SynECO: Incremental Technology Mapping with Constrained Placement and Fast Detail Routing for Predictable Timing Improvement

Anuj Kumar, Tai-Hsuan Wu, Azadeh Davoodi
Department of Electrical and Computer Engineering
University of Wisconsin, Madison WI 53706
{*akumar2, twu3, adavoodi*}*@wisc.edu*

***Abstract*— We present SynECO, a framework to achieve predictable timing improvement via incremental resynthesis and replacement. We target timing-critical paths postplacement and resynthesize and replace promising gates. We show since the wire delays are the non-negligible contributors to a critical-path delay, it is crucial to accurately estimate them to make a predictable synthesis modification. For this purpose, we incorporate an accurate timing analysis tool which uses fast detail routing for wire delay estimation. This allows generating timing estimates that correlate much better with post-routing values compared to Steiner-tree-based estimate of wiring tree and using D2M delay model. Detail routing information allows incorporation of factors such as crosstalk, metal layer assignment and via delays which are crucial for accurate analysis. For fast synthesis, we constrain our logical modifications to be from the physical neighborhood of target gates on the critical paths. Our synthesis framework is completely integrated with the Cadence Encounter tools for physical design.**

I. Introduction And Previous Work

The performance of today's VLSI systems is highly affected by the wires. Accurate estimation of the wire delay and loading is now crucial for achieving timing closure. This estimation is particularly crucial at higher stages of the design such as logic synthesis to reduce the number of iterations in the design cycle and achieve faster closure. It reduces the overhead of post-routing optimization schemes such as buffer insertion [1]. Accurate wire estimation is particularly important when doing timing optimization on the timing-critical paths of the circuit as they typically span over the chip and tend to have long wire segments.

Many of the existing approaches focus on predicting the wire effects such as timing or congestion at the synthesis stage prior to doing placement. For example [2], [3], [4] introduce metrics such as adhesion, distance or probabilistic maps to model routability and congestion during timing-driven synthesis. Although these metrics are intuitive, they are not exact and their degree of effectiveness is unclear as many are not validated post-routing. Also, the lack of placement information results in error in wire-length estimation.

Other approaches were proposed to directly incorporate placement information for more accurate estimation of wire-length and congestion [5], [6], [7]. In [5] logic synthesis and placement were simultaneously solved in a simplified "linear" framework. [6] was based on one-time placement of the design in which the final placement significantly deviated from the one that was originally used for making synthesis decisions. In [7] incremental placement was used to minimize the disturbance to a cost function during synthesis. However, the estimates made for technology independent synthesis with placement information were still inaccurate in [7]. Other incremental approaches similarly suffered because of significant difference in consecutive rounds of placement.

Recently, works such as [1], [8], [9] can more effectively avoid these downsides. Further, [10] includes "don't cares" for more effective logical modification. However most of the recent approaches offer a more accurate physical synthesis by only utilizing the placement information. We argue that even though at the post-placement stage, wirelength can be predicted with high-level of accuracy, accurate timing estimation which accounts for factors such as crosstalk, delay of different metal layers and vias can not be estimated with sufficient accuracy. These factors heavily affect delay in the nanometer regime and can not be estimated solely using post-placement information. What is needed is incorporation of routing in addition to placement for more accurate prediction.

The main contributions of our work are as follows:

- We propose SynECO for physically-aware technology mapping at post-placement stage. To evaluate synthesis moves, we use incremental placement and an accurate timing engine which incorporates a fast detail router.
- Since each of our potential synthesis moves are accurately evaluated down to the routing-level, to handle the runtime overhead, we propose a constrained synthesis which considers those logical neighbors of target gates which are within their close physical proximity. Working with these physical and logical neighbor gates ensures wire delay on the critical paths will not increase.
- We further show that wire delay estimates using the global-routing information (based on Steiner-tree estimation) and D2M delay model underestimates the critical path delay. Even though we observe that total wire-length can fairly accurately be predicted using Steiner-tree estimation, this yet does not provide sufficient delay accuracy. What is needed is detailed routing information which allows timing analysis that can incorporate via and wire delay and topology at different routing layers.
- We show that simple logical modifications using SynECO can replace costly post-routing optimizations and reduce the number of violating tracks post-routing. For one benchmark, we were able to save 59 buffers post-routing by one logic modification after placement. Furthermore, we demonstrate that SynECO does not perturb the layout and does not change the total wire-length significantly.

