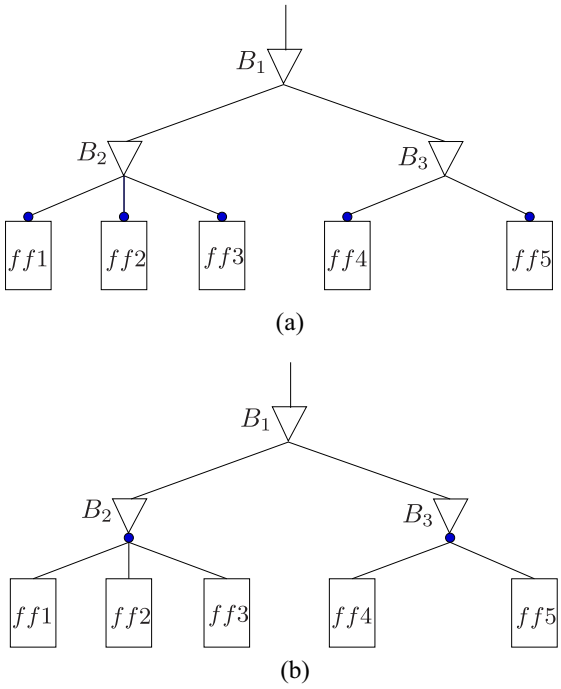

Fig. 3. Granularity reduction in clock scheduling. (a) Leaf level clock scheduling. (b) Branch level clock scheduling.

B. Leaf Level Versus Branch Level Clock Scheduling

Several works [7], [9], [23], [24] have focussed on leaf-level clock scheduling, which means the clock arrivals at the clock input pins of the flip-flops (FFs) are determined in order to optimize time period or improve the timing metrics. On the contrary, we define branch level clock scheduling as the determination of clock-arrival at the output pins of driving clock buffers/gates in the clock network, the objective being the same, i.e., improving TNS/THS. Our LP solver can handle both types of scheduling by computing offsets at the FF clock pins [Fig. 3(a)] or output pins of the clock drivers [Fig. 3(b)]. It should be stressed that the offset computation by the LP-solver is followed by a post-processing step of offset factorization, i.e., if one buffer drives buffers with similar offsets, then offsets are factorized and shifted upward. For instance, the offsets at the output pins of B_5, B_6, and B_7 are respectively 100, 50, and 50 in Fig. 4 and in that case, an offset of 50 would be assigned to the output pin of B_2. But the offset at the output pin of B_3 can not be factorized since there is no prescribed offset at the output of B_4. However, as the offsets computed by the LP-solver are typically sparse, the scope of this offset factorization is very limited.

Due to the coarser granularity in the clock scheduling, leaf-level clock scheduling can potentially provide more improvement in timing metrics, but at the cost of high area overhead. To demonstrate this, we take several industrial designs and run our LP solver for leaf level and branch level clock scheduling.

Table I shows the result of this experiment. Column 2 presents the number of leaf-pins (or FFs) in each design. The predicted TNS improvement and the number of offset counts for leaf-level (branch level) clock scheduling are represented by columns 3 and 4 (6 and 7). Offset count here refers to the

TABLE I
LP SOLVER PREDICTION FOR LEAF LEVEL VERSUS BRANCH LEVEL CLOCK SCHEDULING

Design	# of Leaf pins (NLP)	Leaf-level			Branch-level			
		% TNS imprv.	Offset count (OCT)	$\frac{OCT}{NLP}\times 100$	% TNS imprv.	Offset count (OCT)	$\frac{OCT}{NLP}\times 100$	Optimized leaf-pins
1	34399	30.8	5483	15.9	16.3	229	0.7	2255
2	53683	89.0	6642	12.4	85.1	286	0.5	9461
3	79797	84.7	5069	6.4	84.3	120	0.2	8100
4	143092	38.4	58783	41.1	36.1	2174	1.5	51075
Average		60.7		19.0	55.5		0.7	

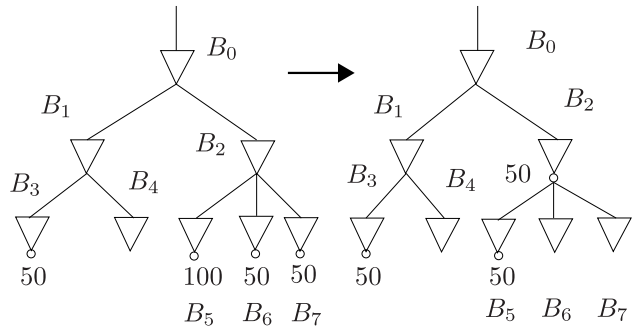

Fig. 4. Offset factorization.

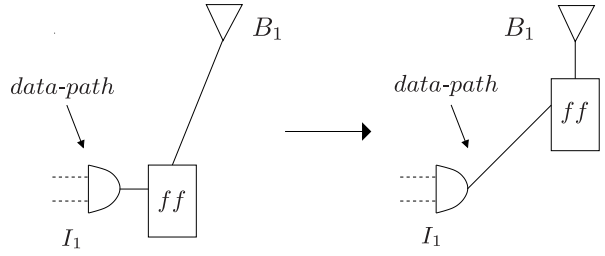

Fig. 5. Moving flops can affect data-path timing profile.

total number of offsets computed by the LP-solver. In case of leaf-level clock scheduling this offset count will be same as the total number of optimized leaf-pins. But for branch-level clock scheduling, one offset at the branch level would affect clock-scheduling at all the flops transitively driven by the corresponding clock-element. For instance, offset at the output pin of B_2 would change the clock arrival at the flops *ff*1, *ff*2, and *ff*3 (Fig. 3). Columns 5 and 8 calculate the percentage ratio of offset count and total number of leaf-pins for leaf-level and branch-level clock scheduling, respectively. Column 9 presents the total number of leaf-pins where the clock-arrival would be affected or optimized by the branch-level clock scheduling.

