COSAT: Congestion, Obstacle, and Slew Aware Tree Construction for Multiple Power Domain Design*

Chien-Pang Lu[1] and Iris Hui-Ru Jiang[2]

[1]MediaTek, Inc., Hsinchu 30078, Taiwan

[2]Dept. of Electrical Eng. & Graduate Inst. of Electronics Eng., National Taiwan University, Taipei 10617, Taiwan

Email: knuth.lu@mediatek.com, huiru.jiang@gmail.com

ABSTRACT

Slew fixing, which ensures correct signal propagation, is essential during timing closure of IC design flow. Conventionally, gate sizing, Vt swapping, or buffer insertion is adopted to locally fix the slew violation on a single gate. Nevertheless, when slew violations are caused by congestion, obstacles, or excessive loadings (e.g., high-fanout nets or long wires), only smart buffering with a global view can fix them. Therefore, in this paper, we propose congestion, obstacle, and slew aware buffered tree construction for excessive loading nets in modern multiple power domain designs. We iteratively cluster sinks into groups by diamond covering and construct Steiner minimal trees. We globally maintain a congestion and obstacle grid map to guide fast grid routing to locate buffers, while avoiding congested regions and obstacles without timing degradation. Our experiments are conducted on seven industrial smartphone designs with TSMC 16/10nm process. Compared with the conventional buffer insertion approach (widely adopted by commercial tools), the minimal chain based approach can reduce 17% buffer count, decrease 14% leakage, and achieve 44% runtime speedup, but incur unwanted timing, design rule, power rule, and routing violations. Our approach can reduce 18% buffer count, decrease 21% leakage, and achieve 92% runtime speedup, while significantly reducing timing, design rule, power rule, and routing short violations. Our results show that our approach is promising for slew fixing on excessive loading nets in modern multiple domain designs.

1 INTRODUCTION

Slew fixing, which ensures correct signal propagation, is essential during timing closure of IC design flow. Slew fixing is beneficial for delay optimization; once slew violations are cleaned, setup time violations, in terms of worst negative slack (WNS) or total negative slack (TNS), can be greatly improved. Conventionally, gate sizing, Vt swapping, or buffer insertion is adopted to locally fix the slew violation on a single gate. Nevertheless, when slew violations are caused by excessive loadings (e.g., high-fanout nets or long wires), congestion, and obstacles, only smart buffering with a global view can fix them.

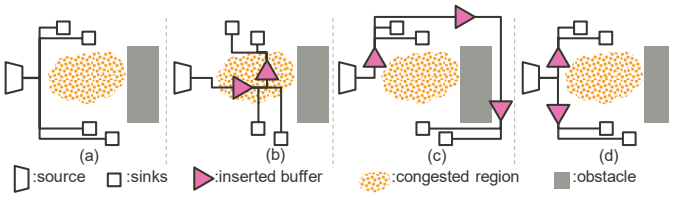

Fig. 1. Buffering. (a) A net with excessive loading around a congested region and obstacles. (b) The inserted buffers cross the congested region thus inducing routing and timing difficulties. (c) The inserted buffers avoid the congested region and obstacles but incur long detours and large buffer area. (d) A congestion, obstacle, and slew aware buffering solution.

Modern low power designs usually adopt multiple power domains to save power by dynamically turning off idle domains; in such designs, a significant number of interconnects cross different domains and incur excessive loadings. Because metal pitches and device sizes continue shrinking as technology advances [1], [2], [3], the affordable wire loading (RC parasitics) of a unit device driving strength decreases, and thus the demand of buffers dramatically increases for slew fixing in multiple domain designs. Moreover, there always exist congested regions and obstacles in these designs. The slew fixing for an excessive loading net becomes even more challenging when it is located in a congested region with obstacles. (see Fig. 1.)

Two more factors further complicate the buffering task in multiple domain designs: First, considering possible working conditions and operating modes, the number of sign-off scenarios could be larger than 200 in 16nm process or beyond. It is common practice to handle only critical scenarios first and then fix remaining scenarios by engineering change order (ECO). Second, when a net passes through some unpowered domains, it may incur signal floating on some unprotected domain inputs. For preventing crossing domain nets from input floating, buffer types should be chosen carefully. Consequently, wisely selecting tree topology and buffer sizes, types, and locations under multiple scenarios determines the buffering quality and timing closure.

