Hindawi Publishing Corporation
VLSI Design
Volume 2011, Article ID 503025, 10 pages
doi:10.1155/2011/503025

Review Article

Shedding Physical Synthesis Area Bloat

Ying Zhou,[1] Charles J. Alpert,[1] Zhuo Li,[1] Cliff Sze,[1] and Louise H. Trevillyan[2]

[1] *IBM Austin Research Laboratory, Austin, TX 78758, USA*
[2] *IBM Waston Research Laboratory, Yorktown Heights, NY 10598, USA*

Correspondence should be addressed to Ying Zhou, nancyzhou8@gmail.com

Received 1 October 2010; Accepted 11 December 2010

Academic Editor: Shiyan Hu

Area bloat in physical synthesis not only increases power dissipation, but also creates congestion problems, forces designers to enlarge the die area, rerun the whole design flow, and postpone the design deadline. As a result, it is vital for physical synthesis tools to achieve timing closure and low power consumption with intelligent area control. The major sources of area increase in a typical physical synthesis flow are from buffer insertion and gate sizing, both of which have been discussed extensively in the last two decades, where the main focus is individual optimized algorithm. However, building a practical physical synthesis flow with buffering and gate sizing to achieve the best timing/area/runtime is rarely discussed in any previous literatures. In this paper, we present two simple yet efficient buffering and gate sizing techniques and achieve a physical synthesis flow with much smaller area bloat. Compared to a traditional timing-driven flow, our work achieves 12% logic area growth reduction, 5.8% total area reduction, 10.1% wirelength reduction, and 770 ps worst slack improvement on average on 20 industrial designs in 65 nm and 45 nm.

1. Introduction

Physical synthesis is the critical part of modern VLSI design methodologies. It refers to the process of placing the logic netlist of the design as well as sizing/adding/removing/logic changing cells, concurrently optimizing multiple objectives under given constraints, where objectives and constraints are choices among area, power, timing, and routability depending on design characteristics. In the last decade, timing closure has been the main focus of physical synthesis flow [1], partially due to interconnect delay dominance over gate delay from technology mitigation.

So why do we suddenly care about area bloat? In 65 and 45 nm technologies, design companies tend to pack more logic functionalities into a small-sized die to balance the expensive fabrication cost, while they also want to keep low power budget to maintain their competitive margin. Such requirement could break the traditional timing-driven flow which tends to consider area as merely a constraint and overdesigns the chip.

Area bloat could cause several problems.

(1) More power consumption: area is the first-order estimation of the power, especially for dynamic power. For leakage power, smaller area device tends to have less leakage even in the gate library family with same threshold voltage (Vt).

(2) Congestion problems: there are many causes for congestion problems, such as inferior floorplan and bad placement. Area bloat is one significant cause, which creates high logic gate density in certain regions, and there are no enough tracks to route all nets.

(3) Timing problems: when the chip area has been fully occupied, there is no extra space for optimizations to further improve timing, or new buffers and resized gates are moved a long distance during legalization and big timing degradation happens.

Each problem described above or their combination could cause designers to increase die size, restart floorplanning and complete physical synthesis flow, which in turn lengthen the total time-to-market.

One example of area bloat causing congestion problems is shown below. For an industrial 45 nm design with 102 K input gates, we first run a traditional timing-driven flow and a global router with 5% detour length control to

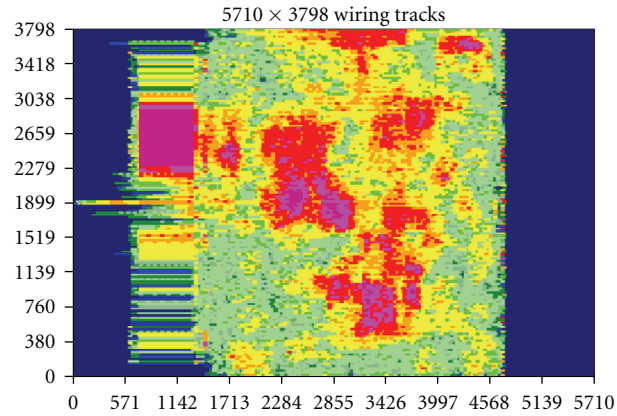


Figure 1: Horizontal congestion from timing driven physical synthesis flow.

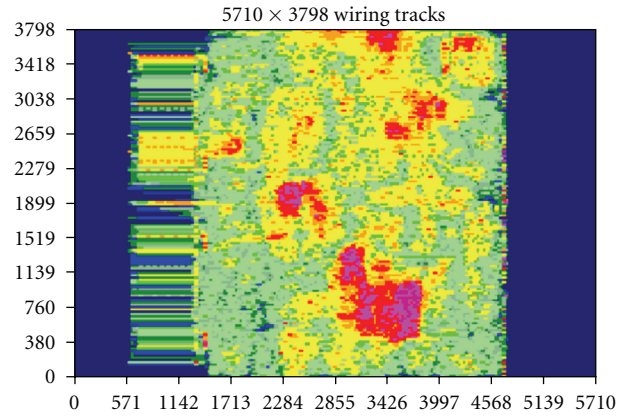


Figure 2: Horizontal congestion from area efficient physical synthesis flow.

measure the congestion. The congestion picture is shown in Figure 1 and the pink area is most congested one. The average congestion metric (measured by taking the worse 20% congested nets and averaging the congestion number of all routing tiles these nets pass through) is 94%, with 5535 nets pass through 100% routing congested tiles. Then with the techniques later discussed in this paper, the area is reduced by 8%, and the congestion picture is shown in Figure 2. The average congestion metric decreases to 89% with only 2856 nets passing through 100% routing congested tiles. The output netlist with the new technique can actually be routed with some further cell spreading techniques, where the original design has even no free space to be spread. Therefore, it is important to achieve a minarea physical synthesis flow.