978-1-4244-2658-4/08/$25.00 ©2008 IEEE

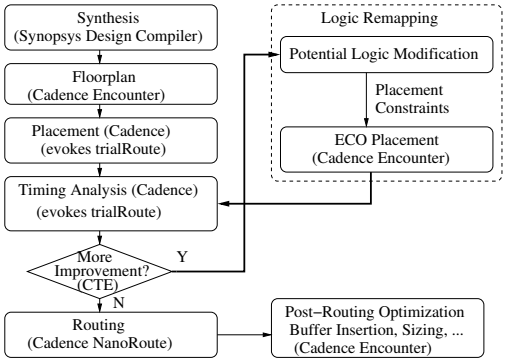

Fig. 1. Design flow: traditional (left) and modified (right)

While SynECO is completely based on the Cadence Encounter tools for placement, routing and routing-aware timing analysis, it can also be integrated with any other tool that supports incremental and constrained optimization. We explain the details of SynECO in the next Section II. We then present our simulation results and conclusion Sections.

II. Physical Synthesis for Timing Closure

A. Design Flow and Motivation

Fig. (1) shows a traditional (left) and our modified (right) design flows. As shown, the design is initially mapped onto a gate level netlist. This is done for area minimization for meeting a timing constraint for a given technology library. The generated gate level netlist is then taken through the physical design flow. The floorplan determines the aspect ratio and the row orientation for placing the mapped gates. Timing aware placement engine is then used to place the design. Placement is conducted in the timing-driven mode for meeting a timing constraint and area minimization. This is followed by timing analysis on the placed design. We use the Common Timing Engine (CTE) tool of Cadence for routing-aware timing analysis. The CTE utilizes a fast router (trialRoute) for accurate timing estimation accounting for the wire effects (e.g. metal layers, loading, crosstalk, topology, congestion, etc.). If the timing is acceptable, we move to the routing stage after which we can apply post-routing optimization techniques such as sizing and buffer insertion to further improve timing. However, if the post-placement timing is not satisfactory, we move to the modified flow given in the right hand side of Fig. (1) and apply SynECO. Here we first do technology remapping with ECO (incremental) placement, followed by routing-aware timing analysis using CTE. To do these steps quickly, we generate placement constraints to enforce minimal placement moves, thereby minimal perturbation to the original placement in case it was generated using objectives other than timing. In this paper, we use Synopsys Design Compiler with a 90nm TSMC library for the initial synthesis and Cadence Encounter for physical design. Other variations of the conventional flow can similarly be extended to be integrated with SynECO.

Our motivation is given by the following example. Consider Fig. (2) which shows the placement of the ISCAS S38584 benchmark. We observed timing violation after placement and highlight the most-critical path.

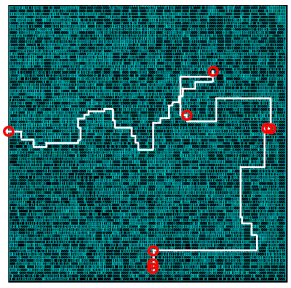

Fig. 2. Critical path in circuit S38584: physical view

The logic cells on the path are marked in red. Note the figure is generated after timing analysis which applies fast detail routing so we can view the routes. Our goal is to make timing improvement by technology remapping on such critical paths. To achieve this goal, it is crucial that we accurately account for wire effects. As can be seen from the figure, in the highlighted critical path, some of the routes connecting the gates are very complicated and they span all over the chip. For this path, the total wire delay contributes to 13% of the overall path delay. Considering the fact that this is a *critical* path and its delay is highly influenced by the routes that span through the chip, it is crucial that the effect of these routes be accurately incorporated when making any logical change for timing improvement.