Buffer insertion has been extensively studied in literature. Given a tree topology, van Ginneken [4] proposed a classic dynamic programming algorithm for delay optimization, and its extensions [5], [6], [7], [8] were also efficient. Liu et al. [9], [10] applied Lagrangian relaxation to handle timing constraints for general networks (i.e., directed acyclic graphs), but their scalability might be limited by the path reconvergence issue. Chen and Zhou [11] adopted min-cut flow with simplified assumptions. For more accurate estimation, Sze et al. [12] presented a path-based buffer insertion method to reduce the

*This work was supported in part by MediaTek, TSMC, Synopsys, and MOST of Taiwan under Grant MOST 106-2628-E-002-019-MY3.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

DAC '18, June 24–29, 2018, San Francisco, CA, USA

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5700-5/18/06…$15.00

https://doi.org/10.1145/3195970.3196016

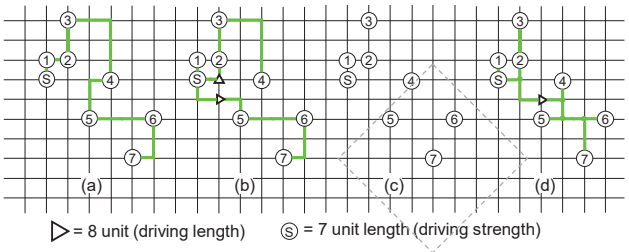

Fig. 2. Minimal chain based buffering in (a)(b) vs. COSAT in (c)(d). (a) Minimal chain. (b) Buffering: 2 buffers, 20 unit wirelength. (c) Diamond covering for sink selection. (d) COSAT: 1 buffer, 15 unit wirelength.

buffer and gate cost for large-scale design, where timing critical paths usually occupy less than 3% of paths in a design, e.g., [13]. For avoiding obstacles, Tang et al. [15] devised a graph-based algorithm to insert buffers from a given set of legal positions. Alternatively, Cong and Yuan [16] proposed a dynamic programming algorithm combined with bottom-up tree construction. For low-power issue, Rao [17] generated tradeoff surfaces in the delay, capacitance, and power space to insert power-aware buffers, but their runtimes might not be acceptable in large-scale designs. For excessive loading nets, Lu et al. [14] iteratively found a minimal chain connecting all sinks and located buffers to drive farthest sinks in reverse order. They did not consider obstacles and restricted subtree topology as a chain. (see Fig. 2.)

Most of these works, however, inserted buffers under fixed buffer locations or a given tree topology. Under these two presumptions, buffering cannot handle congestion and obstacles well, thus inducing timing, area, and power penalties. Moreover, they did not consider multiple scenarios and input floating issues for multiple domain designs.

Therefore, in this paper, we present congestion, obstacle, and slew aware buffered tree construction for excessive loading nets in modern multiple power domain designs. First, we propose a super corner driving strength model which covers multiple scenarios. This model translates a gate driving strength into an effective wirelength. Second, we use diamond covering to iteratively cluster sinks into a group from the farthest sink. We construct a Steiner minimal tree to connect sinks in this group and ensure the tree loading affordable to a buffer. Third, we abstract a congestion and obstacle grid map to guide grid routing to locate a buffer to drive this group while avoiding congested regions and obstacles. Fourth, we remove sinks in the group and introduce a pseudo sink corresponding to the added buffer. Steps 2 to 4 are repeated until no more sinks. Fifth, we choose proper buffer types considering area, leakage saving, and multiple domains. Our experiments are conducted on seven industrial smartphone designs with TSMC 16/10nm process, and our results show that our buffering approach is promising for slew fixing on excessive loading nets in multiple domain designs. Compared with the conventional buffer insertion approach (widely adopted by commercial tools), the minimal chain based approach can reduce 17% buffer count, decrease 14% leakage, and achieve 44% runtime speedup, while incurring unwanted timing, design rule, power rule, and routing violations. Our approach can reduce 18% buffer count, decrease 21% leakage, and achieve 92% runtime speedup, while significantly reducing timing, design rule, power rule, and routing short violations.

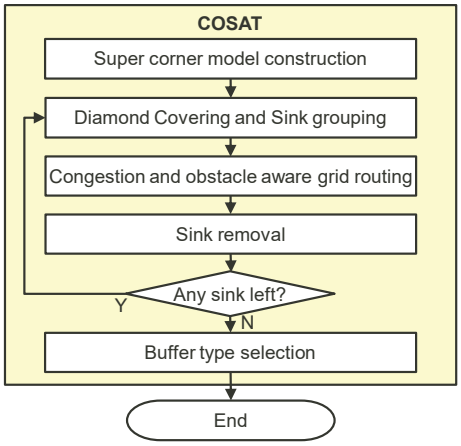

Fig. 3. Overview of COSAT.

2 PROBLEM FORMULATION

Design rule constraints check maximum transition (slew), maximum capacitance (loading), and maximum fanout for all nets in a design. Generally, if slew and loading meet requirements, fanout is a loose constraint. As mentioned earlier, slew fixing for excessive loading nets is challenging for multiple power domain designs. In these designs, congestion, obstacles, multiple scenarios, and input floating issues should be considered during buffering. Hence, in this paper, we address the excessive loading net buffering problem which can be formulated as follows:

The Excessive Loading Net Buffering Problem: Given slew violating nets, the slew constraint, a buffer library, congested regions and obstacles in a placed multiple domain design, our goal is to construct buffered trees for these violating nets such that buffer usage is minimized, the slew constraint is satisfied under all scenarios, and input floating is avoided.