The major sources of the area bloat from the physical synthesis are buffering and gate sizing. Even though there are lots of existing literatures on these problems, there are still practical constraints that existing approaches do not model correctly.

1.1. Buffer Insertion. Buffer (repeater) insertion, is a popular technique to reduce interconnect delay and fix electrical violations. Perhaps the most well-known buffer insertion algorithm is Van Ginneken's classic dynamic programming algorithm [2], which has led to several improvements for the kernel algorithm, such as speedup [3], resource control [4, 5], and slew constrained buffering [6]. Other extensions or related work include buffer tree construction [7, 8], buffering with more accurate delay models [9], buffering for noise reduction [10], and simultaneous wire sizing and layer assignment [11].

Most of these literatures focus on single algorithm for a particular problem. However, creating an area efficient flow based upon these existing techniques and the way to handle all practical constraints are rarely discussed, which have big impact to the design area at the end of the flow. To list a few, we mention the following.

(i) Should one use slew constrained buffering or timing driven buffering (they have different area and timing results)?

(ii) How to set the input slew and slew constraint for buffering algorithms, which has the big impact on the area?

(iii) Can one still use Elmore delay and linear gate delay model in buffer insertion for modern designs and any area impact?

(iv) How to handle rising and falling transitions inside the algorithm?

The impact of slew constraint on buffer area is shown in Figure 3. The experiment is done at a 5 mm long line on a 2X wide/thick layers in 45 nm technology where buffers are placed at the max distance to meet the specified slew constraint in a repeated pattern. As slew constraint becomes smaller, the distance between buffers is smaller, which results in more buffers and bigger buffer area. On another hand, the signal delay per mm, the sum of buffer delay and wire delay divided by the slew reach length, is measured and shown in the same figure. One can also see that by adjusting the slew goal, one can achieve the optimal delay for a buffered wire without performing timing-driven buffering.

The impact of input slew and rising/falling inputs is illustrated in Figure 4, where we choose a buffer from a 45 nm node buffer and show the relationship between the delay and capacitance under different input slew values and rising/falling input transitions. It is clear that the delay is also quite sensitive to the input slew as well as the rising and falling input directions, and the difference could be 10% to 15%.

1.2. Gate Sizing. Gate sizing has also been extensively studied in the literature. Most early work assumes the library is continuous, models the sizing problem as a convex programming with closed form solution [12] or Lagrangian relaxation [13]. These works ignore the fact that most cell library-based designs have discretized gate sizes. Later, some rounding techniques have been proposed to map the continuous solutions to the discrete grid [14]. More recently,

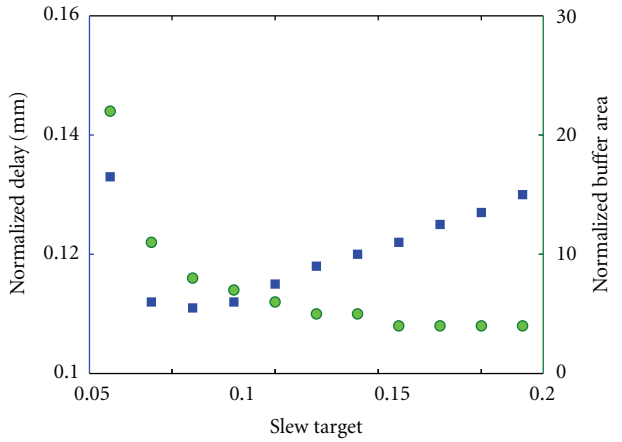


Figure 3: Slew constrait (ns) versus buffer area relationship is shown in green dots. Slew constraint (ns) versus signal delay per mm relationship is shown in blue squares.

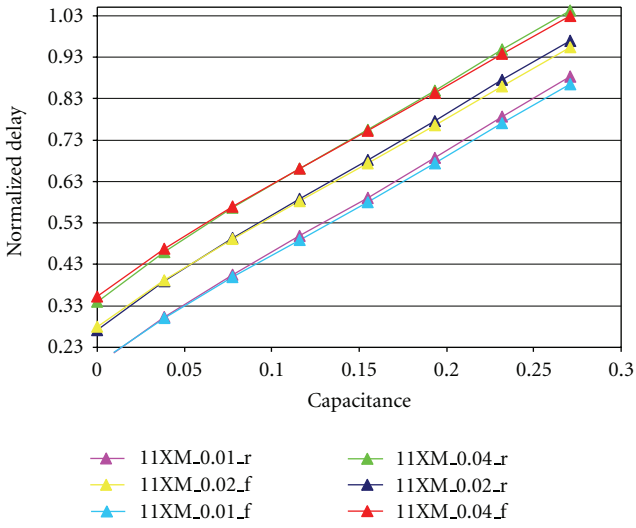


Figure 4: Delay versus cap for a buffer in 45 nm node. Three different input slew values, 10, 20, and 40 ps, are used here. 11XM_0.2_r refers to 11X driving strength buffer, 20 ps input slew and the rising inputs.