We observe that the predicted TNS improvements in case of leaf-level clock scheduling are slightly better than that in case of branch-level clock scheduling, except the first design, where leaf-level scheduling can achieve much better TNS improvement. A reason could be that at the transitive fanout (TFO) of a branch pin there is a race of the scheduling direction i.e., some flops have positive *D* and negative Q-slack and some others have the opposite, hence moving the branch optimizes some flops but degrades others. On average, leaf-level clock scheduling can achieve 60.7% improvement in TNS by adjusting clock-schedules of 19% leaf clock pins, whereas branch level clock scheduling achieves 55.5% improvement in TNS with $19/0.7 = 27.1\times$ lesser number of offset realization. The following are the issues in realizing offsets at the leaf-level.

1) To realize positive offsets (Section II-C), we need to introduce buffering to delay the clock arrivals. Since the number of offsets for leaf-level clock scheduling is very high, this would introduce a significant area/power cost due to the large number of buffers to be incorporated, and secondly it is very difficult to place/route so many extra buffers in the modern space-constrained designs. So post-CTS delay buffering at the leaf level [23], [24] to improve timing is practically difficult to realize for large-scale industrial designs.
2) Branch-level negative offsets (i.e., speeding up clock arrival at the output pins of clock buffers/inverters) may be realized by sizing or moving the driving buffers to be discussed in Section II-D. However, this can not be used to realize leaf-level negative offsets as typically one clock buffer drives multiple FFs and moving/sizing the driving buffer would affect the timing profile of all the sinks driven by that buffer. A possible approach to realize negative offsets at the leaf-level clock pins could be to place the FFs closer to its driving buffers/inverters. But the movement of the flops would not only affect the clock arrival at other leaf-pins, but also can have catastrophic effect on the timing profile of the data-path. The situation is illustrated in Fig. 5, where the clock pin and the data pin of the FF are driven by the clock buffer B_1 and the combinational gate I_1, respectively. So if the flop is moved physically closer to B_1 to speed-up the clock arrival at the clock pin of FF, it might move further from I_1, and in that case, the delay of the data-path ending at FF would increase. Similar effect can also happen for the data-paths starting at FF. This is detrimental to timing closure since the clock scheduling has been performed assuming no change in the data-path timing profile. However, in case of branch level clock scheduling, movement of clock buffers/inverters to realize negative offsets does not affect the data-path timing profile.

Driven by the above findings, this paper focuses on branch level clock scheduling which would be feasible to realize, and then physically implement the offsets predicted by the LP-solver.

C. POR

POR is accomplished by inserting route aware delay elements. Fig. 6 illustrates the realization of a positive offset

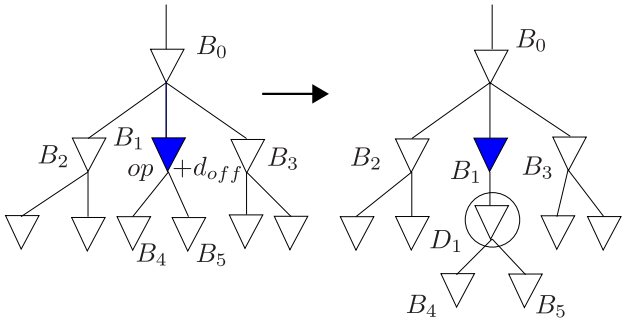

Fig. 6. POR.

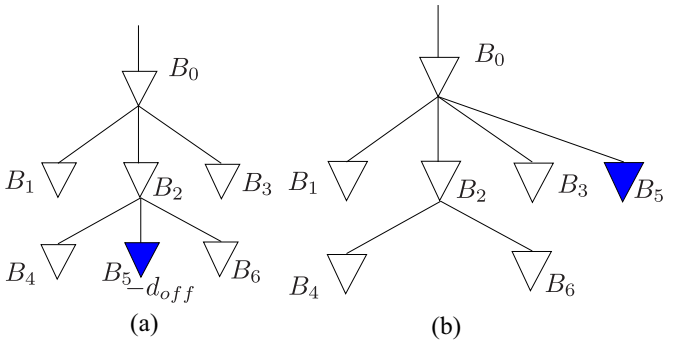

Fig. 7. NOR. (a) B_5 output pin should be accelerated by d_{off}. (b) B_5 is shifted one level upward the tree.

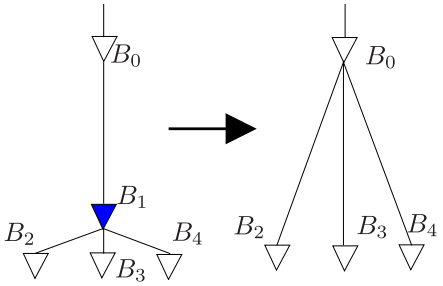

Fig. 8. Buffer removal might not be effective in realizing negative offset.

at the output pin of the 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 POR 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. 6 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 in terms of timing profile. However, this will introduce clock tree area overhead due to the insertion of buffers.

D. Issues in NOR

The NOR 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. 7. 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 AT of all clock tree nodes at B_2's 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 upward cell B_5. In this case, the AT to FFs at the TFO of B_5 is reduced but multiple side effects alter the ATs 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.

Shen *et al*. [24] have mentioned that clock arrivals could be accelerated by removing the corresponding buffer B_1 (Fig. 8). But this can be safe 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: 1) B_0 is now driving 3 buffers instead of 1, *viz.* B_2, B_3, and B_4 and 2) 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 might get delayed at the TFO cone of B_0.

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 upward the tree. As a consequence, more FFs downward the tree will be affected increasing the probability of degrading the timing instead of optimizing it.

E. 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 NOR 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

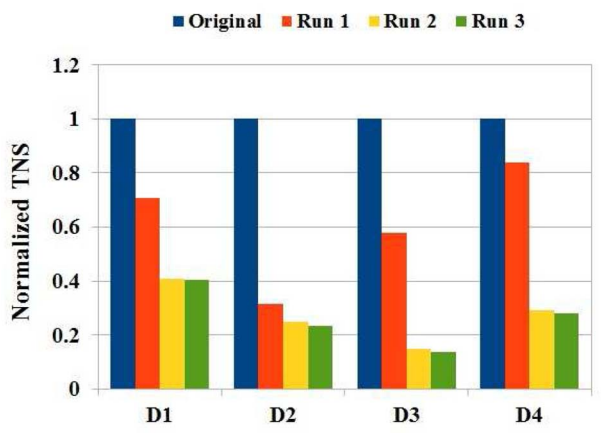

Fig. 9. Normalized TNS prediction by LP solver for industrial designs.