3 CONGESTION, OBSTACLE, AND SLEW AWARE BUFFERED TREE CONSTRUCTION

This section details our congestion, obstacle, and slew aware buffered tree construction approach, COSAT.

3.1 Overview

Different from previous works, which presume fixed buffer locations or a given tree topology, in this paper, we concurrently design the tree topology and insert buffers avoiding congested regions and obstacles for excessive loading nets.

Fig. 3 shows the overview of our approach. 1) We construct a super corner gate driving strength model covering all scenarios; this model translates a gate driving strength into an effective wirelength. 2) For each slew violating net, we iteratively cluster sinks into a group from the farthest sink. We perform online Steiner minimal tree (OSMT) to incrementally connect sinks in this group and check if the corresponding loading is acceptable according to the super corner model. 3) We globally maintain a congestion and obstacle grid map to guide fast grid routing to locate a buffer to drive this group while avoiding congested regions and obstacles. 4) We remove sinks in the group. If there are no more sinks, we go to the next step; otherwise, we introduce a pseudo sink corresponding to the added buffer and

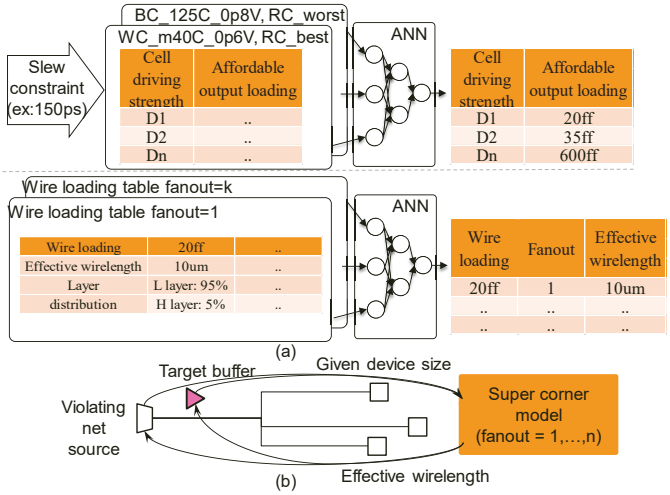

Fig. 4. Super corner model construction and usage. (a) Super corner model contains two parts: The translation from gate driving strength to affordable output loading under a given slew constraint, and the translation from wire loading to effective wirelength. (b) Querying the super corner model to obtain effective wirelength for an investigated cell driving strength.

go back to step 2. 5) We choose proper buffer sizes and types with area, leakage, and input floating consideration. We follow the same flow to buffer all slew violating nets. The super corner model is constructed only once and shared by all violating nets.

3.2 Super Corner Model Construction

Timing check over multiple scenarios is required for timing closure. Because a modern multiple power domain design may contain over 200 scenarios, the common practice performs buffering for slew fixing under the most critical scenario and then adjusts the solution to fix remaining scenarios by ECO.

For expediting timing closure, in contrast, we construct a super corner model to estimate a gate driving strength in terms of wirelength covering all scenarios. The output loading of the driving gate (source) of a net is contributed by its fanout gate (sink) input capacitance and wire loading. Our super corner model thus contains two parts: We extract one model for affordable output loading (corresponding to driving strength) and one model for effective wirelength (corresponding to wire loading).

First, we determine the affordable output loading of a cell satisfying the given slew constraint under all corners. Inspired by [14], considering some corner, the affordable output loading makes the output slew equal input slew, and the slew value is less than maximum slew. The loading value can be obtained by binary search on the output slew lookup table of this cell. Different process/voltage/temperature (PVT) conditions correspond to different affordable output loading values. As shown in Fig. 4(a), under a given slew constraint, the super corner driving strength model, which can be constructed based on artificial neural networks (ANNs) covering multiple corners, translates a cell driving strength into an affordable output loading value.

Second, we convert an allowable wire loading to an effective wirelength. By analyzing routing results from previous designs, we extract a wire loading table to record the relationship between wire loading, effective wirelength, metal layer distribution for each possible fanout value. Fig. 5(a) shows the

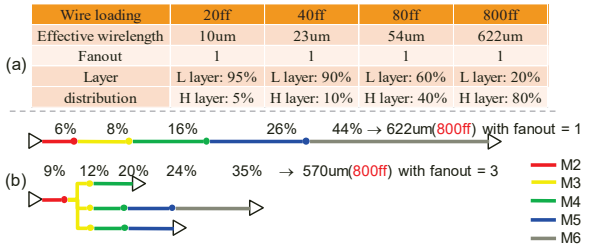

Fig. 5. Wire loading table. (a) Single fanout net. (b) Different fanouts under the same wire loading.

wire loading statistics for single fanout nets under some design corner. A modern router tends to utilize more high metal layers for large loading nets (i.e., long wires for single fanout nets). For different fanout values under the same wire loading, Fig. 5(b) shows large fanout nets utilize more low layer segments, i.e., greater resistance but shorter wirelength. Based on the profile under different fanout values and corners, we train a super corner wire loading model as shown in Fig. 4(a).