Liu and Hu [15] proposed to use dynamic-programming style algorithms for solving discretized gate sizing problem directly. However, the slew impact is still ignored when propagating the solutions in the dynamic programming. Also all previous work tends to optimize the worst critical path, where the sum of negative slacks (referring to Figure of Merit(FOM)) are always ignored, which is also an important factor to measure design quality, especially when the worst critical path stuck during the optimization with either logic structure problems or wrong timing assertions.

The following figures show the relationship between delay and area for one complete buffer library in 65 nm node (Figure 5) and one in 45 nm node (Figure 6). The delay is normalized and the buffer area is measured by its width. The capacitance load is the sum of the capacitance of a typical

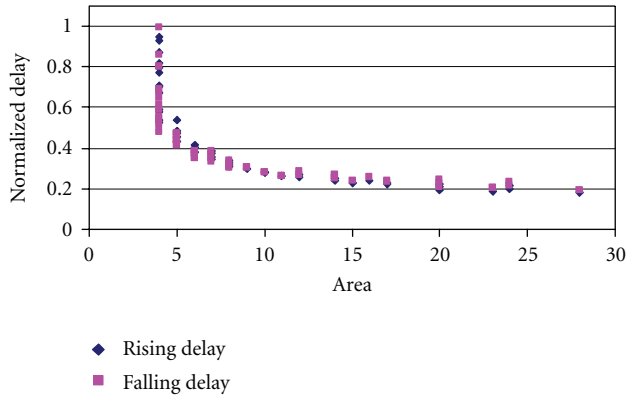


Figure 5: Delay versus area for a buffer library in 65 nm node.

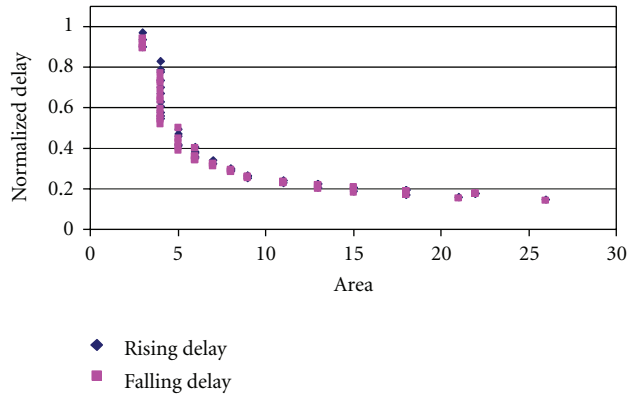


Figure 6: Delay versus area for a buffer library in 45 nm node.

interconnect length plus the capacitance of a typical buffer for the corresponding technology. The input slew is set at 200 ps for 65 nm node and 40 ps for 45 nm node. Both rising and falling delay values are shown in the figures. Note that not only the library is discrete (the area of a gate is generally measured by its width in the standard cell methodologies; since the vertical track is generally fixed) there are also many buffers with the same area (or footprint) but totally different delay values. It is caused by different N/P transistor strength for rising/falling balance or the choice of transistor sizes in the first and second inverters. Therefore, all assumptions such as "convex" and "continuous" do not work for cell based designs, and even rounding approach will meet problems when many gates share the same area. Also, as shown in Figure 4, the delay is sensitive to the input slew too. Such relationship between delay and area is also found in other logic gate library, such as AND/NAND/OR/XOR gates.

1.3. Our Contribution. In this work, we present practical guide and experience of how to put an efficient incremental optimization step inside a physical flow with good area/timing tradeoff. Papers in this category are rarely seen. For a designer or a new EDA startup, we hope this work can provide a quick guide without looking through 100 papers on buffering and gate sizing. Our contributions in this paper are as follows:

(i) an area efficient iterative slew-tighten approach for slew-driven buffering and gate sizing (iterative EVE);

(ii) a simple area efficient timing-driven gate sizing method for cell library designs;

(iii) a new area efficient optimization flow with practical buffering and gate sizing techniques to handle modern design constraints.

2. Overview of Existing Physical Synthesis Flow

A timing-driven physical synthesis flow described in [16] includes the following steps: (1) initial placement and optimization, (2) timing-driven placement and optimization, (3) timing-driven detailed placement, (4) clock insertion and optimization, and (5) routing and post routing optimization; A simple diagram is shown in Figure 7(a) with the placement and optimization part, where refine step refers to optimization at the finer level.

Further, optimization can also be broken into 3 steps shown in Figure 7(b): (1) electrical correction, (2) critical path optimization, and (3) histogram compression.

The purpose of electrical correction is to fix the capacitance and slew violations with buffering and gate sizing. In general, one wants to first perform electrical correction in order to get the design in a reasonable state for the subsequent optimizations. In [17], an electrical violation eliminator (EVE) technique is used to fix electrical violations through fast slew-based gate sizing and buffering.

Then more accurate but slower timing-based buffering approach and gate sizing method are applied in the critical path optimization and histogram compression stages for the rest of the critical paths. During critical path optimization, a small subset of the most critical paths are examined and optimization is performed to improve the timing on those paths. Sometimes, critical path optimization still can not close on timing, and physical synthesis could return its best so far solution. The purpose of the histogram compression phase is to perform optimization on these less critical, but still negative paths.

This flow has the speed and timing performance advantage as shown in [16]. However, it has several drawbacks to cause the area bloat. In the following sections, we describe two main techniques to shed the area bloat for this flow, though the merit of techniques may benefit other flows too.

