Gate Planning During Placement for Gated Clock Network

Weixiang Shen, Yici Cai, Xianlong Hong
EDA Lab, Dept.of Computer Science and Technology
Tsinghua University, Beijing, 100084, China
cwx04@mails.tsinghua.edu.cn

Jiang Hu
Dept.of Electrical and Computer Engineering
Texas A&M University, College Station, TX 77843, USA
jianghu@ece.tamu.edu

***Abstract*— Clock gating is a popular technique for reducing power dissipation in clock network. Although there have been numerous research efforts on clock gating, the previous approaches still have a significant weakness. That is, they usually construct a gated clock tree after cell placement, i.e., cell placement is performed without considering clock gating and may generate a solution unfriendly to subsequent gated clock tree construction. As a result, the control gates inserted in the tree construction is very likely to cause cell overlap. Even though the overlap can be eventually removed in placement legalization, remarkable wirelength/power overhead is incurred. In this paper, we propose a gate planning technique which is integrated with a partition-based cell placer. During cell placement, the planning judiciously inserts clock gates based on power estimation. In addition, pseudo edges are inserted between clock gates and registers in order to reduce clock wirelength and enable long shut-off periods. At the end, when a relatively detailed placement is obtained, a post-processing is performed to degrade the inefficient clock gates to clock buffers. We compared our approach with recent previous works on ISCAS89 benchmark circuits. Our method reduces the clock tree wirelength and power by 22.06% and 40.80%, respectively, with a very limited increase on signal nets wirelength and power compared with the conventional (register-oblivious) placement. The results also indicate that our algorithm outperforms the clock-gating-oblivious placement [9] on power reduction and performance improvement.**

I. Introduction

Clock gating has been widely applied as an effective technique to reduce the dynamic power of clock network [1]-[9]. It is based on the observation that some portions of the circuit may be idle at a certain time. Then, the part of clock network corresponding to the idle portions can be shut off through logic masking such as AND gate. Evidently, shutting off parts of the clock network can reduce the dynamic power dissipation. The set of active/idle times for the circuits is referred as activity pattern in [1][2], which is a string of 1s and 0s. Based on the definition of activity pattern, [2] proposed an algorithm for constructing the topology of gated clock tree to minimize the total power including the clock network and gate logics. However, the routing overhead and power dissipation of the control signals for the gates were not modeled. In [3][4], Oh et al. implemented zero skew gated clock routing that improved upon [2] by accurately accounting for the power of the clock tree and the gate control signals. In [5][6], Donno et al. proposed a clock tree planning methodology that constructed gated clock tree topology with consideration of both switching activities and cell locations.

Although the previous approaches are effective, their efficiency on power reduction is limited by a weakness. That is, they construct gated clock tree after conventional cell placement, which does not consider clock gating. Consequently, their tree construction sometimes is based on a placement unfriendly to clock gating and therefore results in limited power reduction. There are a few cell placement methods [10]-[12] considering clock network design, but these methods do not consider clock gating. Clock gating requests that registers with similar switching activities are placed close to each other [8][9]. Obviously, a clock-gating-oblivious placer may ignore this request and place registers with similar activities far apart. For such placement, it is difficult for clock gating to reduce power efficiently. The method introduced in [8] can obtain the optimal placement of registers by pulling and clustering the registers with the similar activity patterns during placement. But it introduced many gate logics after the gated clock tree construction and needed resort to Engineering Change Order (ECO) or incremental placement to remove the overlaps. In order to solve this problem, [9] proposed a new design flow, which contained two stages of placement. In the initial placement, it clustered and pulled the registers, especially for the registers with the similar activity patterns. Then zero skew gated clock tree construction and optimization were implemented to get the optimal gated clock network which determined the exact positions of the inserted gate logics and their interconnect information with the registers. After that, an incremental placement was invoked to remove the overlaps introduced by the gate logics while preserving the optimal gated clock network constructed before.