As shown in Fig. 4(b), at the subsequent sink grouping step, for a target buffer cell (or net source gate) and sink candidates, by checking the super corner model, we first obtain the affordable output loading and then query the effective wirelength according to the allowable wire loading (i.e., the affordable output loading minus total sink gate input capacitance). Furthermore, because of sink grouping, the fanout value of each added buffer or the net source gate is well controlled. To reduce the model construction time, super corner models are prepared for fanout less than a threshold value. Extrapolation is adopted for higher fanout subnets (only few in our experiments).

3.3 Diamond Covering and Sink Grouping

Previous work [14] uses minimal chain to cluster sinks in an intuitive way: The minimal chain based approach starts from the net source, iteratively connects to the nearest sink, and forms a chain. However, minimal chain may result in a buffered tree with pessimistic wirelegnth and thus possibly requires more ECO efforts for timing closure. In contrast, we use two techniques to perform sink grouping. Basically, we iteratively cluster sinks into a group from the farthest sink. First, we use a diamond shaped polygon to quickly identify sink candidates to form a group. Second, we perform online Steiner minimal tree (OSMT) [18] to incrementally connect sink candidates and check if the corresponding loading of the current tree is acceptable according to the super corner model. Temporarily, the loading test is based on the driving strength of a predefined target buffer cell; subsequently, buffer type selection further optimizes the size and type of each added buffer.

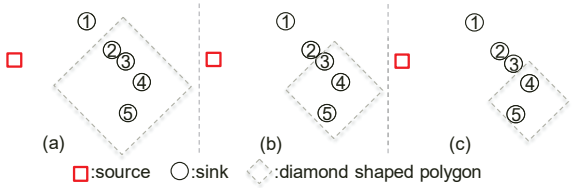

Fig. 6. Diamond covering. (a) Large diamond: sink candidates = {2, 3, 4, 5}. (b) Medium diamond: sink candidates = {3, 4, 5}. (c) Small diamond: sink candidates = {4, 5}.

Conceptually, sinks in geometric proximity (in terms of Manhattan distance) should be clustered together and driven by the same buffer. Thus, we use a diamond shaped polygon to identify sink candidates. A larger diamond may cover more sinks, thus leading to longer OSMT construction time but a smaller buffer count, and vice versa. (see Fig. 6.) We prefer each sink group of a proper size for a good tradeoff between runtime and buffer count. Fig. 7 shows an example. Consider a 150ps slew constraint, a target buffer cell BUFD16. Assume that according to super corner model (e.g., Fig. 4), we obtain that affordable output loading, corresponding to 200 fanouts plus 400um wire loading. Then, we start from checking the longest possible radius; in this case, all sinks lie in a monotonic path, i.e., the distance from the driving buffer to farthest sink should be 400 um. In most of cases, the radius is usually smaller. After performing binary search, we set radius as 75 um.

After collecting sink candidates, we construct an OSMT incrementally from the farthest sink. Every time, we add a sink, incrementally update the OSMT, and do the loading test. Once the accumulated loading exceeds the affordable output loading of the target buffer cell, the sink grouping process terminates. (see Fig. 8(a)-(d).) In Section 3.4, grid routing determines the buffer location and connects the buffer with the final sink included in the sink group. After grid routing, we remove sinks in the group and introduce a pseudo sink corresponding to the added buffer. ((see Fig. 8(e)-(f).)

3.4 Congestion and Obstacle Grid Map and Fast Grid Routing

After sink grouping, we perform fast grid routing to determine a buffer location to drive the group as well as to connect the buffer to the corresponding Steiner tree of the group. For avoiding congested regions and obstacles, we globally maintain a congestion and obstacle grid map to guide grid routing.

First, we construct a congestion and obstacle grid map as follows. An initial uniform grid map is constructed as shown in Fig. 9(a). Grid points overlapping with obstacles are removed by Boolean operation, and the utilization (i.e., a factor combining area utilization and routing utilization) is recorded at each grid point as shown in Fig. 9(b).