3. Iterative EVE

The concept of original EVE technique is to combine slew-based gate sizing and slew constrained min-cost buffering [6], and process gates from primary output (or latch inputs) to primary inputs (or latch outputs) in the combinational circuits. If a net has no violations, size the source gate down to save area and reduce load on inputs. If a net has violations, size the source up; if the biggest size cannot fix the slew violation, perform buffering on the net. This approach has several advantages: (1) combining buffering and gate sizing seamlessly, (2) high efficiency, and (3) no reconvergence problem (all decisions are local for electrical corrections).

EVE is motivated by the fact that most nets in modern designs either (1) have only electrical violations but without timing violations, or (2) have positive slacks after electrical violations have been fixed.

More importantly, slew constrained min-cost buffering is much faster than timing-driven buffering because slew constraints can usually be handled locally.

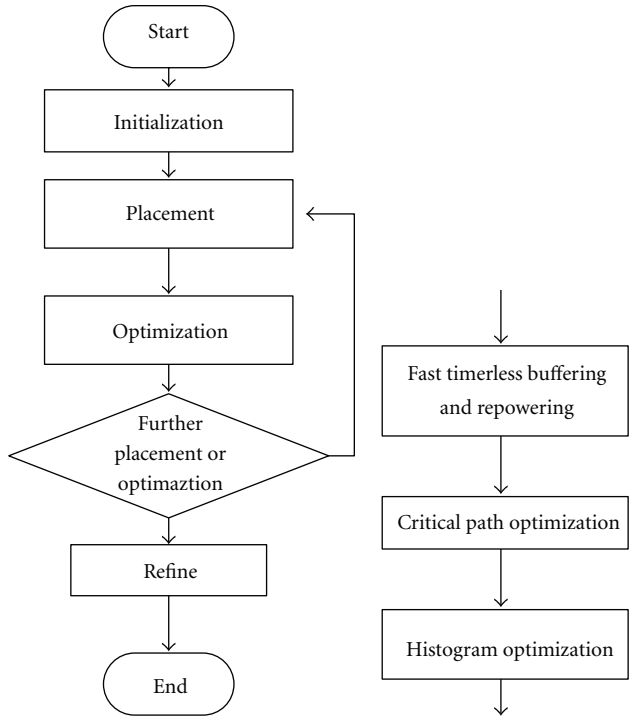
However, as explained in [6], the buffering algorithm is very sensitive to the input slew and the slew target at the sinks. Buffering with a tight slew target (e.g., the slew target is set to be the saturation slew in a long optimal buffered-line [12]) can usually achieve similar timing performance as timing-driven buffering, however, surplus buffer insertion will be done for nets with positive or near-positive slacks and loose original slew targets. On the contrary, using a relaxed slew target can save area but it unacceptably sacrifices performance, and more nets need timing driven buffering afterwards. It also slows down the runtime because timing verification has to be done for each new solution to make sure no timing degradation occurs. Similar situation happens to gate sizing operations, though we use buffering as the main example for the rest of the section. In the following, we propose a new iterative method which gradually tightens slew target.

Instead of starting with a tight slew target, the initial slew target is set based on the operating frequency of the design, which could be the tightest slew constraint for all data phases excluding scan. Sometimes, the initial slew constraint is provided by the designers. Comparing to the initial slew target, each net may have its own slew constraint (e.g., from design rules), and the tighter one is used during optimization for each net. In the first iteration, most of the nets may end up using its own slew constraint, but later in the process, the global slew target gets tighter and eventually overrides the local ones to guide timing optimization.

For the input slew, we start with the saturation slew along a long optimal buffer chain. This is applicable for nets which need the most aggressive buffering. For the other nets, this is a conservative assumption and results in better area since the saturation slew is usually smaller than the initial slew target.

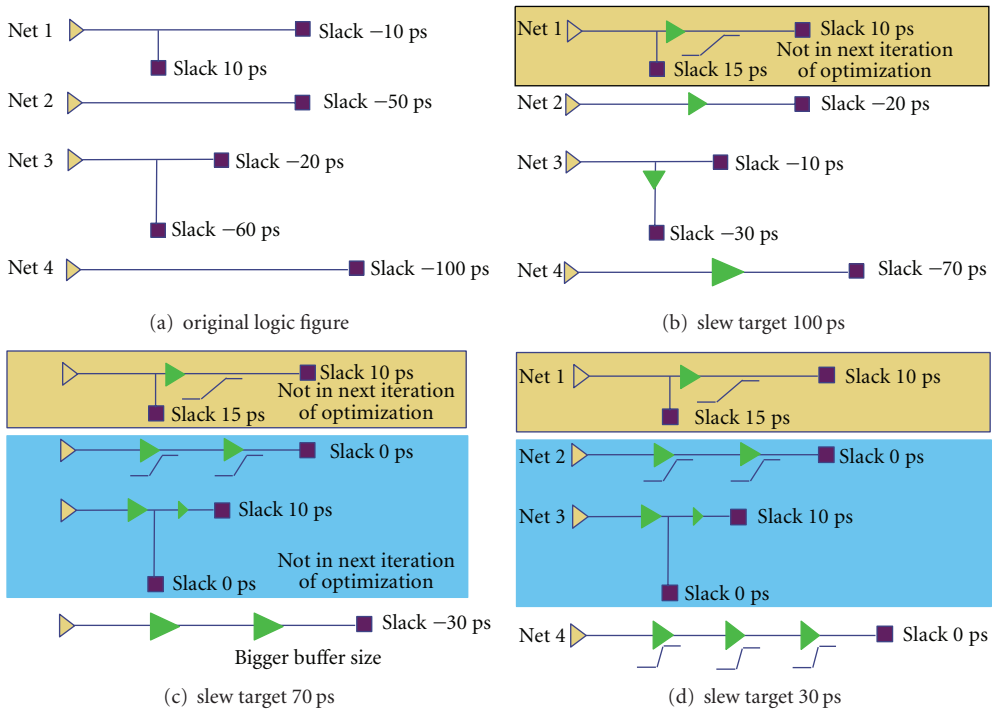
With these settings, we run EVE and follow with a full static timing analysis. Then, the slew target is reduced by a given percentage (say 10% or 20%) and input slew is updated by taking a set of worst paths and averaging their input slew. EVE is run again for all negative nets with the right to left order. In all iterations, buffers on negative paths are completely ripped up and rebuilt. The process is repeated until the slew target is smaller than the saturation slew of an optimal buffered chain. Our experiments generally converge in 3 to 5 iterations for 65 nm and 45 nm technology, and the overhead due to static timing analysis is acceptable. This approach is significantly faster than the traditional timing-driven buffering approach for all nets, which returns similar area results.