II. Motivation

In order to shut off the clock signal more cycles for the gated clock tree, [8][9] took this concern to placement stage. By clustering and pulling the registers, especially for the registers with the similar activity pattern, they achieved more optimal position of the registers and indeed decreased clock tree wirelength and power. However, these methods only considered the gated clock tree in advance, they did not do gate planning in placement, and the exact gates' positions were obtained in clock tree construction after placement, so it inevitably needed post placement or incremental placement to deal with the inserted gate logics.

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

On the other hand, although [9] effectively decreased the clock tree wirelength and power, the small overheads of the total half perimeter wirelength (HPWL) and the power of signal nets could not be neglected. Especially for the signal nets power, even though it increased within 3% comparing to [8], it still could not be ignored since the signal nets power accounts for large weight in total power budget, so the total power reduction obtained by [9] was still small. In addition, the increase of HPWL makes the routing area more congested. Moreover, in order to remove the overlaps introduced by the gate logics after gated clock tree construction and routing, it needs feed the gated clock network information and the gate logics back to the previous design and then implements incremental placement (as shown in Fig.1). These two stages of placement take considerable run time, which is more obvious for the circuit with large scale.

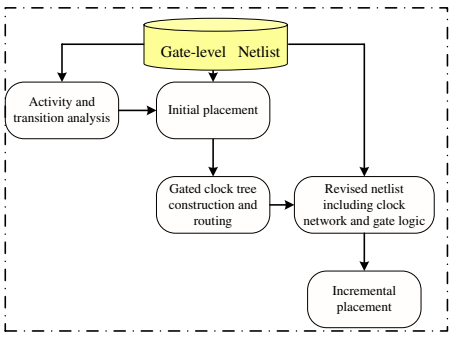

Fig. 1. Design flow of [9]

Furthermore, in the gated clock tree construction after placement, assuming a gate is inserted before each register (or the sink of the clock tree) at the beginning and then followed by a heuristic gate moving [5][6][8][9] is somewhat ad hoc, which will result in redundant gate logics and not all of the resultant gates are in the optimal position. There are two reasons behind. First, inserting a gate before each register may not reduce the power, sometimes it even increases it inversely if the register is active at most of the cycles, then there is few chance to shut off the clock signal. According to the power model, only when the power introduced by the gate logic and its control signal is smaller than the power saved by gating, it is worth while to insert a gate. Second, the upper level nodes have less chance to shut off the clock signal, as the activity of the node increases monotonically as we go up the tree, so it becomes difficult to move the gate towards the upper level.

So in this paper, we integrate gate planning with a multilevel cut-based placer to decrease total power and the run time. As the partition goes during the placement, we dynamically insert and delete gate among the registers to derive an optimal gated clock topology for each local clock tree. After placement, at the process of zero skew clock routing using deferred merge embedding (DME) algorithm [13], we re-evaluate the effectiveness of the inserted gate and delete the useless gates or treat them as buffers. The experimental results on ISCAS89 benchmark circuits show the effectiveness and efficiency of our algorithm. Compared with the conventional design flow, which constructs gated clock tree after general placement, our method decreases the clock tree wirelength, clock tree power, gate number and total power by 22.06%, 40.80%, 60.02% and 13.07% on average, respectively. Compared our approach with a recent previous work, which constructed gated clock tree after a register-aware but clock-gating-oblivious placement [9], it decreases the clock tree power, gate number, HPWL, total power and total run time by 29.74%, 43.01%, 6.90%, 13.79% and 51.43% with an increase of clock tree wirelength within 8.95% on average.

The rest of the paper is organized as follows: in Section III we will introduce the placer and the power model used in our work. In Section IV we briefly describe our design flow. Section V presents the gate planning during placement in detail. Zero skew clock tree construction and gate removal will be described in Section VI. Section VII gives the experimental results, and we conclude the paper in Section VIII.