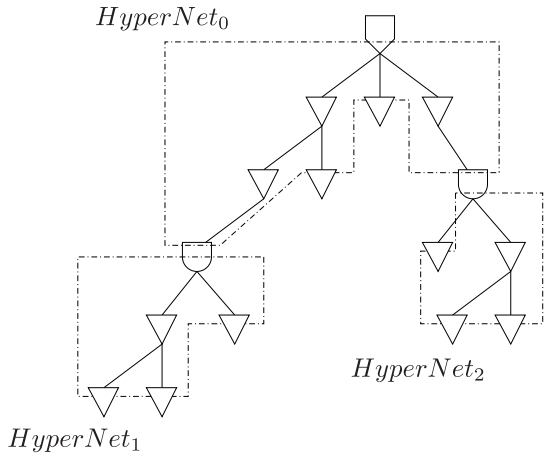

Fig. 10. Clock tree decomposition to hyper-nets.

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. 9, where TNS predicted by the solver for each one of the three aforementioned experiments are normalized with respect to the original TNS of each design, which is the TNS after placement, CTS 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 paper as the solver will be bounded to produce solutions with a single level of negative offset.

III. Clock Tree Resynthesis

Section II-E showed that significant TNS gains can be enjoyed if pins which can carry offsets are bounded to -1 level ($L_{\min}^{\text{off}} = -1$). In this section, we present a methodology for clock tree resynthesis to improve timing in a routed clock 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. 10 demonstrates a clock tree comprised of three hyper-nets. The datapath logic and the enable signals at the clock tree clock gates are omitted in the figure.

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 TFO it is annotated as a potential acceptor of pins with negative offset. In this way, negative offsets are realized accurately without degrading the TNS. We develop an engine, called slack manager, which helps to extract the potential acceptors for pins with negative offset.

A. 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 ($\text{leafCells}_{\text{fo(pin)}}$). We define the following parameters.

1) $\text{Qslack}_{\text{sum}}(\text{pin})/\text{Dslack}_{\text{sum}}(\text{pin})$ = sum of the negative Q/D-slacks at $\text{leafCells}_{\text{fo(pin)}}$.
2) $\text{Qslack}_{\text{cnt}}(\text{pin})/\text{Dslack}_{\text{cnt}}(\text{pin})$ = count of $\text{leafCells}_{\text{fo(pin)}}$ having negative Q/D-slack.

These parameters are calculated recursively in a bottom-up fashion. Algorithm 1 presents the recursive procedure "BUSlackParamCalculate(pin, mode)", which stores the slack-parameters in any pin for all the corners active in that mode. Since these slack parameters depend on both modes and corners, we store these parameters per scenario, i.e., corner and mode combination. However, only mode but not corner is used as the argument for the procedure. This is because the recursive traversal of the clock-network in the procedure is decided by the parent–child relationship of the clock elements, and this parent–child relationship is invariant across different corners, but may vary with different modes due to the presence of clock-multiplexers in the clock network.

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 and 14) we set $\text{Qslack}_{\text{cnt}}$ to be 1 and $\text{Qslack}_{\text{sum}}$ 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 and 24). In our implementation, we have set this threshold to be 0, and thus these parameters, respectively estimates the count of $\text{leafCells}_{\text{fo(pin)}}$ with negative Q-slack and sum of negative Q-slacks of $\text{leafCells}_{\text{fo(pin)}}$.

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
```

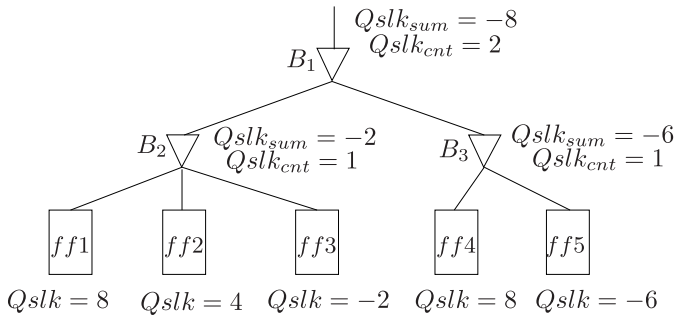

Fig. 11. Q-slack parameter calculation.

The execution of the algorithm is demonstrated with a representative example in Fig. 11. The output pins of the cells B_2 and B_3 have $\text{Qslack}_{\text{cnt}}$ equal to 1 due to the cells ff3 and ff5, respectively. B_1's output pin has $\text{Qslack}_{\text{cnt}}$ equal to 2 which results from the addition of its children's corresponding values. The $\text{Qslack}_{\text{sum}}$ values are calculated accordingly.

Similar calculation is done for D-slack parameters and it has not been shown in Algorithm 1 or Fig. 11 for brevity.

B. NOR Algorithm (NORA)

The slack manager exposes the space that is available for NOR in terms of slack. The "NORA" utilizes this space to: 1) accurately realize all negative offsets and 2) gain the improvement in TNS calculated by the LP solver.

Algorithm 2 Procedure to Realize a Negative Offset