Second, we determine the buffer location as follows. Basically, we prefer a buffer location which facilitates subsequent sink grouping (e.g., some point closer to the net source tends not to

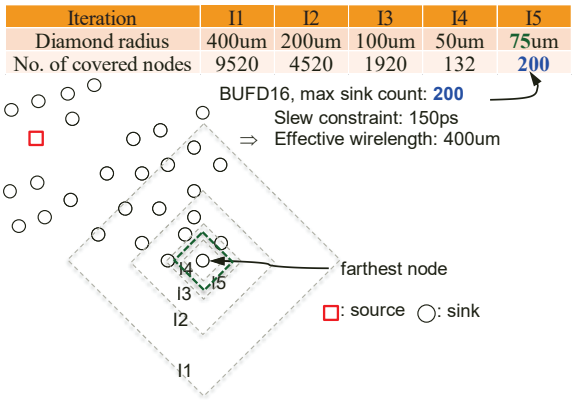

Fig. 7. Binary search for a proper diamond radius.

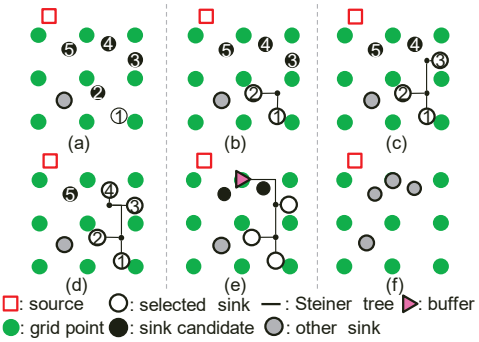

Fig. 8. Online Steiner minimal tree construction for a sink group. (a) Sink 1 is selected. (b) Sink 2 is selected, and OSMT is constructed. (c) Sink 3 is selected, and OSMT is updated. (d) Sink 4 is selected and then rejected because of exceeding affordable output loading. (e) Grid routing locates a buffer to drive sinks 1, 2, and 3. (f) A pseudo sink is introduced, and sinks 1, 2, and 3 are removed.

lengthen the total wirelength). Thus, for each grid point, we compute the routing cost from it to the net source (in terms of grids). The cost computation can be done by Dijkstra shortest path algorithm, where the grid point nearest to the net source is associated with 0 routing cost. (The cost is indicated by the number of each point in Fig. 10(a).) Because the loading of a constructed Steiner tree may not consume all affordable output loading of a target buffer cell, the residual output loading can be used to connect the buffer and the tree. The residual output loading equals the affordable output loading minus the loading contributed by the Steiner tree. Similar to effective wirelength queried from the super corner model, the residual output loading corresponds to a residual wirelength. We identify buffer location candidates within the residual wirelength range with respect to the final included sink. (Candidates are the grid points inside the

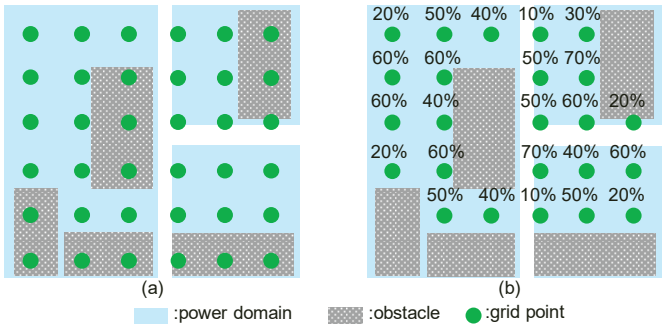

Fig. 9. Grid map. (a) Initial grid map. (b) Congestion and obstacle grid map.

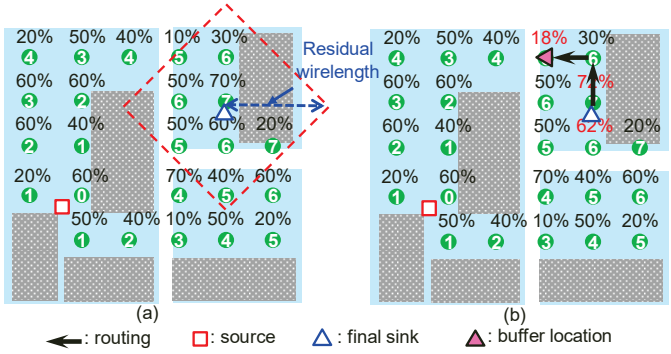

Fig. 10. Grid routing. (a) Identifying buffer location candidates. (b) Determining the buffer location, connecting the final included sink and the buffer, and updating grid map.

dashed diamond in Fig. 10(a)). The buffer location is at the grid point with minimum routing cost to source as well as minimum grid utilization, i.e.,

$$\arg\min_i \left(routing\_cost\_source\_i, grid\_utilization\right).$$

Third, we connect the buffer with the final included sink in the Steiner tree. The final included sink is connected to its nearest grid point, and grid routing (maze routing) is performed from the nearest grid point to the buffer location, where the grid routing cost equals the total distance from the starting grid point and accumulated grid utilization of passed grid points. Grid utilization of each passed grid point is then updated. (see Fig. 10(b).) The grid routing is fast because it is confined within the residual wirelength range of the final included sink.