III. Preliminaries

A. Multilevel Cut-Based Placement

The core placer we use is a multilevel cut-based placement tool, which is available on [15]. The placer works as follows. Initially, the placement region is represented as a block (referred to as bin) and the circuit is presented as a hypergraph. The placer recursively divides each bin, partitions its associated hypergraph and assigns the subhypergraphs to subbins, aiming at minimizing the total number (weight) of the nets incident to nodes in multiple partitions.

B. Power Model

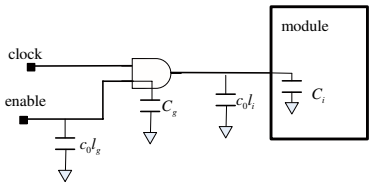

Fig. 2. Power model

For comparison, we use the same power model as [8][9] (it is also used in [5][6]), which includes the power dissipated by both the clock tree and the control signal, just as Fig.2 shows. Let c_0 be the unit wire capacitance, l_i, l_g be the interconnection length of the clock tree and the control signal, respectively. C_i and C_g denote the input capacitances for the register and the gate logic. Power dissipation is then modeled as:

$$[(c_0 l_i + C_i)p(i) + 0.5 * (c_0 l_g + C_g)p_{tr}]fV_{dd}^2 \qquad (1)$$

Where $p(i)$ represents the probability for the register to be active and p_{tr} is the probability of having a transition on the control signal net.

IV. Overview of Our Design flow

Our design flow implements gate planning based on a multilevel cut-based placer, as shown in Fig.3. When the physical adjacency information is briefly available after several partitions, we firstly evaluate the total activity for a cluster of registers in the same bin, and then estimate the power of these registers with a gate or not. If the former is the better, we will temporarily insert a gate among these registers and push the gate to this bin's cells link, then we add some pseudo edges between these registers and the gate to avoid placing this gate far away from these registers at next partitions. Since these registers may be partitioned into subbins at the next partition, while we do not hope the registers place far away from each other, especially for the registers with the similar activity patterns (The reason of it could refer to [8] and [9]). So before the next partition, we also add some pseudo edges between the registers in the same bin, the weight of the pseudo edges are determined according to the activity and transition analysis of the registers it connects [9]. At the next partition, if the registers at last level are partitioned into two subbins, we estimate the power by adding gates among these two child registers or not, then we choose the best case and dynamically insert and delete the gates, the detail of it will be presented in Section V. During the process of zero skew clock routing after placement, we re-evaluate the effectiveness of the inserted gates and delete the useless gates or treat some of them as buffers, Section VI will explain it in detail.

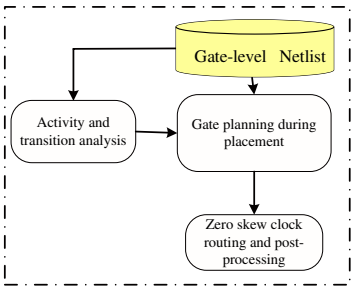

Fig. 3. Our design flow

V. Gate Planning during Placement

As pointed out in [8][9][10], clustering registers or pulling registers closely will deteriorate the traditional placement objectives and incur large overheads of signal nets wirelength and signal nets power, if these overheads are larger than the benefit of clock tree, it is not worth doing this. In order to reduce these overheads, our gate planning is after several partitions that the coarse placement result and the interconnect information are available. In addition, observing the power model in Eqn.1, for a register (or registers), only when the power saved by shutting off the clock signal is larger than the power introduced by the gate logic and the control signal, it is worth while to insert a gate, which is different from inserting a gate before each register in [5][6][8][9] at the beginning, and then followed by gate moving and deleting. So after several partitions, we evaluate the total activity by OR-ing the activity of the registers in the same bin, if it is active at all the cycles, inserting a gate is no doubt useless for reducing the power. Otherwise, we temporarily insert a gate to control these registers and put the gate into the bin's cells link for next level's partition and placement. In fact, these registers form a local clock tree and we add a gate before its root. At the next level's partition, if these registers are partitioned into two subbins, then we generate two child nodes, each child with respect to the registers in each subbin. Fig.4 shows an example, at a partition level, reg1∼reg6 are in the same bin and it is useful to add a gate among them to shut off the clock signal, at the next level, bin1 is partitioned into two subbins bin1_1 and bin1_2 with registers reg1, reg4, reg5 and reg2, reg3, reg6, respectively, then the local clock tree is as the right part of Fig.4 shows.