```
1: Procedure NORA(p, offset);
2: scn ← getConstraintScenario;
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 ps 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;
```

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 $\text{Qslack}^{\text{eff}}_{\text{sum}}$ and $\text{Qslack}^{\text{eff}}_{\text{cnt}}$ and they are calculated according to the following formulas:

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

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

The effective D-slack values are calculated accordingly.

If $\text{Qslack}^{\text{eff}}_{\text{count}}$ is greater than $\text{Dslack}^{\text{eff}}_{\text{count}}$ for p_{par}, then it is preferable to reduce the load at p_{par}'s 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 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 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 $\times$ offset are disregarded (Line 8) as connecting p to them would not result to the desired AT 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 (Lines 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 Q-slack 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 ten 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 $\text{Qslack}_{\text{sum}}$ 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 one level of intrinsic buffer delay or $D^{\text{buf}}_{\text{min}}$.

The execution of the above algorithm is illustrated with a representative example shown in Fig. 12(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. 12(b).

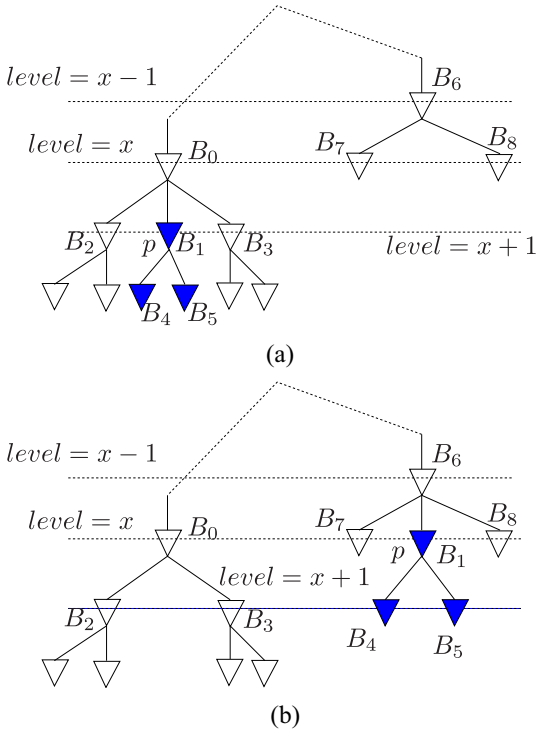

Fig. 12. NOR example. (a) Clock tree hyper-net where p has negative offset of one clock tree level. (b) Resultant clock tree hyper-net where the negative offset at p is realized by restructuring.

C. 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 methodology presented in Section II-C (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). Note that, the algorithm has been implemented into an industrial Placement & Routing tool, and so whenever the slack manager is updated and the D/Q-slack parameters are calculated, the timer embedded in the tool is incrementally called to always give correct D/Q-slack values.

IV. Experimental Results

We have implemented the algorithms presented in this paper in C++ and ran it on a Linux machine with 16-Core 3 GHz CPU and 256 GB RAM. Table II presents the characteristics of seven industrial designs using cutting-edge technology nodes (20–32 nm), in terms of total number of cells (column 2), number of scenarios (column 3) and initial timing metrics after placement, CTS and routing by an industrial tool. Columns 4–6, respectively specify the TNS, worst

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
```

TABLE II
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

negative slack (WNS) and failure-end-point (FEP) across all scenarios.

Table III 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 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 NOR does not increase the clock tree area significantly, as it is only gate up-sizing 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 clock tree area 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 positive offsets are allowed as well, TNS reduction reaches 56.68% on average, 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. 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 POR and NOR and by 14.5% and 13%, respectively with only NOR (not mentioned in Table III). 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 2 *M* cells and it has six scenarios. Our approach achieves 62% improvement in TNS with 11% overhead in clock tree area. Runtime for this benchmark is less than seven hours, which is quite reasonable. However, it is counter-intuitive that run time is high in a 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. Note that, this run-time includes computing the offsets by the LP-solver and realizing the offsets, followed by global and detail routing of the clock nets.

In Fig. 13 the percentage TNS improvements predicted by LP-solver and that after actual physical offset (positive + negative) realization are compared for each of the 7 designs. Corresponding to each design, the first column represents the % TNS improvement computed by the LP-solver that would have been achieved on exact realization of all offsets, and the second column represents the actual % TNS improvement by our offset realization algorithm. On average, our algorithm achieves 56.68% improvement in TNS while that predicted by the LP-solver is 63.58%. The discrepancy is because the physically implemented offsets are not exactly the same as those computed by the LP-solver, nevertheless we have achieved most of the timing improvements predicted by the LP-solver. It should be noted that the offsets computed by the LP-solver are bounded and discrete valued (Section II-A), and so may not give the optimum results. Thus the inexactness in offset implementation has benefited in case of design '*D*', where we got more timing improvements than predicted.

Table IV presents the results with only POR. An average improvement of 56.14% and 9.21% is achieved in TNS and WNS, respectively with an average clock tree overhead of 29.51%. In comparison to POR + NOR (PNOR), the timing improvements in this case are very close but with some additional clock tree overhead. For designs '*A*' and '*D*', PNOR is more effective both in terms of timing optimization and clock tree overhead than POR. In case of designs '*F*' and '*G*', timing

TABLE III
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	-

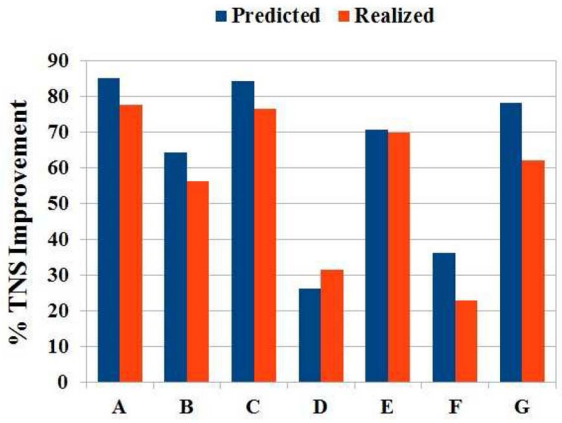

Fig. 13. Comparison of % TNS improvement between LP-solver prediction and after actual physical offset realization.