An exemplary circuit is shown in Figures 8(a) to 8(d). In this simple example, there are 4 nets and the initial design structure in Figure 8(a) could be an optimized placement



(a) a simple physical synthesis flow (b) detail steps of optimization

FIGURE 7: Flow diagram.



(a) original logic figure (b) slew target 100 ps

(c) slew target 70 ps (d) slew target 30 ps

FIGURE 8: A simple example for iterative EVE.

from a commercial tool, a random placement or a custom placement. In Figure 8(a), there is no repeaters. The slack of all sinks is negative as shown in figure.

Figure 8(b) represents the structure after the first iteration of EVE which generates the first optimized design structure. This iteration uses a global slew target of 100 ps, and a buffer has to be inserted in each net in order to meet the slew target. These buffers may have different sizes in a buffer library. After the first iteration, the slacks of all nets are shown in Figure 8(b). Since the slack of both sinks in the first net become positive, the first net is skipped in the future iterations.

After the first iteration, the slew target is down to 70 ps from 100 ps. Figure 8(c) shows the result of the second iteration of EVE. In this iteration, additional buffers have been inserted into the second, third nets to make their slack positive. They will then be skipped in future iteration.

In the third round, the slew target becomes 30 ps. The optimized design structure is shown in Figure 8(d). Since the first three nets were skipped for this iteration, additional buffer is only inserted in the forth net as shown in Figure 8(d). The final slack of the forth net is 0 ps. This is an ideal case where all nets now have positive slack, and further timing-driven optimization is not necessary.

While the example of Figures 8(a) to 8(d) illustrates only 4 nets and a total of eight inserted buffers, in real designs, the number of nets is typically in the thousands, with the insertion of as many as 500,000 buffers.

With this method, we can insert as fewer buffers as possible to meet timing requirement and the total area and wirelength is greatly reduced.

4. Area Efficient Timing-Driven Gate Sizing

Timing-driven gate sizing is used in the critical path optimization and histogram compression stage. It needs to be accurate, incremental, and harmless.

As discussed in the Section 1.2, the discrete nature of the cell library for standard cell-based designs, the slew impact and the FOM problems, make existing approaches such as convex programming either be unrealistic to use, or inaccurate. Also, previous approaches tend to find the sizing solution for the whole circuit, or a group of hundreds to thousands of gates with internal delay models, and then apply it. The scale of changes, combined with the model inaccuracy, may result in big rejection rate from the static timing analysis with slew propagation, and even some good partial solutions may be thrown away.

In our implementation, we choose to use a simple gate sizing approach. We first give an order of all boxes (could be based on slack or sensitivity), and then work on each single box at each time. After choosing the right power level, perform the incremental timing analysis to update the timing graph, and move on to the next box. It looks like quite naive, but is more accurate for the local change, and also tends to give the best FOM result since even a box is on the not-near-critical paths as long as the slack is still negative and can be improved, a better solution will be chosen. The whole process can also be iterated. One can speed up the process by simply limiting the update scope of static timing analysis engine when every size of the gate is evaluated.

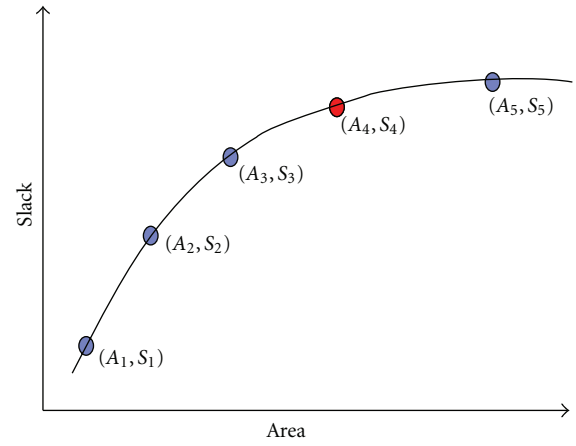


Figure 9: Area aware gate sizing.