Moreover, if placement information is used to provide the net terminal locations, popular estimates such as minimum Steiner tree are far from accurate for estimating the topology of the wire segments. For the highlighted path in the Figure, we verified that the Steiner-tree estimate of the length of the longest wire on the path (obtained from Flute [11]) was within 0.2% accuracy of the one obtained using trialRoute of Cadence. However, the topology of this wire was significantly different than the Steiner tree. Furthermore, information about layer assignment (and metal layer delay) and vias can not be accurately provided by Steiner-tree estimation but are very important in delay estimation. Specifically, since we are dealing with a timing-critical path, it becomes crucial to accurately estimate wire delays and the impact on placement to obtain delay improvement.

We therefore propose SynECO which resynthesizes (and replaces) the design to make timing improvement. During the replacement, it makes accurate timing estimates using CTE of Cadence which is integrated with trialRoute for routing-aware timing estimation. To manage the run-time overhead of placement and routing, SynECO utilizes the placement tool in a constrained incremental mode. This also ensures the wire segments on the critical path do not degrade as a result of synthesis modification.

Our objective of logic-level timing improvement can serve two purposes. First, when violating a timing constraint, we show that by making minimal yet accurate logic modifications, we can reduce the overhead of post-routing timing optimization techniques such as buffer insertion, thereby improving other metrics such as routing congestion. Second, by accurate logic modifications we can more aggressively improve the performance, as shown in our simulation results. Next we explain the details of our SynECO framework.

B. ECO-based Technology Remapping: Algorithm

We iteratively make logical modifications. At each iteration one gate (or a set of consecutive gates) on the critical path is remapped using its neighboring non-critical gates. In the next iteration, if timing improvement is still required, we repeat the same procedure to select and remap another critical gate. The algorithm below describes selection of gates from the critical paths for remapping in one iteration:

Algorithm 1: Gate selection from the critical paths for remapping

1. Identify set G_c of gates that are on the critical paths P_c
2. Find subset $G_{mc} \in G_c$ with maximum connections to critical sinks
3. For each $g \in G_{mc}$, select a segment of its critical path that includes its consecutive fanins $g_f \in P_c$ that are in physical proximity of g which have the largest delay among other critical path segments
4. Build macro-cell around g and selected g_fs using non-critical gates and apply remapping

After post-placement timing analysis, we first identify the set of gates G_c that belong to the critical paths. Among these gates, we then identify the subset of gates G_{mc} that connect to maximum number of sinks (output registers) that are timing critical. The subset G_{mc} is of more interest to us because improving the timing of such gates can simultaneously improve the timing of multiple critical outputs. For each gate $g \in G_{mc}$ we then identify a segment on the critical path that includes g and that has maximum delay. Specifically, the critical path segment is selected to include consecutive fanins of g (indicated by g_f) which ends at g. Furthermore, we constrain all the selected g f to be within the physical proximity of g. We empirically set this physical proximity such that the Manhattan distance of g and g f is within 2% of the chip dimension. It is important that we select consecutive gates on the critical path. Otherwise, we are not able to calculate the arrival times for non-consecutive gates on the same path when making simultaneous synthesis decisions.

To choose the path-segment ending at g with maximum delay, we use the CTE routing-aware timing analysis. Therefore among all the critical path segments, we select the one with maximum delay which also connects to many number of critical sinks. Remapping of such path segment for timing improvement is a promising strategy. If more timing improvement is needed, we repeat the procedure to identify another path segment in the next iteration.

Figure (3) elaborates the above. The bold path is the logic view of the critical path of S38584 in Fig. (2). For this design, we had 17 critical paths post-placement. Based on the above criteria we found that the gate NORD0 connected to maximum number of critical outputs while it had the maximum delay (it had 22 fanouts which are not plotted in the figure). We therefore selected this gate to be remapped. In this case, the fanin INV was not in the physical proximity of the gate so it was not included for remapping. However, from the Fig. (2), the gates in the end of the path located at the bottom of the chip are physically close to each other, yet their cumulative delay was not more than of NORD0.

Next, we explain the details of the remapping process. First we discuss selecting the neighboring non-critical gates and then explain a physically-aware remapping algorithm.