TABLE IV
POR

Design	% TNS Imprv.	% WNS Imprv.	% Clock Tree Overhead	Run Time (min)
A	74.22	0.31	23.39	39
B	56.97	-7.97	43.64	158
C	80.76	65.6	8.41	117
D	13.17	17.64	11.84	118
E	69.19	6.04	53.79	364
F	26.84	0.68	48.33	139
G	71.88	-17.84	17.15	523
Average	56.14	9.21	29.51	-

TABLE V
QOR COMPARISON

Design	Pos. Offset	Neg. Offset	Pos.+ Neg. offset
A	3.17	4.18	3.86
B	1.31	1.59	1.19
C	9.60	12.71	8.88
D	1.11	29.55	2.74
E	1.29	1.65	1.27
F	0.56	34.76	1.15
G	4.19	857.50	5.58

improvements for POR are more but with significant overhead in clock tree area. However, POR gives similar or better TNS improvement with less clock tree overhead when compared to PNOR for rest three designs.

The scope or effectiveness of NOR is limited in those designs due to the presence of short depth clock trees. In a few cases, the leaf-level integrating clock cells in the designs drive the sinks either directly or through few buffers. Since restructuring to realize negative offsets is performed within the scope of a hyper-net, and as the driver pin of a clock gate is treated as a hyper-root (as explained in Section III), it becomes infeasible to realize negative offsets by restructuring in those situations. A possible solution to this issue may be to make the LP-solver aware of this, i.e., identify these cases beforehand, and feed this information to the LP-solver such that it is prevented to assign any negative offset on those driver pins.

In order to analyze the results of POR, NOR, and combined offset realization, we define quality of results (QoR) as the ratio of % TNS improvement to the % clock tree area overhead. QoR for the three cases are compared across all designs in Table V. We can see that combined offset realization gives better "QoR" in 4 out of 7 designs in comparison to POR, whereas NOR always gives the best QoR due to its very low clock tree area overhead.

Although the comparison between POR and PNOR shows similar timing improvement, there are several advantages of PNOR or specifically the NOR. Firstly, as discussed earlier, the overhead in clock tree area does not contribute much to the total design area. But clock nets typically switch faster than the signal nets, and consequently, 30%–70% of total dynamic power of the design is consumed in the clock network [28]. So any sort of area-overhead in clock network will contribute significantly in increasing the total dynamic power of the design. The parameter QoR is thus an indicator of timing versus power trade-off, and better QoRs in NOR signify more power-efficient solutions. For designs like '*D*' and '*F*', we get more than 2× improvement in QoR for PNOR compared to POR. Secondly, NOR, but not POR, is the preferable way to fix violations in certain cases. For instance, a clock-gate may drive (directly or transitively) hundreds of FFs in modern designs. If there are setup violations in the timing paths associated with the path group through that clock gate, then it would be convenient to fix those by speeding up the clock arrival at the clock pins of the flops which are at the transitive fan-in cone of the clock gate, since these flops are relatively fewer compared to those in the TFO cone of the clock gate.

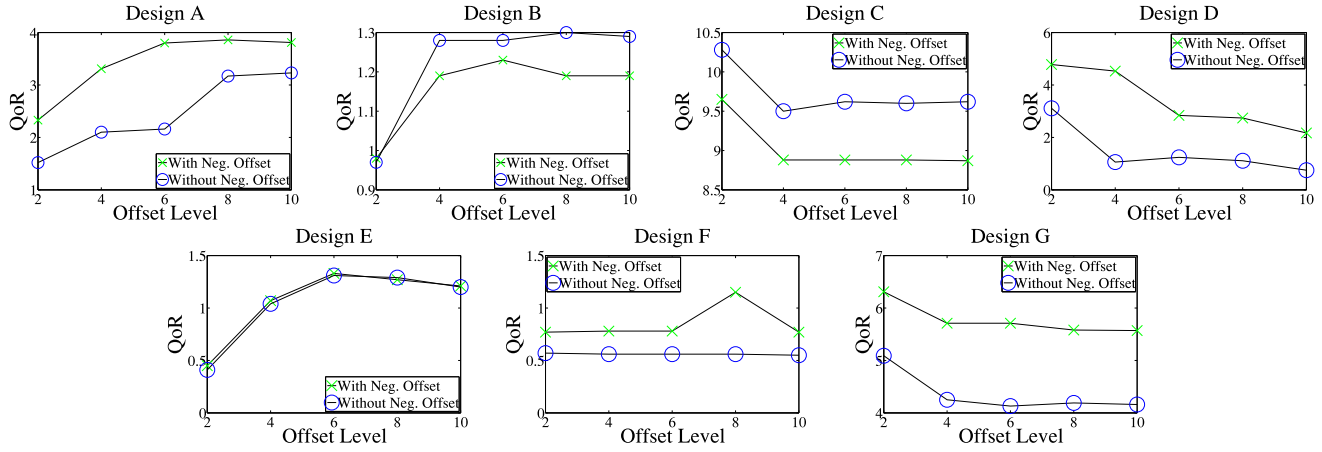

Fig. 14. QoR versus offset level.

In addition, if there is any timing violation pertaining to the timing path between a FF and a primary output, it can also be fixed by speeding up the clock arrival at the clock pin of the flop. Finally, POR is more prone to OCV compared to NOR to be discussed later.

A. Impact of Offset Levels on QoR

Next, we study the impact of offset levels on QoR. We run our algorithm by varying the positive offset levels from 2 to 10 in steps of 2, and negative offset levels with 0 and 1. Fig. 14 shows the plot of QoR with different positive offset levels for two cases, with (PNOR) and without (POR) NOR. PNOR achieves better QoR in designs '*A*', '*D*', '*F*', and '*G*', and POR achieves better QoR in design '*C*' than their respective counterparts for all positive offset levels. QoR for PNOR and POR are comparable for design '*E*'. For design '*B*', POR achieves better QoR except the case for offset level = 2. There is no general trend for dependence of QoR on offset levels, and it varies from design to design.