We can resize boxes as long as the slack is improved, however, the slack only gets a little bit improvement with lots of area resource sometimes. To be area efficient, the minimum improvement threshold δ is defined as the minimum required slack improvement per unit area change. When a particular gate is resized, we only accept the new solution, if the slack improvement of new solution compared to the previous best solution is bigger than δ times the area difference. As shown in Figure 9, rather than choosing the best slack solution with area A_5, we pick the solution with much smaller area A_4, with acceptable timing degradation.

5. New Area Efficient Optimization Flow

The new flow assembled from iterative EVE is illustrated in Figure 10. The area aware gate sizing is the critical path and histogram stages, where the cost can be tuned for different design requirement.

6. Other Practical Techniques

In any physical synthesis flow, a timing-driven buffering tool plays an essential role. We implement the techniques described in [5] to control the buffer resources, but to make it more practical, several tunings need to be done.

Handling Rising and Falling Transitions. The buffering algorithm needs not only to handle polarities and inverters, also to distinguish the rising/falling signals during the bottom-up dynamic programming since the delay for both edges are noticable different (Figure 4).

Delay Modeling. Elmore delay model is too conservative and causes overbuffering where moment-based computation is too slow. We use scaled Elmore delay model (0.8 as factor) and linear gate delay models, and found the buffer locations are quite close to the solution from the moment-based wire

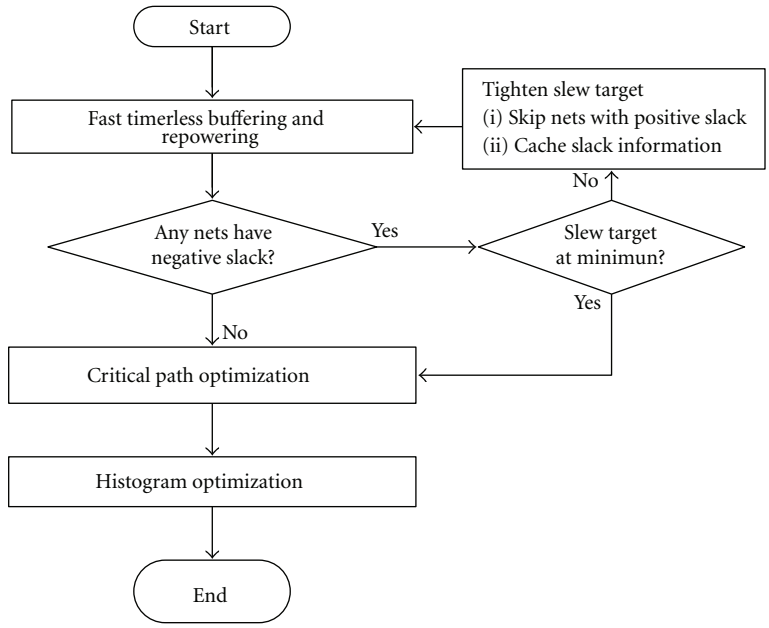


Figure 10: New optimization flow.

delay models and lookup tables-based gate delay models, while the runtime is much faster. We suspect the buffering is a global scaled operation.

Speed. We implement following techniques: range search tree pruning, convex and predictive pruning techniques to gain speedup from solution pruning; buffer solution storage technique in [3] to gains speedup by avoiding naive copying solution list during adding wire operations; layer assignment techniques in [18] to gain speedup by efficiently utilizing available high level metals for each technology.

7. Experiments

We implement the new optimization flow in C++ and embed in the flow shown in Figure 7(a). Twenty industrial designs (eighteen 65 nm and two 45 nm) are selected, ranging from 100,000 to 1 millon gates in the input netlist. All experiments are run on a 2.9GHz machine with Linux operation system.

7.1. Iterative EVE versus Single EVE. In this experiment, we compare our iterative EVE algorithm with the single EVE algorithm which uses the aggressive slew target to get the best timing. Both optimizations are performed after initial placement, which produce a huge slack and FOM improvement since no net (including high fanout nets) has been buffered (other than polarity inverters). We compare the worst slack (WSLK) improvement (the difference of before/after worst slack), FOM improvement, and area increase due to the optimization. Data for only 10 designs are shown in Table 1 to save the space. In summary, iterative EVE approach uses as less as 20% area of single EVE, while achieving similar timing quality. There is runtime overhead since we run multiple iterations, which is generally 2 to 4 times depending on the design.

7.2. Timing-Driven Gate sizing. In this section, experiment results with different minimum improvement threshold δ for timing-driven gate sizing are shown in Table 2. The timing-driven gate sizing is performed in "critical path optimization" stage after iterative EVE, which also explains the scale of the improvement compared to Table 1. The unit of δ is picoseconds per unit area change. When $\delta = 0$, it gives the best timing, and when $\delta > 0$, area cost is considered. From Table 2, the increased area becomes smaller when δ increases. It is interesting to see that when δ is big enough, the area starts to decrease and one can still achieve the worst slack and FOM improvement. Generally we do not use such aggressive minimum improvment threshold like $\delta = 0.05$ in our flow, area saving and final timing performance are considered at the same time. Based on our experimental results, $\delta = 0.005$ is a good choice.

7.3. Overall Flow Comparison. In this part, we put iterate EVE and area aware gate sizing (with $\delta = 0.005$) in the flow and compare to the baseline (single EVE and $\delta = 0$). Both flows go through complete physical synthesis, include 2 iterations of placement (the second one is timing-driven placement with net weight updated according to the timing information), optimizations, timing-driven buffering, detail placement, legalization and the refine part. Both flows also use the practical techniques mentioned in Section 5 too.