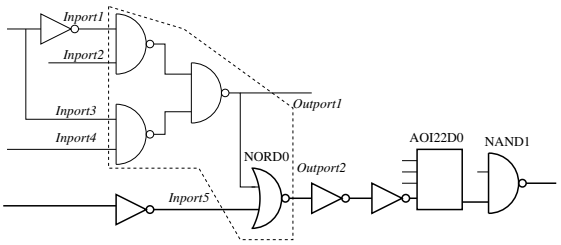

Fig. 3. Critical path (bold) in circuit S38584: logic view; building macro-cell of the logical neighboring non-critical gates

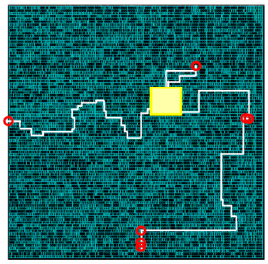

Fig. 4. The selected gates in a macro-cell should also be in the same physical rectangular neighborhood

1) Building Macro-Cells of Neighboring Gates: Once the critical path segment is selected, we select some neighboring gates of this segment that does not fall on the critical path such that they fall into a macro-cell. We then remap all these gates in order to improve the arrival time at the output of g. To identify the non-critical gates, we recursively traverse the fanins of g and g f s (i.e., non-critical fanins of the selected critical path segment) and assign them to the same macrocell, as illustrated in Fig. (3).

Our idea of building macro-cells is inspired by [5] where it was originally proposed. However, here we impose additional constraints during gate selection to ensure that they are in the same physical neighborhood (which can be done given that we are at the post-placement stage). First, we empirically assume a rectangular target macro-cell area of 2chip area. We require all the selected gates in the macro-cell to fall within this rectangular area, as shown in Fig. (4) for NORD0. We only allow logic modifications of the logicallyneighbor gates in a macro-cell. Later on during incremental placement, we replace all the cells in the macro-cell which can include gates from other paths as well. More specifically, our macro-cell selection is based on the algorithm below.

Algorithm 2: Gate selection for a target macro-cell

1. Initialize subcircuit SC=$g \bigcup g_f$
2. Set bounding box: BB(SC) = 0
3. Initialize queue and add SC to it: $SC \rightarrow Q$
4. While Q not empty
   - Extract current gate cg from top of queue
   - If BB($SC \bigcup cg$)$\leq$ target macro-cell area
     - Add current gate to subcircuit SC=$SC \bigcup cg$
     - Update bounding box BB(SC)
     - Add all fanins of cg to end of Q

Here we recursively choose the gates in the fanin cone of the critical gate until their encapsulating bounding box becomes larger than the target macro-cell area.

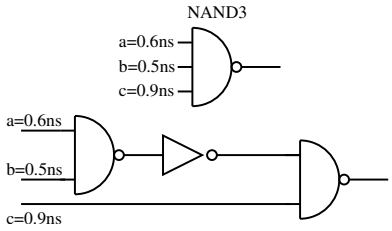

Fig. 5. Decomposing each gate to decrease the delay of critical fanin to output: example decomposition of NAND3

2) Technology Remapping Algorithm: For the selected macro-cell we highlight the following observations from [5] which should be considered during timing-driven remapping:

- The macro-cell has some outputs and some input ports as shown in the example of Fig. (3). The output of critical gate g is also an output port of the macro-cell.
- The input ports have an arrival time given from the previous stages of the circuit.
- The arrival time of the critical output port (output of g) needs to be improved.
- The "non-critical" output ports (other than g) have a required arrival time that need to be satisfied when the macro-cell is remapped. This required arrival time is in fact the arrival time of these output gates before remapping. This ensures that the remapping process will not affect the timing of the other paths in the circuit that include the non-critical gates in the macro-cell.

We next apply a standard timing-driven technology mapping algorithm to remap the gates inside the selected macro-cell. However, we evaluate each potential remapping move using incremental placement and routing-aware timing analysis. The remapping problem is defined as:

Given a gate-level macro-cell, arrival time at the input ports, required arrival times at the non-critical output ports, remap the gates such that the arrival time at the critical output port is minimized. This should be done such that the required arrival time at the non-critical output ports are also honored. Our remapping procedure is similar to the classic dynamic programming algorithm of [12]. Note that [12] is for area improvement while ours is for timing improvement.