B. Impact of OCV-Derates on Clock-Tree Resynthesis

In any industrial timer, typically two types of clock arrivals are calculated for each pin, namely early arrival and late arrival to accommodate OCV, and are characterized by OCV-derates [27]. For instance, suppose one level of buffer delay is 50 ps and OCV-derates are 0.97–1.05. Then the early delay (late delay) for that level would be $50 \times 0.97 = 48.5$ ps ($50 \times 1.05 = 52.5$ ps). So as OCV-derates increase, i.e., derates become more apart from 1.00, it becomes difficult to achieve the timing closure under worst-case scenario. Consequently with increase in OCV-derates, TNS of the designs as well as the count of offsets predicted by LP solver to improve the timing metrics increase. Fig. 15 shows the TNS changes of 5 industrial designs with different OCV-derates. We consider three situations: 1) 1.00–1.00, i.e., no OCV; 2) 0.95–1.05; and 3) 0.90–1.10 and TNS for each case is normalized with respect to the TNS without any OCV. We can observe that as OCV-derates increase, TNS for the designs rise up significantly. For D_1, the increase in TNS is very high as the absolute TNS is very small in absence of OCV. Similar trend is exhibited in

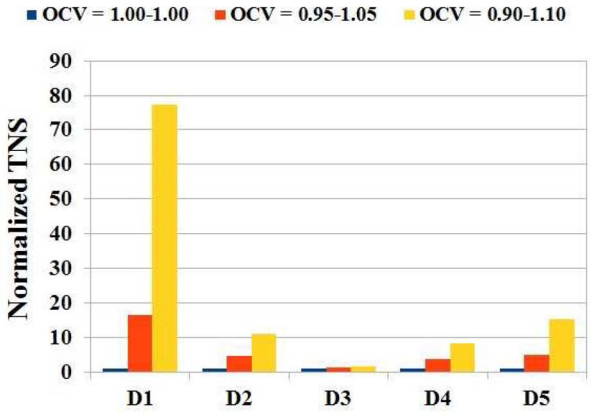

Fig. 15. TNS of designs increase as OCV-derates increase.

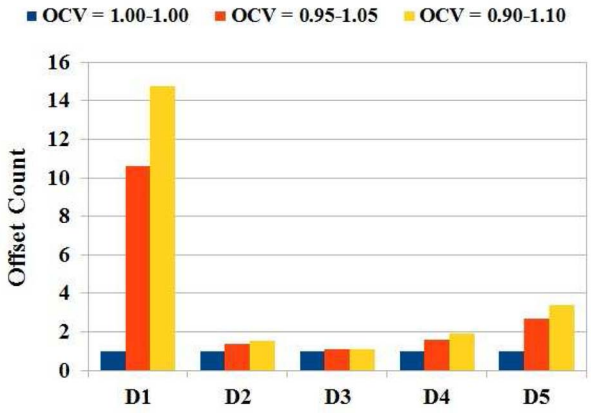

Fig. 16. Offset counts increase as OCV-derates increase.

Fig. 16, where the offset counts for different OCV-derates are normalized with respect to that with no OCV. Note that the designs used in this experiment are not exactly same as those used in Table III, with D_3 and D_4 corresponding to *E* and *F*, respectively. D_5 is the biggest design in this suite with around 216 k FFs, 2.16 *M* cells, and four scenarios. The run-time for this design for OCV-derate 0.90–1.10 (including computing offsets by LP-solver, offset realization followed by global and detail routing of the clock nets) is around 12 h.

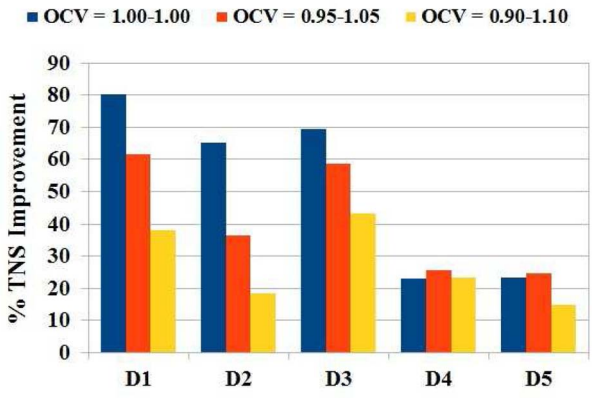

Fig. 17. Percentage TNS improvement with different OCV-derates.

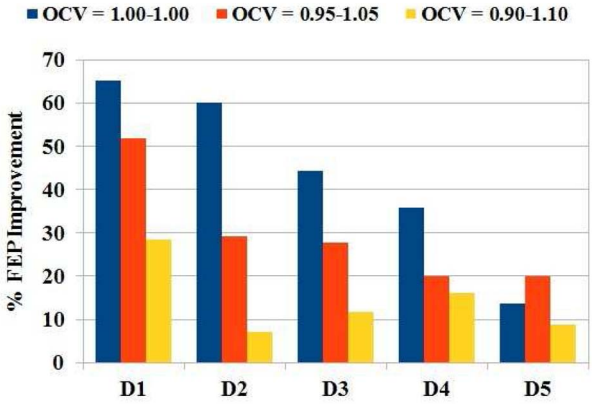

Fig. 18. Percentage FEP improvement with different OCV-derates.

Next, we compare the improvement of the timing metrics and clock tree overhead with different OCV-derates. Fig. 17 shows the percentage TNS improvement of the designs under three OCV-derates. We observe that percentage TNS improvement is typically maximum with no OCV, and as OCV derates increase it decrease. However, for *D*5 and *D*4, this behavior is not monotonic. On average, the percentage TNS improvements for the OCV-derates 1.00–1.00, 0.95–1.05, and 0.90–1.10 are respectively 52.3%, 41.3%, and 27.6%. Similar trend is also observed in case of FEP improvement (Fig. 18). The corresponding percentage improvements are 43.8%, 29.7%, and 14.4%, respectively. It should be stressed that these reported improvements are after clock tree resynthesis or realization of the offsets, but not the improvements predicted by the LP-solver. The degradation in timing improvement with OCV-derates is due to several reasons. Firstly, the predicted TNS improvement by LP solver decreases with increase in OCV-derates. This is because as OCV-derates increase, it becomes more and more difficult to achieve the timing closure. Secondly, incorporating the buffers to realize positive offsets increase the number of levels and path-length in the clock tree and consequently, OCV-impact becomes more severe. Finally, restructuring might benefit or aggravate the OCV-impact depending on the increase or decrease of the common path between the clock pins of the launch and capture flops due to the common-path-pessimism-removal.