For all experiments, we compare area, worst slack (WSLK), FOM, wirelength (WL) and runtime at the end of physical synthesis and the results are shown in Table 3.

From Table 3, our new flow saves 5.8% total area compared to the baseline flow on average, and the maximum

Table 1: The QOR comparison for iterative EVE.

	Type (ns)	FOM Imp (ns)	WSLk Imp	Area Inc
test1	Single EVE	1150890.0	464.87	946003
	Iterative EVE	1153195.6	464.05	198297 (20.96%)
test2	Single EVE	3425630.0	718.20	677286
	Iterative EVE	3344172.7	593.24	189253 (27.94%)
test3	Single EVE	993808.0	125.45	369589
	Iterative EVE	1093409.7	148.62	151493 (40.98%)
test4	Single EVE	8564860.0	426.96	1221551
	Iterative EVE	8378022.8	395.96	490440 (40.14%)
test5	Single EVE	1416620.0	160.65	1611279
	Iterative EVE	1340335.3	167.00	331856 (20.60%)
test6	Single EVE	12550800.0	968.76	811444
	Iterative EVE	12767810.0	1027.0	271742 (33.49%)
test7	Single EVE	9480330.0	369.28	1200267
	Iterative EVE	7593774.2	366.07	454308 (37.85%)
test8	Single EVE	35511100.2	1918.0	1168543
	Iterative EVE	35530547.4	1928.0	414317 (35.45%)
test9	Single EVE	11674800.3	984.47	1103223
	Iterative EVE	11157401.0	1013.3	331287 (30.03%)
test10	Single EVE	66256.2	66.04	326920
	Iterative EVE	64255.5	65.47	93924 (28.73%)

Table 2: The QOR comparison for area-efficient gate sizing.

	δ	WSLK Imp	FOM Imp	Area Inc
test1	0	0.08041	2373.05	200062
	0.005	0.00993	1198.84	62733
	0.010	0.00791	629.81	7217
	0.030	0.00146	268.06	−10162
	0.050	0.00146	252.72	−10370
test2	0	0.03775	1167.43	68958
	0.005	0.00902	646.73	18254
	0.010	0.00813	307.5	1940
	0.030	0.00813	138.93	−1785
	0.050	0.00813	123.46	−1862
test3	0	0.02486	164.51	31861
	0.005	0.00209	91.42	6859
	0.010	0.00209	45.89	325
	0.030	0.00209	19.15	−2575
	0.050	0.00209	19.09	−2593
test4	0	0.06008	309.02	13329
	0.005	0.02325	241.25	3862
	0.010	0.00021	83.67	−120
	0.030	0.00021	39.39	−862
	0.050	0.00021	39.05	−870

area saving is 12.5%. Considering the area is the first-order estimation of the gate power, the number is significant. In addition to total area, we reduce the logic area growth by 12%. Logic area is defined as the amount of area which a physical synthesis tool can "optimize", which excludes fixed and nonsizable cells, such as memory, SRAM, and fixed macroblockages. The logic area growth is (the increased logic area/the initial logic area) × 100%.

As we mentioned before, area bloat also causes the routing and timing problems. The design shown in Figures 1 and 2 is actually the "test1" design in Table 3. As discussed before, the routability of this design is very sensitive to

Table 3: The QOR comparison of baseline and new flow.

Circuit	Type	Area	CPU	WSLK	FOM	WL
test1	base	1.30777	2812.7	−12.738	−115965	46175031
#gates: 102046	new	1.15874	4483.7	0.099	0	18100494
test2	base	20.2130	45229.6	−0.170	−2605	381929854
#gates: 717075	new	19.5043	52505.1	−0.288	−5669	383202138
test3	base	1.92936	5948.8	−0.026	−2	34700143
#gates: 110118	new	1.8862	6326.5	0.100	0	34175838
test4	base	10.9556	17771.0	−0.436	−928	121271648
#gates: 253815	new	10.7207	22537.3	−0.436	−203	116457562
test5	base	5.99029	28559.2	0.099	0	150358750
#gates: 663693	new	5.51185	31977.5	0.096	0	141428293
test6	base	8.29468	13441.5	−0.747	−37	54379592
#gates: 94220	new	8.15084	17410.4	−0.754	−36	50603208
test7	base	16.3154	18454.7	−0.089	−1	273579209
#gates: 367478	new	15.8426	22535.1	0.039	0	257814794
test8	base	7.46228	18598.5	0.012	0	136021272
#gates: 476495	new	7.12724	22208.1	0.013	0	142562718
test9	base	4.81608	14899.8	−0.512	−357	101211621
#gates: 168379	new	4.53251	17373.3	−0.375	−118	89450246
test10	base	5.54470	27398.3	−1.805	−27248	165713654
#gates: 347502	new	4.85059	24215.5	−0.032	−53	90467156
test11	base	9.97560	30120.5	−0.528	−696	151761936
#gates: 517459	new	9.27245	37599.5	−0.525	−726	144139156
test12	base	9.92720	28309.0	−0.347	−117	128984471
#gates: 517583	new	9.27470	36257.1	−0.346	−156	124913294
test13	base	6.04183	18874.4	0.087	0	157218211
#gates: 554423	new	5.82644	20980.1	0.098	0	154116834
test14	base	3.11515	6193.2	0.100	0	33563568
#gates: 142542	new	3.01044	6713.1	0.100	0	28902330
test15	base	9.95405	31284.2	−0.094	−1	269291914
#gates: 797963	new	9.24077	39238.8	−0.057	−22	255280710
test16	base	13.7727	57974.5	−0.743	−4498	404978253
#gates: 1066512	new	12.1618	71217.3	−0.646	−4048	397577302
test17	base	14.4742	30467.7	−0.253	−55	160451598
#gates: 424465	new	13.7434	37764.0	−0.486	−63	142016166
test18	base	5.15163	15858.4	−0.412	−30	158948356
#gates: 416142	new	4.79116	17207.0	−0.403	−29	137246466
test19	base	4.66864	9726.1	−0.200	−58	61616468
#gates: 246524	new	4.59064	11020.5	−0.200	−58	62023304
test20	base	7.00458	27110.2	−0.688	−332	139740251
#gates: 494645	new	6.41238	33254.7	−0.100	−54	130307063