Algorithm 3: Technology mapping with accurate timing evaluation

1. Compute routing-aware arrival times t_{in} for input ports of macro-cell
2. Compute routing-aware arrival times t_{out} for non-critical output ports and set t_{out} to be the required arrival time constraints in those output ports
4. Decompose the gates in terms of INV and NAND2 gates (honor the implementation that minimizes the delay of the critical fanin path for each gate as shown in Fig. (5))
5. Apply dynamic programming technology mapping algorithm for timing improvement
   - For each potential match at each internal node in the macro-cell determine the arrival time using incremental placement and the routing-aware timing analysis CTE
   - Find the implementation that satisfies the output constraints t_{out} that gives the most timing improvement at the critical output gate

We decompose the macro-cell into a few number of basic logic gates (INV and NAND2 gates (step 4 of the algorithm)). Among different ways to decompose each gate in terms of these basic gates, our priority is to decrease the delay of the most critical fanin of a gate similar to [7].

Among different fanins of a gate, the one that has the largest arrival time is the most critical one. During the decomposition of a gate, we choose the implementation that decreases the delay of the path that connects this critical fanin to the output of the gate. As an example, in Fig. (5), the NAND3 gate has one critical fanin with arrival time of 0.9nsec. We show the decomposition which allows the minimum delay path of this fanin to the output by going through only one NAND2 gate.

After decomposition, we topologically traverse the graph from the inputs towards the outputs. At each node, we determine the best match corresponding to mapping the fanin cone of the node. The best match is the one that gives the minimum arrival time at that node. This arrival time is computed using incremental placement and routing-aware timing analysis CTE. Once all the nodes are topologically processed, we obtain the minimum achievable arrival time at the critical output node. To determine its corresponding match, we traverse the circuit in reverse topological order which allows identifying and finalizing the match at each node. During the matching process, we disregard any implementation that violates the required arrival time constraints at the non-critical output ports. This is important to ensure that the mapping at the macro-cell will not increase the delays of the other paths in the circuit.

C. Efficient Physically-aware Timing Evaluation

During the matching phase, *to evaluate each potential match* in the fanin cone of a gate, we employ incremental placement followed by routing-aware timing analysis. Here we elaborate the general procedure and describe how we perform placement when some gates are logically decomposed.

First consider the **simple case** in which we want to evaluate a potential remapping of the entire macro-cell. In this case, we evoke incremental placement. We constrain all the gates outside the macro-cell to their previous positions but we allow all the gates inside the macro-cell to be repositioned. This also includes other gates inside the physical bounding box defined by the macro-cell that are not considered for logical modification. We constrain this repositioning to be within the macro-cell area. This ensures the wire segments on the critical path will not be degraded after logic modification, in addition to ensuring minimal perturbation to the layout. The incremental placement is done very fast because placement is only done on the small macro-cell area. Next, we use the routing-aware timing engine. These timing estimates are far better than other estimates in the logic level. Furthermore, since trialRoute is evoked for fast detail routing, the timing estimates include crosstalk, delay on different metal layers and via delays as noted in the CTE manual for timing analysis. The runtime of timing analysis is faster than constrained placement. Overall, both steps (constrained placement and timing analysis) are done fast which make their integration during synthesis practical.

Finally, consider the more **complex case**, when we want to evaluate timing during the matching process at an internal node. In this case, we face a problem during placement because the gates in the macro-cell are decomposed.

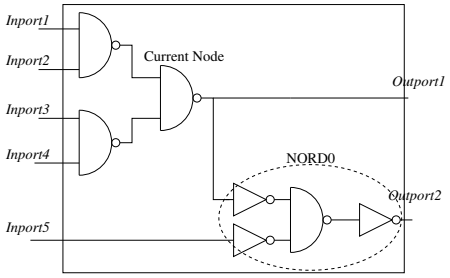

Fig. 6. Remembering the gate assignment before decomposition for accurate incremental placement in a macro-cell

For example consider the gate marked as current node in Fig. (6). When calling incremental placement for a match at the fanin cone of this node, we do not want to use the decomposed structure of its fanout gate (NORD0 in the figure). We need to identify their original gate types and their corresponding positions for incremental placement during mapping of an internal node. To work around this problem, we remember the original gate assignment of the decomposed gates that are not in the fanin of the current node. For example in the figure, we tag those INV and NAND2 gates that compose the original NORD0. Now, to do incremental placement, we re-identify the original gates that are not in the fanin cone of the current node. These gates together with the newly mapped gates all go through incremental placement in the macro-cell. In case some decomposed nodes belong to the same gate as the current-node, we also tag these decomposed nodes but leave the current node as decomposed during placement.