In addition, for the finally formed sink group, if the net source can drive this group, we do not insert a buffer, and grid routing connects the source and the final included sink instead.

3.5 Buffer Type Selection

After sink grouping and grid routing are done, the tree topology and buffer locations are determined. Both are determined based on a predefined target buffer cell. (see Fig. 11(a).) We further choose proper sizes and types for area, power, and input floating consideration. First, because some buffers may not consume their affordable output loading, we may adjust their sizes for area and power optimization. (see buffer b1 in Fig. 11(b).) Second, if an inserted buffer and driven sinks are located in different power domains, this buffer should be replaced with an always-on buffer of the same driving strength for preventing input floating. (see buffers b3 and b4 in Fig. 11(b).) Because the tree topology and buffer locations are fixed at this step, the size adjustment and type change can be optimally done in a bottom-up fashion.

4 EXPERIMENTAL RESULTS

We implemented our approach in the C++ with Tcl language on a 2.9 GHz Intel Xeon Linux workstation with 128GB memory. Table 1 lists benchmark circuit statistics, including seven industrial smartphone designs, four with 16nm and three with 10nm TSMC process. 'No. of Instances' denotes the number of instances in a design, 'Area' the standard cell area, 'No. of Power Domains' the number of power domains, 'No. of Violating Nets' the number of slew violating nets, 'No. of Violating Sinks' the total number of sinks from violating nets, and 'No. of Valid Grids' the number of valid grids in a congestion and obstacle grid map. In our experiments, all designs are placed with timing optimization by commercial tools [19], [20]. After placement, there still are slew and setup time violations in these designs.

Table 2 compares the conventional buffer insertion approach (CA), the minimal chain based approach (MC) [14], and our COSAT method. The buffering results of the conventional

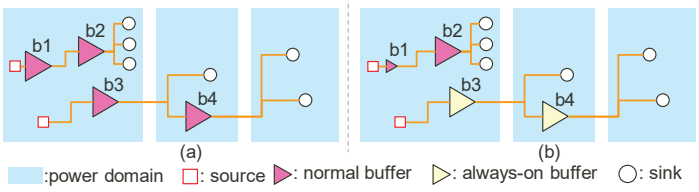

Fig. 11. Buffer type selection.

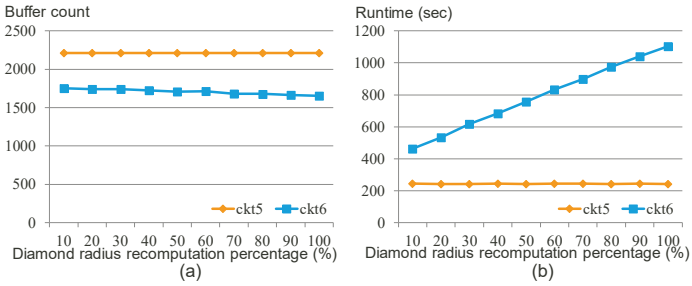

Fig. 12. Statistics of buffer count and runtime under different diamond radius recomputation percentage.

approach are generated by buffer optimization provided by commercial tools, while those of minimal chain based buffering are generated by the binary provided by the authors of [14]. After buffering, we complete physical design flow by commercial tools and report the detailed routing results of three methods. 'No. of Buffers' denotes the number of inserted buffers, 'No. of Over Obstacle Buffers' the number of buffers overlapping with obstacles, 'WNS (ns)' the worst negative slack of setup time checking, 'No. of DRVs' the number of design rule violations, 'No. of shorts' the number of routing short violations, and 'Runtime (sec)' the execution time. Overall, COSAT outperforms the conventional approach and the minimal chain based approach; for each design, our approach allocates the least number of buffers, avoids congested regions and obstacles, completes buffering in the shortest runtime. Meanwhile, compared with these two methods, along with removing slew violations, we can significantly improve worst negative slacks and have no power rule violations. (These short and design rule violations (totally less than 10) are easy to be fixed in practice.) It can be seen that our approach is promising to be incorporated into the practical physical design flow. The conventional approach uses global routing to find the routing paths and estimate the corresponding delay for buffer insertion. This step usually incurs longer runtime and more design rule and short violations. Because the minimal chain based approach [14] does not consider obstacles, the inserted buffers may be located in illegal positions, and thus after detailed routing, the number of slew, slack and short violations are significant.

In our experiments, our approach determines the diamond radius (for diamond covering) at only the first iteration (the first group forming) instead of at each iteration. Fig. 12 reports the statistics of buffer count and runtime under diamond radius recomputation percentage (i.e., the percentage of radius recomputation iterations). Circuit 'ckt5' is a general case mixed with long wires and high-fanout nets, while circuit 'ckt6' has an extremely high-fanout net. Fig. 12(a) shows the buffer count remains almost the same while recomputing radiuses. Fig. 12(b) shows the runtime overhead of radius recomputation is large for 'ckt6'.