the area. With our new flow, "test1" is not only routable, but also the timing is near to close (the slack threshold is 0.1 ns). Compared to the baseline flow, the wirelength is reduced by 60.8%, and the worst slack is improved by over 12 ns. "test10" shows the similar trend with 45.4% wirelength reduction and 1.8 ns worst slack improvement. On average, our new flow achieves 10.1% wirelength reduction. The worst slack is improved by 770 ps and FOM is improved by 42.9% on average. Out of 20 designs, our new flow gives better slack for 12 designs compared to the baseline flow (same for 2 designs), and gives better FOM for 10 designs (same for 5 designs).

The main reason for the timing and the wirelength improvement is (1) more free space to insert buffers at the desired location or size gate up without moving, (2) better timing before timing-driven placement will make placement move less objects and results in less wirelength increase, and (3) with more freespace, legalization tends to have minimal impact on the wirelength and timing.

8. Conclusion

As we see, timing, congestion, area and power problems are coupled together. By significantly reducing the area bloat, our techniques improve the timing, wirelength, congestion and power consumption as well. Compared to a traditional timing-driven flow, our work achieves 12% logic area growth reduction, 5.8% total area reduction, 10.1% wirelength reduction and 770 ps worst slack improvement on average on 20 industrial designs in 65 nm and 45 nm.

References

[1] L. Trevillyan, D. Kung, R. Puri, L. N. Reddy, and M. A. Kazda, "An integrated environment for technology closure of deep-submicron IC designs," *IEEE Design and Test of Computers*, vol. 21, no. 1, pp. 14–22, 2004.

[2] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '90)*, pp. 865–868, 1990.

[3] W. Shi and Z. Li, "A fast algorithm for optimal buffer insertion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 6, pp. 879–891, 2005.

[4] J. Lillis, C. K. Cheng, and T. T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 3, pp. 437–446, 1996.

[5] Z. Li, C. N. Sze, C. J. Alpert, J. Hu, and W. Shi, "Making fast buffer insertion even faster via approximation techniques," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '05)*, pp. 13–18, 2005.

[6] S. Hu, C. J. Alpert, J. Hu et al., "Fast algorithms for slew-constrained minimum cost buffering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 11, pp. 2009–2022, 2007.

[7] X. Tang, R. Tian, H. Xiang, and D. F. Wong, "A new algorithm for routing tree construction with buffer insertion and wire sizing under obstacle constraints," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD '01)*, pp. 49–56, 2001.

[8] P. Cocchini, "Concurrent flip-flop and repeater insertion for high performance integrated circuits," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '02)*, pp. 268–273, 2002.

[9] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion with accurate gate and interconnect delay computation," in *Proceedings of the 36th Annual Design Automation Conference (DAC '99)*, pp. 479–484, 1999.

[10] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion for noise and delay optimization," in *Proceedings of the 35th Design Automation Conference*, pp. 362–367, 1998.

[11] Z. Li, C. J. Alpert, S. Hu, T. Muhmud, S. T. Quay, and P. G. Villarrubia, "Fast interconnect synthesis with layer assignment," in *Proceedings of the ACM International Symposium on Physical Design (ISPD '08)*, pp. 71–77, 2008.

[12] C. Chu and D. F. Wong, "Closed form solution to simultaneous buffer insertion/sizing and wire sizing," *Proceedings of the ACM Transactions on Design Automation of Electronic Systems (TODAES '01)*, vol. 6, no. 3, pp. 343–371, 2001.

[13] C. P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '98)*, pp. 617–624, 1998.

[14] S. Hu, M. Ketkar, and J. Hu, "Gate sizing for cell library-based designs," in *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC '07)*, pp. 847–852, June 2007.

[15] Y. Liu and J. Hu, "A new algorithm for simultaneous gate sizing and threshold voltage assignment," in *Proceedings of the International Symposium on Physical Design (ISPD '09)*, pp. 27–34, 2009.

[16] C. J. Alpert, S. K. Karandikar, Z. Li et al., "Techniques for fast physical synthesis," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 573–599, 2007.

[17] S. K. Karandikar, C. J. Alpert, M. C. Yildiz, P. Villarrubia, S. Quay, and T. Mahmud, "Fast electrical correction using resizing and buffering," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '07)*, pp. 553–558, 2007.

[18] Z. Li, C. J. Alpert, S. Hu, T. Muhmud, S. T. Quay, and P. G. Villarrubia, "Fast interconnect synthesis with layer assignment," in *Proceedings of the ACM International Symposium on Physical Design (ISPD '08)*, pp. 71–77, April 2008.