D. Discussion on Complexity and Convergence

Because of the small area of the macro-cell, we capture a small, yet relevant number of neighboring logic gates to improve the arrival time of the target critical gate(s). Each round of incremental placement is done fast (about fraction of a second for our largest test designs) because placement is done only on the cells in the macro-cell. Furthermore, timing analysis is done also fast (faster than constrained placement).

At each iteration of our algorithm, we guarantee that we will consistently improve the timing of each macro-cell. This is due to accurately evaluating all the potential mappings; *when we predict an improvement, we will indeed obtain improvement*. While our algorithm is limited to the number of cells that fall in a macro-cell which can be used for resynthesis, we still can perform effective optimization, because we focus our attention to fix the timing of only the timing-critical paths and not the entire design. Moreover, even though the macro-cell area is small but it still includes many gates (some of which are not the logical neighbors of the critical gate). All of these gates are subject to replacement. As shown in our simulation results we require few number of iterations with large-sized benchmarks to fix the timing violation.

III. Simulation Results

We used Synopsys Design Compiler with 90nm TSMC technology library for initial synthesis for area minimization under a timing constraint. We then placed the design using Cadence Encounter. Placement was done for minimum area under a timing constraint.

TABLE II

Benchmark and information prior to optimization

	Benchmark Info		Post-Placement Info		
Bench	#gates	#nets	Areaμ^2	#critical paths	%critical gates
s820	285	255	690.3	3	15.4
s9234	1711	933	4140.1	6	5.70
s13207	3460	1798	8372	42	3.22
s38584	14726	8041	35625.9	17	0.61
s38417	17194	8187	41595.7	16	1.03
s35932	18271	9241	44201.2	17	0.57
b17	34715	25577	83984.1	12	0.08

Consequently, note that we have very little white space for each benchmark because of area minimization under a timing constraint. This made working with benchmarks of various sizes possible and the lack of white space made the incremental placement more challenging and more realistic.

We implemented the **Regular** flow in which we only applied the above-mentioned synthesis, placement and routing once. In case of timing-violation post-routing, we applied post-routing optimization with the target timing constraint which according to Cadence manual used a combination of gate sizing and buffer insertion.

We also implemented **SynECO** as our resynthesis framework. SynECO was implemented in C and interacted with the physical design tools by generating placement constraints and receiving the timing information. We used the Cadence Encounter tools for physical design (placement, routing, routing-aware timing analysis and post-routing optimization). Incremental placement also ran in the timing-driven mode with an area constraint obtained after the original placement.

To highlight the importance of accurate routing-aware timing analysis, we implemented the following alternative approach: We used the same mapping algorithm and Cadence for incremental placement. However, for routing-aware timing analysis, we used Flute [11] for minimum Steiner-tree approximation of the wires. Flute generates the interconnect tree which is described as horizontal and vertical branches which we assumed to belong to two different metal layers. For each metal layer, we obtained the wire width, edge capacitance, capacitance and resistance per square distance from the TSMC 90nm library. Similarly, the gate delay and capacitance information were extracted from the library. With the approximated interconnect tree and the technology parameters, we used the second-order D2M delay model [13] to estimate the delay of wire segments on the critical paths. We denote this approach as **Steiner-D2M**. Our benchmarks were selected from the ISCAS89 and ISCAS99 suite. Their gate count, net count are listed in Table II along with the area, number of critical paths and gates after initial placement.