Fig. 13 shows our results of the extremely high-fanout net in circuit 'ckt6'. Fig. 13(a) shows the inserted buffer locations (black spots), and Fig. 13(b) shows the connection of the buffered net. Circuit 'ckt7' is a congested design. Fig. 14 shows the congestion maps of circuit 'ckt7' generated by the conventional approach and COSAT. It can be seen that congestion is well-handled by our approach.

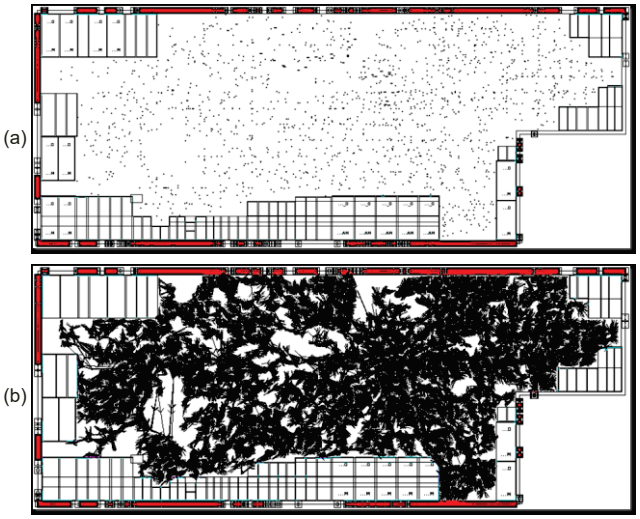

Fig. 13. An extremely high-fanout net (scan enable) with 46,212 sinks in circuit ckt6. (a) Buffer locations (indicated by black spots). (b) Buffered net connections.

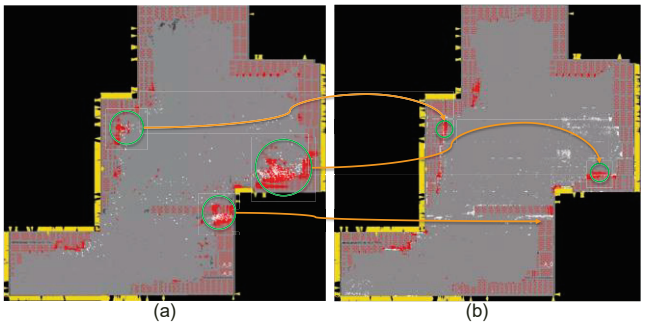

Fig. 14. Congestion maps of circuit ckt7. Congestion hotspots are indicated by red spots, and inserted buffers are indicated by white spots. (a) The result of the conventional approach. (b) Our result.

Table 1: Benchmark Statistics

Circuit (Process)	No. of Instances	Area (um^2)	No. of Power Domains	No. of Violating Nets	No. of Violating Sinks	No. of Valid Grids
ckt1 (16)	720K	1073K	2	547	160477	35783
ckt2 (10)	1926K	733K	2	1246	18109	4767
ckt3 (16)	1280K	1940K	3	61	18087	189
ckt4 (10)	1802K	488K	2	172	105270	392
ckt5 (16)	1297K	486K	2	889	7135	3375
ckt6 (16)	1036K	495K	2	1	46212	467
ckt7 (10)	3179K	1277K	2	6985	30861	687

Table 2: Comparison with Conventional Approach (CA) and the Minimal Chain Based Approach (MC)

Circuit	Method	No. of Buffers	Buffer leakage (nW)	No. of Over Obstacle Buffers (Paths)	WNS (ns)	No. of DRVs	No. of Shorts	Runtime (sec)
ckt1	CA	7454	4159.3	0 (0)	0.0	23	0	5683
	MC	7428	4144.8	4742 (328)	-2.8	5126	720	8236
	COSAT	7296	3972.2	0 (0)	0.0	0	0	1537
ckt2	CA	8929	4982.4	0 (0)	-0.3	29	129	4627
	MC	8813	4917.7	3721 (492)	-3.4	4076	923	742
	COSAT	8777	4597.3	0 (0)	-0.1	2	0	213
ckt3	CA	959	535.1	0 (0)	-0.1	12	0	5128
	MC	905	505.0	75 (28)	-0.8	83	16	97
	COSAT	893	498.3	0 (0)	0.0	0	0	15
ckt4	CA	1428	796.8	0 (0)	-2.2	89	45	4130
	MC	1242	693.0	198 (18)	-3.7	367	71	7129
	COSAT	1165	650.1	0 (0)	0.0	0	0	843
ckt5	CA	2597	1449.1	0 (0)	-0.1	18	5	3810
	MC	2152	1200.8	426 (240)	-1.2	623	53	432
	COSAT	2210	1198.2	0 (0)	0.0	0	0	247
ckt6	CA	1698	947.5	0 (0)	0.0	34	0	5527
	MC	1655	923.5	0 (0)	-0.3	51	32	4325
	COSAT	1629	909.0	0 (0)	0.0	0	0	312
ckt7	CA	15928	8887.8	0 (0)	-4.2	1282	4258	9132
	MC	10199	6327.0	4141 (3636)	-3.9	5231	7242	310
	COSAT	10316	5456.3	0 (0)	0.0	0	4	183
Ratio	CA	1.00	1.00	-	1.00	1.00	1.00	1.00
	MC	0.83	0.86	-	2.33	10.46	2.04	0.56
	COSAT	0.82	0.79	-	0.01	<0.01	<0.01	0.08