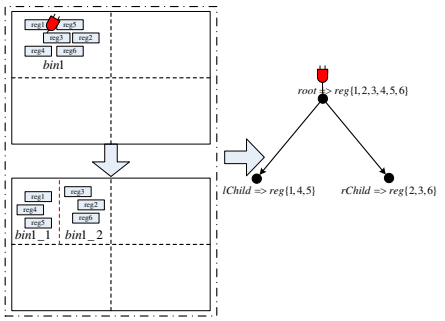

Fig. 4. An example of partition and its corresponding local clock tree

Here we should point out two important aspects:

1) As the partition continues in Capo [15], the cells in the same bin are treated as assembling at the center of the bin, there is no exact position for each cell. So it is difficult to estimate the exact wirelength of the local clock tree, and it will only be available until clock tree routing finishes. However, the power estimation is inaccurate for the tree with a gate or not if we neglect the wire capacitance, and in fact, wire capacitance accounts for much larger weight than the cell capacitance as the technology advances. To overcome the gap between this stage and routing, in this paper, we employ the metric introduced in [14] for wirelength estimation, which demonstrated that the total wirelength of the resultant zero skew clock tree could be estimated by $\sqrt{N}D$, where N and D are, respectively, the number of registers and the farthest distance between any pair of registers. Although this estimation correlates well with actual wirelength of the clock tree, it always overestimates and the farthest registers in the bin may not distribute at the corner of the bin. So we use the following equation to estimate the total capacitance of each cluster P with N registers. Where c_i is the load capacitance of register i, c_0 is the unit wire capacitance, and D is defined as above.

$$C(P) = \sum_{i=1}^{N} c_i + 0.7 * c_0 \sqrt{N} D \qquad (2)$$

2) As the partition continues, the information of the ad-

jacency of the registers will become more detailed, then it is more accurate to estimate the power for a cluster of registers with or without a gate. So we dynamically add gate before each child or delete the gate added at the parent. That means, if the parent is added a gate at last level, and the registers controlled by the gate are divided into two subbins, we estimate the power under the following cases as shown in Fig.5: i) preserve the gate added at parent (Fig.5(b)); ii) preserve the gate at parent and add a gate at each child (Fig.5(c)); iii) preserve the gate at parent and add a gate at one child (Fig.5(e) or Fig.5(f)); iv) delete the gate at parent and add a gate at each child (Fig.5(d)); v) delete the gate at parent and add a gate at one child (Fig.5(g) or Fig.5(h)); vi) delete the gate at parent and do not add a gate at any child (Fig.5(a)). Then we choose the best case and update the cells link of the corresponding bins for next level's partition and placement. In order to avoid placing the gate far away from these registers, we add a pseudo edge between each register and the gate, the weight of it should be relatively larger than the common net weight. As pointed out in [8][9], in order to divide the registers with the similar activity patterns into the same bin at the next level's partition, we also add pseudo edges between the registers of the local clock tree, the weights of them are determined by the partition level and the similarity of the registers' activity patterns as [8][9], the more similar of the activity patterns, the larger weight of the pseudo edge. Fig.6 shows an example, the width of the edge means the weight of the pseudo edge we add, the string of numbers beside the register denote its activity pattern. Based on the rules above, we add a gate among reg1∼reg6 since we could shut off the clock signal 4 cycles by gating. Before next partition, we add pseudo edges between the gate and each register with a relatively larger weight, to avoid placing the gate far away from these registers. The pseudo edge added between two registers is based on the activity pattern analysis, in this example, we assume that it is useful if there are at least 5 same idle cycles between two registers by gating technology.