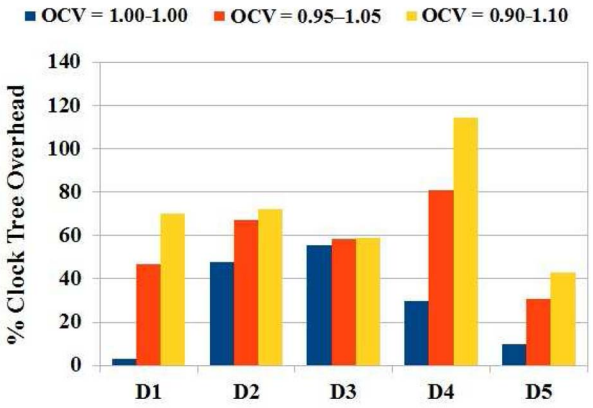

Fig. 19. Percentage clock tree overhead with different OCV-derates.

Fig. 19 shows the percentage clock-tree overhead for the designs with different OCV-derates. With increase in OCV-derates, the clock tree overhead increases and the plot is similar to that of offset counts versus OCV-derates (Fig. 16). This is intuitive, as number of offsets increase, more buffers will be introduced and more clock tree restructuring would be performed causing increase in clock tree overhead. The percentage clock tree overhead for the OCV-derates 1.00–1.00, 0.95–1.05, and 0.90–1.10 are respectively 29.1%, 56.9%, and 71.7%. Note that, this clock tree overhead is calculated based on the area of the clock tree buffers/inverters etc. and does not include the sequential elements. The maximum overhead is observed for *D*4 with OCV-derate 0.90–1.10, but it is less than 1%, if we consider the total area of the design, however, in terms of power consumption the overhead can be significant due to the higher switching activity in the clock network.

V. Discussion

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.

We plan to extend this framework to improve on the area overhead in the clock tree. We can see that the area overhead in the clock tree is mainly due to the POR. 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 AT 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 placeholders (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.

VI. Conclusion

This paper 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 define the QoR metric and study its dependence on offset levels. We also study the impact of OCV-derates on our approach and have proposed to extend our current framework to improve in clock tree area overhead. In the future, we plan to examine the space between solutions with only negative offsets and that with both negative and positive offsets by using area and power bounds.

References

[1] R. Tsay, "Exact zero skew clock routing algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 2, pp. 242–249, Feb. 1993.

[2] K. D. Boese and A. B. Kahng, "Zero skew clock-routing trees with minimum wirelength," in *Proc. 5th Annu. Int. ASIC Conf. Exhibit.*, Rochester, NY, USA, 1992, pp. 17–21.

[3] J. L. Tsai, T. H. Chen, and C. C. Chen, "Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 4, pp. 565–572, Apr. 2004.

[4] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[5] R. Deokar and S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *Proc. Int. Symp. Circuits Syst.*, vol. 1. London, U.K., 1994, pp. 407–410.

[6] L. F. Chao and H. M. Sha, "Retiming and clock skew for synchronous systems," in *Proc. Int. Symp. Circuits Syst.*, vol. 1. London, U.K., 1994, pp. 283–286.

[7] I. S. Kourtev and E. G. Friedman, "Clock skew scheduling for improved reliability via quadratic programming," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 1999, pp. 239–243.

[8] X. Liu, M. C. Papaefthymiou, and E. G. Friedman, "Maximizing performance by retiming and clock skew scheduling," in *Proc. 36th Design Autom. Conf.*, New Orleans, LA, USA, 1999, pp. 231–236.

[9] V. Nawale and T. W. Chen, "Optimal useful clock skew scheduling in the presence of variations using robust ILP formulations," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2006, pp. 27–32.

[10] Y. Taur *et al.*, "CMOS scaling in nanometer regime," *Proc. IEEE*, vol. 85, no. 4, pp. 486–504, Apr. 1997.

[11] V. Mehrotra and D. Boning, "Technology scaling impact of variation on clock skew and interconnect delay," in *Proc. Int. Interconnect Tech. Conf.*, Burlingame, CA, USA, 2001, pp. 4–6.

[12] A. Rajaram and D. Z. Pan, "Robust chip-level clock tree synthesis for SoC designs," in *Proc. 45th ACM/IEEE Design Autom. Conf.*, Anaheim, CA, USA, 2008, pp. 720–723.

[13] S. Jilla, "Multi-corner multi-mode signal integrity optimization," *EDA Tech Forum*, 2008. [Online]. Available: http://www.techdesignforums.com/practice/technique/multi-corner-multi-mode-signal-integrity-optimization/

[14] D. Lee and I. L. Markov, "Obstacle-aware clock-tree shaping during placement," in *Proc. Int. Symp. Phys. Design*, Monterey, CA, USA, 2011, pp. 123–130.

[15] Y. Wang, Q. Zhou, X. Hong, and Y. Cai, "Clock-tree aware placement based on dynamic clock-tree building," in *Proc. Int. Symp. Circuits Syst.*, New Orleans, LA, USA, 2007, pp. 2040–2043.

[16] K. Rajagopal *et al.*, "Timing driven force directed placement with physical net constraints," in *Proc. Int. Symp. Phys. Design*, Monterey, CA, USA, 2003, pp. 60–66.

[17] Y. Liu, R. S. Shelar, and J. Hu, "Delay-optimal simultaneous technology mapping and placement with applications to timing optimization," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2008, pp. 101–106.

[18] S. W. Hur, A. Jagannathan, and J. Lillis, "Timing driven maze routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 2, pp. 234–241, Feb. 2000.

[19] K. Sato, H. E. M. Kawarabayashi, and N. Maeda, "Post-layout optimization for deep submicron design," in *Proc. Design Autom. Conf.*, Las Vegas, NV, USA, 1996, pp. 740–745.

[20] Y. P. Chen, J. W. Fang, and Y. W. Chang, "ECO timing optimization using spare cells," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2007, pp. 530–535.

[21] M. Ni and S. O. Memik, "A revisit to the primal-dual based clock skew scheduling algorithm," in *Proc. Int. Symp. Qual. Electron. Design*, San Jose, CA, USA, 2010, pp. 755–764.