5 CONCLUSION

In this paper, we have proposed a congestion, obstacle, and slew aware buffered tree construction approach for excessive loading nets in modern multiple power domain designs. We iteratively cluster sinks into groups by diamond covering and construct Steiner minimal trees. We globally maintain a congestion and obstacle grid map to guide fast grid routing to locate buffers, while avoiding congestion and obstacles without timing degradation. Based on experiments conducted on seven industrial smartphone designs with TSMC 16/10nm process, our results show that our approach outperforms the conventional buffer insertion approach and previous minimal chain based approach. Along with removing slew violations, we can significantly improve worst negative slacks and have no power rule violations, promising to be incorporated into the practical physical design flow.

REFERENCES

[1] P. Saxena, N. Menezes, P. Cocchini and D. A. Kirkpatrick, Repeater scaling and its impact on CAD. IEEE. TCAD, 23(4), pp. 451–463, 2004.

[2] J. Cong, L.He, C.-K. Koh, and P. H. Madden, Performance optimization of VLSI interconnect layout. Integration, VLSI Journal, 21(1-2), pp. 1–94, 1996.

[3] M. Kim, B. G. Ahn, J. Kim, B. Lee, and J. Chong, Thermal aware timing budget for buffer insertion in early stage of physical design. Proc. ISCAS, pp. 357–360, 2012.

[4] L. P. P. P. van Ginneken, Buffer placement in distributed RC-tree networks for minimal Elmore delay. Proc. ISCAS, vol. 2, pp. 865–868, 1990.

[5] W. Shi and Z. Li, A fast algorithm for optimal buffer insertion. IEEE TCAD, 24(6), pp. 879–891, 2005.

[6] S. Hu, Charles J. Alpert, J. Hu; S.-K. Karandikar, Z. Li, W. Shi, C.-N. Sze, Fast algorithms for slew-constrained minimum cost buffering. IEEE TCAD, 26(11), pp. 2009–2022, 2007.

[7] R. Murgai, Layout-driven area-constrained timing optimization by net buffering. Proc. ICCAD, pp. 379–386, 2000.

[8] R. Murgai, Improved layout-driven area-constrained timing optimization by net buffering. Proc. ICVD, pp. 92–102, 2005.

[9] I.-M. Liu, A. Aziz, D.-F. Wong and H. Zhou, An efficient buffer insertion algorithm for large network based on Lagrangian relaxation. Proc. ICCD, pp. 210–215, 1999.

[10] I.-M. Liu, A. Aziz and D.-F. Wong, Meeting delay constraints in DSM by minimal repeater insertion. Proc. DATE, pp. 436–440, 2000.

[11] R. Chen and H. Zhou, Efficient algorithms for buffer insertion in general circuits based on network flow. Proc. ICCAD, pp. 322–326, 2005.

[12] C.N. Sze, C. J. Alpert, J. Hu, and W. Shi, Path-based buffer insertion. IEEE TCAD, 26(7), pp. 1346–1355, 2007.

[13] X. Lou, Y.-J. Yu, P.-K. Meher, Lower bound analysis and perturbation of critical path for area-time efficient multiple constant multiplications. IEEE TCAD, 36(2), pp. 313–324, 2017.

[14] C.-P. Lu, Mango C.-T. Chao, C.-H. Lo, and C.-W. Chang, A metal-only-ECO solver for input-slew and output-loading violation. Proc. ISPD, pp. 191–198, 2009.

[15] 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. Proc. ICCAD, pp. 49–56, 2001.

[16] J. Cong and X. Yuan Routing tree construction under fixed buffer locations. Proc. DAC, pp. 379–384, 2000.

[17] R.-R. Rao, D. Blaauw, D. Sylvester, C. J. Alpert, and S. Nassif, An efficient surface-based low-power buffer insertion algorithm. Proc. ISPD, pp. 86–93, 2005.

[18] A. Gupta and A. Kumar, Online Steiner tree with deletions. Proc. SODA, pp. 455–467, 2014.

[19] IC Compiler, Synopsys, Inc.

[20] Innovus, Cadence, Inc.