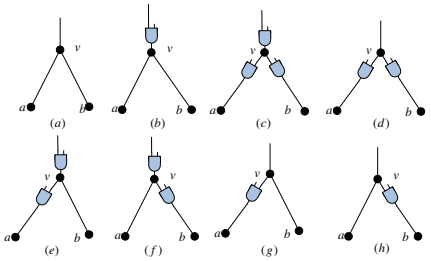

Fig. 5. Different gated clock tree topologies

VI. Gated Clock Tree Construction

A. Gate Removal at Bottom Level

When the placement is finished, we get the detailed positions of the registers and the gates, also including the local clock tree topology. For clock tree construction and routing, we need to get the exact positions of the internal

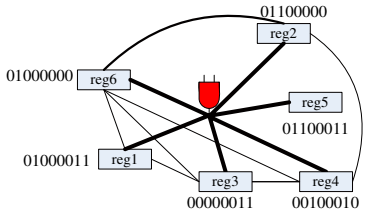

Fig. 6. An example of adding the pseudo edges

nodes and then we could interconnect the gate with the registers it controls. In order to minimize the clock skew, we use the deferred merge embedding (DME) algorithm to achieve zero skew clock routing as [9].

Since in Capo placer, some bin may contain more than one cell when the partition procedure (the CapoPlacer function in Capo [15]) terminates, the exact position of these cells is determined by next legalization (such as greedy movement and swapping, cell orientation optimization and row ironing). While our gate planning is integrated with partition, so at the bottom level of some local clock tree after gate planning, there may be more than one register in the same bin, and we will regard them have the same location, but in fact, their positions are different after legalization. So at the bottom-up node merging process of the DME algorithm, we must determine the internal node if there are more than one register at the bottom level of the local clock tree, and then re-evaluate the effectiveness of the gate (if exists) among them. If the gate cannot decrease the clock power, we delete it. We explain it by Fig.7. Reg1 and reg2 belong to the same bin when partition terminates, and we think adding a gate between them could decrease the clock power in gate planning. But the detailed placement of them are different after legalization. So we firstly determine the internal node of reg1 and reg2 satisfying zero skew and then evaluate the total power with the gate or not using Eqn.1.

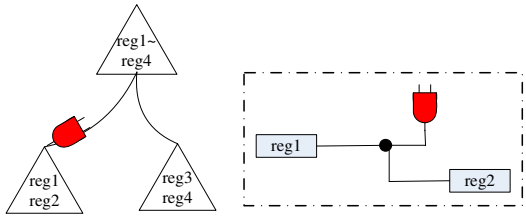

Fig. 7. Routing example for the bottom level

B. Gate to Buffer Transformation for Some Gates

It was pointed out in [4] that inserting too many gates may result in a large area and increase the complexity of the control circuit and the routing of the enable signals, at last it could even increase the total power, so [4] reduced the number of gates based on three cases and included a rule for enforcing a gate insertion when the subtree capacitance of the node reaches 20 times of the input capacitance of the gate. But in [4], these optimizations were taken after zero skew clock routing, while the internal nodes were determined on

the assumption that a gate is inserted before each node, so removing gate makes skew dissatisfied any more.

In our algorithm, after the gate planning during placement, we have gotten the local clock tree topology, which means whether adding a gate for the internal node or not is determined. After the DME algorithm, we determine the exact location of the internal node and then interconnect it with the gate according to the tree topology. Since adding a gate may not be effective after including the detailed interconnect capacitance, and sometimes it may even increase the total power. But if we delete these gates at this stage, it will deteriorate the clock skew or need re-construct the clock tree. In order to satisfy the clock skew and decrease the clock tree power, we treat these gates as common buffers and do not mask off the clock. Obviously, the control signal will be removed if a gate is regarded as a buffer. Using buffer and removing control signals contribute to further reduction of the total capacitance and power.