We considered the following two cases in our experiments. In the first case, our goal was to fix the timing violation post-placement to meet a target timing constraint. First, Table I lists the clock delay constraint in column 2 (note information about number of critical paths and gates are in Table II). We applied SynECO and stopped as soon as timing was met. We then applied routing to verify the results of SynECO and the Regular approach. In Table I, we report the degree of timing violation at different stages for these two approaches. We make the following observations:

TABLE I

COMPARISON OF TIMING VIOLATION AT VARIOUS STAGES OF THE PHYSICAL FLOW

			Post-Placement Timing (ns)		Post-Routing Timing (ns)		Post-Routing Optimized (ns)	
Bench	Clock(ns)	Steiner-D2M(ns)	Regular(%viol)	SynECO(%viol)	Regular (%viol)	SynECO(%viol)	Regular-Opt(%viol)	SynECO-Opt(%viol)
s820	0.62	0.38	0.669 (7.90%)	0.608(-1.94 %)	0.688 (10.97%)	0.627 (1.13 %)	0.61 (-1.61 %)	0.603 (-2.74 %)
s9234	0.81	0.57	0.842 (3.95%)	0.809(-0.12 %)	0.872 (7.65 %)	0.832 (2.72 %)	0.806 (-0.49 %)	0.785 (-3.09 %)
s13207	0.4	0.37	0.485 (21.25%)	0.415(3.75 %)	0.502 (25.50%)	0.434 (8.50 %)	0.418 (4.50 %)	0.424 (6.00 %)
s38584	1.91	1.29	1.913 (0.16%)	1.32 (-30.89%)	2.308 (20.84%)	1.434 (-24.92%)	1.516 (-20.63%)	1.424 (-25.45%)
s38417	1.25	1.03	1.253 (0.24%)	1.235(-1.20 %)	1.316 (5.28 %)	1.314 (5.12 %)	1.226 (-1.92 %)	1.224 (-2.08 %)
s35932	1.1	0.83	1.113 (1.18%)	0.975(-11.36%)	1.345 (22.27%)	1.094 (-0.55 %)	1.094 (-0.55 %)	1.098 (-0.18 %)
b17	1.7	1.33	1.717 (1.00%)	1.637(-3.71 %)	1.862 (9.53 %)	1.795 (5.59 %)	1.699 (-0.06 %)	1.645 (-3.24 %)
Average			(5.10%)	(-6.50 %)	(14.58%)	(-0.34 %)	(-2.97 %)	(-4.40 %)

TABLE III

MINIMAL PERTURBATION IN CONSTRAINED PLACEMENT

Bench	#gates moved (%gates)	Areaμ2	Mean(x+y) μm	Max(x+y) μm
s9234	22(2.40%)	690.3	1.60	3.80
s13207	87(5.09%)	4140.1	1.91	1.91
s38584	28(0.35%)	8372	1.57	1.44
s38417	45(0.55%)	35625.9	1.14	5.04
s35932	29(0.32%)	41595.7	1.76	5.40
b17	53(0.20%)	44201.2	1.75	5.76

1) As shown in column 3 of Table I, Steiner-D2M always underestimated the critical path delay and satisfied the clock delay constraint. However, the actual timing estimates obtained with CTE timing engine on the same placed design clearly indicated the timing violations on the critical paths which were masked using Flute based timing analysis.

2) The degree of timing violation in the regular flow increased from 5.1% post-placement to 14.58% post-routing on average, while in our case, we always satisfied the timing constraint post-placement and our timing improvement dropped from 6.5% post-placement to 0.34% post-routing. This drop is with a smaller rate in our case than the regular flow because we better accounted the impacts of the synthesis moves on placement and routing.

3) On an average we always satisfied the timing constraint post-routing (-0.34% timing violation) while the regular approach had 14.58% timing violation and required post-routing optimization. So SynECO could eliminate the need for post-routing optimization in the majority of the cases.

Table IV elaborates the overhead of post-routing optimization applied to the regular approach. Column 4 shows the number of upsized gates and inserted buffers to the solution of the regular approach. Note for the largest circuit b17, 59 extra buffers were inserted in the regular case while this was completely avoided in our case. Moreover, the regular approach had more difficulty during routing. Column 3 shows the extra number of violating routing tracks of the regular approach compared to SynECO. Overall, SynECO resulted in a slight increase in total wirelength as shown in column 2 which on average was less than 1%.

SynECO also caused minimal perturbation to the layout as shown in Table III which reports the number of gates moved, the area, and the average and maximum perturbation in x and y directions for each benchmark.