[22] S. M. Burns, *Performance Analysis and Optimization of Asynchronous Circuits*, Ph.D. dissertation, Dept. Comput. Sci., California Inst. Technol., Pasadena, CA, USA, 1991.

[23] J. Lu and B. Taskin, "Post-CTS clock skew scheduling with limited delay buffering," in *Proc. Int. Midwest Symp. Circuits Syst.*, Cancun, Mexico, 2009, pp. 224–227.

[24] W. Shen *et al.*, "Useful clock skew optimization under a multi-corner multi-mode design framework," in *Proc. Int. Symp. Qual. Electron. Design*, San Jose, CA, USA, 2010, pp. 62–68.

[25] V. Ramachandran, "Functional skew aware clock tree synthesis," in *Proc. Int. Symp. Phys. Design*, Monterey, CA, USA, 2012.

[26] S. Roy, P. M. Mattheakis, L. Masse-Navette, and D. Z. Pan, "Clock tree resynthesis for multi-corner multi-mode timing closure," in *Proc. Int. Symp. Phys. Design*, Monterey, CA, USA, 2014, pp. 69–76.

[27] J. Bhaskar and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*. New York, NY, USA: Springer, 2009.

[28] V. G. Oklobdzija, V. M. Stojanovic, D. M. Markovic, and N. M. Nedovic, *Digital System Clocking: High-Performance and Low-Power Aspects*. Hoboken, NJ, USA: Wiley, 2003.

Subhendu Roy (S'13) received the B.E. degree in electronics and telecommunication engineering from Jadavpur University, Kolkata, India, and the M.Tech. degree in electronic systems from the Indian Institute of Technology, Bombay, Mumbai, India, in 2006 and 2009, respectively. He is currently pursuing the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA.

He was at Atrenta, Noida, India, where he was involved in developing tools in the architectural power domain and register transfer level (RTL) domain for three years. He underwent summer internships at IBM T. J. Watson Research Center, Yorktown Heights, NY, USA in 2012 and at Mentor Graphics, Fremont, USA in 2013 and 2014. His current research interests include design automation for logic synthesis, physical design, and cross-layer reliability. He has authored papers in major EDA conferences/journals, such as DAC, ISPD, ASPDAC, the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, and holds one U.S. patent.

Mr. Roy was the recipient of the Best Paper Award at ISPD'14.

Pavlos M. Mattheakis received the M.S. and Ph.D. degrees in computer science from the University of Crete, Heraklion, Greece, EU, in 2007 and 2013, respectively.

During his M.S. he was also with the Institute of Computer Science, FORTH, Heraklion, Greece, EU and with ISD S.A. Athens, Greece, EU. From 2007 to 2010 he was with Nanochronous Logic Inc, San Jose, CA, USA. During his Ph.D. he was also with the FORTH and the Technical University of Crete, Chania, Greece, EU. He is currently an R&D Engineer at Mentor Graphics, Grenoble, France, EU. His research interests include the specification and synthesis of asynchronous circuits, the optimization of clock networks in synchronous circuits, the architecture of high performance computers and high level synthesis. Dr. Mattheakis received the best paper award at ISPD14. He holds two patents.

Laurent Masse-Navette received the M.S. degree (Diplôme d'Ingénieur) in computer science from the Institut National Polytechnique de Grenoble, Grenoble, France, and the M.A.S. degree in microelectronics from Université Joseph Fourier, Grenoble.

After studying at INPG-CSI under professor G. Saucier between 1991 and 1994, he held positions as CAD and Research Engineer in the IC design and EDA industries in companies such as ST-Micro, Synopsys Inc. and Pulsic Ltd., where he built up expertise in the physical design and IC layout implementation domains, with a special focus on Clock Tree Synthesis, before joining Mentor Graphics in 2010. He is now leading the Research and Development team on Clock Tree Synthesis based in the Grenoble area as part of the Olympus product R&D team in Mentor Graphics IC Implementation Division.

David Z. Pan (S'97–M'00–SM'06–F'14) received the B.S. degree from Peking University, Beijing, China, and the M.S. and Ph.D. degrees from the University of California, Los Angeles (UCLA), Los Angeles, CA, USA.

From 2000 to 2003, he was a Research Staff Member at IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He is currently the Engineering Foundation Endowed Professor with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA. His current research interests include cross-layer nanometer IC design for manufacturability/reliability, new frontiers of physical design, and CAD for emerging technologies such as 3-D-IC, biochip, and nanophotonics. He has published over 200 papers in refereed journals and conferences and holds eight U.S. patents.

Prof. Pan was the recipient of several awards for the contributions and services, including the SRC'13 Technical Excellence Award, the DAC Top Ten Author in Fifth Decade, the DAC Prolific Author Award, the ASP-DAC Frequently Cited Author Award, 11 Best Paper Awards, including the ISPD'14, ICCAD'13, ASPDAC'12, ISPD11, IBM Research 2010 Pat Goldberg Memorial Best Paper Award, ASPDAC'10, DATE'09, ICICDT'09, the SRC Techcon in 1998, 2007, and 2012, and 11 other Best Paper Award nominations at DAC/ICCAD/ASPDAC/ISPD, the Communications of ACM Research Highlights in 2014, the ACM/SIGDA Outstanding New Faculty Award in 2005, the NSF CAREER Award in 2007, the SRC Inventor Recognition Award thrice, the IBM Faculty Award four times, the UCLA Engineering Distinguished Young Alumnus Award in 2009, the UT Austin RAISE Faculty Excellence Award in 2014, the ISPD Routing Contest Awards in 2007, the eASIC Placement Contest Grand Prize in 2009, and the ICCAD CAD Contest Awards in 2012 and 2013. He served as a Senior Associate Editor of the *ACM Transactions on Design Automation of Electronic Systems*, an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART I, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART II, *Science China Information Sciences*, the IEEE CAS Society Newsletter, and the Executive/Program Committees of several major conferences, such as DAC, ICCAD, ASPDAC, and ISPD.