VII. Experimental Results

The proposed algorithm and the design flow have been implemented with C++ based on Capo 9.3 [15], and the experiments are performed on ISCAS89 benchmark circuits (since they contain the registers information which is needed in our experiment) with specified timing constraints and activity profile [9]. Other parameters used in our experiment are listed as follows: $r_0 = 0.207\Omega/um$, $c_0 = 0.015fF/um$, the gate's input capacitance C_g, output resistance r_g and intrinsic delay τ_g are $10.0fF$, 80Ω and $25fs$, respectively. Considering the high connection of the register with other cells, the input capacitance of the register is assumed relatively large, $C_{reg} = 100.0fF$. In order to show the effectiveness of our algorithm, we compare it with: (a) conventional placement (as the initial Capo) + gated clock tree construction in [9]; (b) the design flow introduced in [9]. The comparison results are shown in Table I.

First, we compare the results of clock tree. Since (a) does not especially care the registers during placement, so at last the registers loosely distribute on the chip. While in our gate planning during placement, we cluster the registers, and especially pull the registers with the similar activity pattern close to each other. That is why our algorithm reduces the clock tree wirelength by more than 22%. [9] constructs the clock tree after placement for all of the registers, that means it chooses a pair of nodes to merge at every iteration until there is only one node (the clock root) left. While our algorithm has formed many local clock tree topologies during gate planning, and in clock routing, we construct the clock tree according to the topology, so clock wirelength increases compared to [9], since our clock tree construction is somewhat only local optimization to reduce total wirelength. For clock tree power, we insert the gate logic only when it really decreases the power for a cluster of registers, and as the partition goes, we dynamically update (insert or delete gate) the gated clock tree topology configured at last level. At the post-processing after placement, we firstly delete the harmful gates after including the interconnect capacitance. Secondly, we treat some gates as general buffers and remove their control signal after gated clock tree construction. So the position of the gate logics gotten by our algorithm is more optimal and effective than the gated clock tree construction in [6] and [9]. That is why we achieve smaller clock tree power using much fewer gate logics.

Second, we compare the results of signal nets. The main difference between (a) and (b)/our method is that: the gated clock tree of (a) is constructed after placement, so the gate logics inserted during tree construction are not considered during placement. In other words, the position of the gate logics is ideal, in fact we can not place many of them in their optimal placement since they overlap with the already placed cells. (b) solves this problem by feeding the gate logics and gated clock network back to the previous design after gated clock tree construction, and then performs an incremental placement to remove these overlaps. So HPWL and signal nets power increase when comparing our algorithm and (b) with (a). We believe it is fair to compare our algorithm with (b) since both of them consider the gate logics in placement. Since (b) inserts more gate logics than our method, adding these gate logics to the previous design will inevitably move other cells positions during incremental placement, so it introduces large overhead of signal nets wirelength and power. While our design flow integrates gate insertion and previous placement in a seamless manner, we dynamically insert gate for some cluster of registers after several partitions, and then add it to the bin's cells link for the next level's partition, so the impact on the previous placement is minimal. As we expected, we reduce HPWL and signal nets power by 6.90% and 0.26% compared with [9]. HPWL reduction leaves more space for routing. Although the results of (a) are derived without considering the overlap removal of the gate logics with the other cells, it is still meaningful to compare the total power (including the clock tree power and signal nets power). Though our algorithm increases the signal nets power against (a), the power saved on the clock tree is much larger than the power increased on signal nets, so it is worth implementing gate planning during placement. While comparing our method with (b), we observe that it still reduces the total power by more than 13%, the main contributor of the power reduction is the clock tree power. That proves the method of gated clock tree construction in [6] and [9] will result in redundant gate logics, and at last some gate logics even increase the power inversely.