Finally, for the top 3 largest benchmarks we did aggressive timing improvement after the timing constraint was originally satisfied. Table V shows the post-placement timing improvement of SynECO and number of iterations applied compared to the regular case. As can be seen we obtained large improvements by applying several iterations of SynECO.

TABLE IV

COMPARISON OF WIRELENGTH AND NUMBER OF VIOLATING ROUTING TRACKS POST-ROUTING (PR) AND POST-ROUTING OPTIMIZED (PRO)

	SynECO-PR Wirelength	Reg.-PRO Additional Overhead	
Bench	%extra	#Routing Viol.	Extra Upsized/Buffers
s820	7.79	11	16Upsized
s9234	0.17	10	28Upsized
s13207	0.28	4	2Upsized
s38584	0.17	-	30Buffers
s38417	0.01	5	14Buffers
s35932	0.11	1	1Buffers,75Upsized
b17	0.05	8	59Buffers

TABLE V

AGGRESSIVE TIMING IMPROVEMENT POST-PLACEMENT WITH SYNECO

		Post-Placement Timing (ns)			
Bench	Clock(ns)	Reg.	SynECO	%improvement	#iter
s38584	1.91	1.913	1.081	43.49%	9
s35932	1.1	1.113	.732	34.23%	8
b17	1.7	1.717	1.518	11.58%	7

IV. CONCLUSIONS

We presented SynECO, a post-placement technology mapping engine to iteratively improve the timing of critical paths by using incremental placement and fast detail routing for predictable timing evaluation. We show SynECO can eliminate the need of post-routing optimization and facilitate routing by reducing the number of routing violations.

REFERENCES

[1] C. Alpert, S. Karandikar, Z. Li, G.-J. Nam, S. Quay, H. Ren, C. Sze, P. Villarrubia, and M. Yildiz, "Techniques for fast physical synthesis," vol. 95, no. 3, pp. 573–599, May 2007.

[2] P. Kudva, A. Sullivan, and W. Dougherty, "Metrics for structural logic synthesis," in *Proc. ICCAD*, 2002, pp. 551–556.

[3] Q. Liu and M. Sadowska, "Wire length prediction-based technology mapping and fanout optimization," in *Proc. ISPD*, 2005, pp. 145–151.

[4] R. S. Shelar, P. Saxena, X. Wang, and S. S. Sapatnekar, "An efficient technology mapping algorithm targetting routing congestion under delay constraints," in *Proc. ISPD*, 2005, pp. 137–144.

[5] J. Lou, W. Chen, and M. Pedram, "Concurrent logic restructuring and placement for timing closure," in *Proc. ICCAD*, 1999, pp. 31–36.

[6] D. Pandini, L. T. Pileggi, and A. J. Strojwas, "Congestion aware logic synthesis," in *Proc. DATE*, 2002, pp. 664–671.

[7] A. L. S.-V. W. Gosti, S. P. Khatri, "Addressing the timing closure problem by integrating logic optimization and placement," in *Proc. ICCAD*, 2001, pp. 224–231.

[8] J. Roy and I. Markov, "Eco-system: Embracing the change in placement," in *IEEE Trans. on CAD*, vol. 26, no. 12, 2007, pp. 2173–2185.

[9] K.-H. Chang, I. L. Markov, and V. Bertacco, "Saferesynth: A new technique for physical synthesis," *Proc. Integration: the VLSI Journal*, vol. 41, pp. 554–556, 2008.

[10] S. Plaza, I. Markov, and V. Bertacco, "Optimizing non-monotonic interconnect using functional simulation and logic restructuring," in *Proc. ISPD*, 2008, pp. 95–102.

[11] C. N. Chu, "Fast lookup table based wirelength estimation technique," in *Proc. ICCAD*, 2004, pp. 696–701.

[12] K. Keutzer, "Dagon: Technology binding and local optimization by dag matching," in *Proc. DAC*, 1987, pp. 341–347.

[13] C. Alpert, A. Devgan, C. Kashyap, "Rc delay metrics for performance optimization," *IEEE Trans. CAD*, vol. 20, no. 5, pp. 571–582, 2001.