For run time, (b) needs two stages of placement: initial placement during which registers are clustered and pulled closely according to their activity patterns; incremental placement after adding the clock network and gate information to the previous netlist. During incremental placement, it still needs top-bottom level partitions to optimize HPWL. It really reduces HPWL comparing to some ECO method, but the increase of the run time cannot be ignored. Our algorithm integrates gate planning and top-bottom partition in a seamless manner, when the placement finishes, we also get the clock tree topology and the position of the inserted

TABLE I

COMPARISON RESULTS OF OUR ALGORITHM AGAINST PREVIOUS WORK (WL=WIRELENGTH, PW=POWER, THE UNITS FOR WIRELENGTH AND POWER ARE UM AND W, RESPECTIVELY. '+' MEANS INCREASE AND '-' MEANS DECREASE.)

Circuit	Method	Clock WL	Clock PW	#Gate	Delay(ns)	HPWL	Signal PW	Total PW	Run Time(sec)
s1488	(a)	25278.6	2.74E-4	6	0.261	1.81E6	0.003205	0.003479	18.26
	(b)	9168.63	1.94E-4	4	0.105	1.90E6	0.003296	0.003490	42.46
	our	11161.9	2.18E-4	2	0.114	1.86E6	0.003210	0.003428	21.48
	our Vs (a)	-55.84%	-20.44%	-66.67%	-56.32%	+2.76%	+0.16%	-1.47%	+17.63%
	our Vs (b)	+21.74%	+12.37%	-50.00%	+8.57%	-2.11%	-2.61%	-1.78%	-49.41%
s15850	(a)	2379250	0.014732	218	102.1	2.16E7	0.019801	0.034533	438.32
	(b)	1873020	0.012857	133	82.6	2.71E7	0.021783	0.034640	1055.27
	our	2026050	0.007619	90	57.0	2.52E7	0.021770	0.029389	492.92
	our Vs (a)	-14.85%	-48.28%	-58.72%	-44.17%	+16.67%	+9.94%	-14.90%	+12.46%
	our Vs (b)	+8.17%	-40.74%	-32.33%	-30.99%	-7.01%	-0.06%	-15.16%	-53.29%
s35932	(a)	6518700	0.018858	327	823.5	4.12E7	0.017321	0.036179	974.01
	(b)	5456660	0.017455	243	646.5	5.21E7	0.018977	0.036432	1914.44
	our	5740320	0.010544	140	623.2	4.71E7	0.019207	0.029751	913.38
	our Vs (a)	-11.94%	-44.09%	-57.19%	-24.32%	+14.32%	+10.89%	-17.77%	-6.22%
	our Vs (b)	+5.20%	-39.59%	-42.39%	-3.60%	-9.60%	+1.21%	-18.34%	-52.29%
s38417	(a)	6687290	0.067533	315	749.5	4.74E7	0.071576	0.139109	1004.35
	(b)	5449440	0.061749	229	627.2	5.90E7	0.078032	0.139781	2574.74
	our	5663830	0.037342	128	683.3	5.57E7	0.079739	0.117081	1220.87
	our Vs (a)	-15.30%	-44.71%	-59.37%	-8.83%	+17.51%	+11.40%	-15.84%	+21.56%
	our Vs (b)	+3.93%	-39.53%	-44.10%	+8.21%	-5.59%	+2.19%	-16.24%	-52.58%
s38584	(a)	6025670	0.031999	289	781.0	5.81E7	0.040371	0.072370	1229.65
	(b)	4994980	0.029136	225	638.3	7.26E7	0.045056	0.074192	2227.52
	our	5280720	0.017132	121	436.3	6.52E7	0.044132	0.061264	1122.63
	our Vs (a)	-12.36%	-46.46%	-58.13%	-44.14%	+12.22%	+9.32%	-15.35%	-8.70%
	our Vs (b)	+5.72%	-41.20%	-46.22%	-31.65%	-10.19%	-2.05%	-17.43%	-49.60%
Average	our Vs (a)	-22.06%	-40.80%	-60.02%	-35.56%	+12.70%	+8.34%	-13.07%	+7.35%
	our Vs (b)	+8.95%	-29.74%	-43.01%	-9.75%	-6.90%	-0.26%	-13.79%	-51.43%

gates. So the reduction of the run time is up to 50% on average comparing with (b). On the other hand, the increase for the run time of our algorithm is only within 8% over (a), which demonstrates the efficiency of our gate planning.

VIII. CONCLUSION

In this paper, we propose a new design flow for gated clock tree planning and construction. In order to decrease the impact on signal nets and the design closure of previous methods, we firstly evaluate the effectiveness of gate insertion for a cluster of registers, and then dynamically insert a gate for them if it could reduce the power. We achieve gate planning when placement finishes, and during clock routing, we firstly remove the useless gate to save area and power. After determining the exact locations of the internal nodes and the detailed interconnect information, we regard some gates as only common buffers and remove their connection of the control signal, to avoid re-constructing the clock tree and increasing the total power inversely. Experimental results show the effectiveness and efficiency of our method.

IX. ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (NSFC) 60776026.

REFERENCES

[1] Chunhong Chen, Changjun Kang, Majid Sarrafzadeh, "Activity-sensitive clock tree construction for low power", in Proc. ISLPED, pp. 279-282, 2002.

[2] A. Farrahi, C. Chen, A. Srivastava, G. Tellez, M. Sarrafzadeh, "Activity-driven clock design", IEEE Transactions on CAD/ICAS, Vol. 20, No. 6, pp. 705-714, June 2001.

[3] Jaewon Oh and Massoud Pedram, "Power reduction in microprocessor chips by gated clock routing", in Proc. ASP-DAC, pp. 313-318, 1998.

[4] Jaewon Oh and Massoud Pedram, "Gated clock routing for low-power microprocessor design", IEEE Trans. on CAD/ICAS, Vol. 20, No. 6, pp. 715-722, June 2001.

[5] Monica Donno, Alessandro Ivadldi, Luca Benini, Enrico Macii, "Clock-tree power optimization based on RTL clock-gating", in Proc. Design Automation Conf., pp. 622-627, 2003.

[6] Monica Donno, Enrico Macii, Luca Mazzoni, "Power-aware clock tree planning", in Proc. ISPD, pp. 138-147, 2004.

[7] Qi Wang, Sumit Roy, "Power minimization by clock root gating", in Proc. ASP-DAC, pp. 249-254, 2003.

[8] Weixiang Shen, Yici Cai, Xianlong Hong, Jiang Hu, "Activity-aware registers placement for low power gated clock tree construction", in Proc. ISVLSI, pp. 383-388, 2007.

[9] Weixiang Shen, Yici Cai, Xianlong Hong, Jiang Hu, "Activity and register placement aware gated clock network design", in Proc. ISPD, pp. 182-189, 2008.

[10] Yongseok Cheon, Pei-Hsin Ho, Andrew B. Kahng, et al, "Power-Aware Placement", in Proc. Design Automation Conf., pp. 795-800, 2005.

[11] Yongqiang Lu, C.N. Sze, Xianlong Hong, et al, "Navigating registers in placement for clock network minimization", in Proc. Desgin Automation Conf., pp. 176-181, 2005.

[12] Yanfeng Wang, Qiang Zhou, Xianlong Hong, Yici Cai, "Clock-tree aware placement based on dynamic clock-tree building", in Proc. ISCAS, pp. 2040-2043, 2007.

[13] Ting Hai Chao, Yu Chin Hsu, Jan Ming Ho, et al, "Zero skew routing with minimum wirelength", IEEE Transactions on Circuites & System II-Analog & Digital Signal Process, 39(11): 799-814, 1992.

[14] M. Edahiro, "A clustering-based optimization algorithm in zero skew routing", in Proc. Design Automation Conf., pp. 612-616, 1993.

[15] http://vlsicad.eecs.umich.edu/BK/PDtools/.