Challenges in Clockgating for a Low Power ASIC Methodology

David Garrett and Mircea Stan
University of Virginia
Department of Electrical Engineering
Charlottesville, VA 22903
{garrett,mircea}@virginia.edu

Alvar Dean
IBM Microelectronics Division
Essex Junction, VT
alvar@us.ibm.com

ABSTRACT

Gating the clock is an important technique used in low power design to disable unused modules of a circuit. Gating can save power by both preventing unnecessary activity in the logic modules as well as by eliminating power dissipation in the clock distribution network.There is an inherent pitfall though in implementing gating groups for hierarchical gated clock distribution because the groups are typically developed at the logic level with no information of the physical layout of the clocktree. Depending on the distribution of underlying sinks, maintaining gating groups can cause a wiring overhead that is potentially greater than the savings due to reduced switching. We look at modifications of zero-skew tree algorithms to consider both the physical and logical aspects of hierarchical gating. The algorithms are applied to data taken from a low power ASIC design. The best gated clocktree is created using both physical and logical information.

Keywords: clocktree, clockgating, low power, physical design

1 INTRODUCTION

The most fundamental control signal in a digital circuit is the clock signal. For a long time, the general theory behind clock design was that the signal should be kept as clean as possible, and that a circuit designer should not interrupt or disable a clock signal. More recently though it was realized that the clock signal is a major source of power dissipation, both in the signal itself, and in the unnecessary activity created in the underlying logic modules. Thus the push was made to gate the clock signal and disable portions of the clocktree distribution for reducing power dissipation[2][3][4][5]. A comprehensive review of clock distribution techniques can be found in [1].

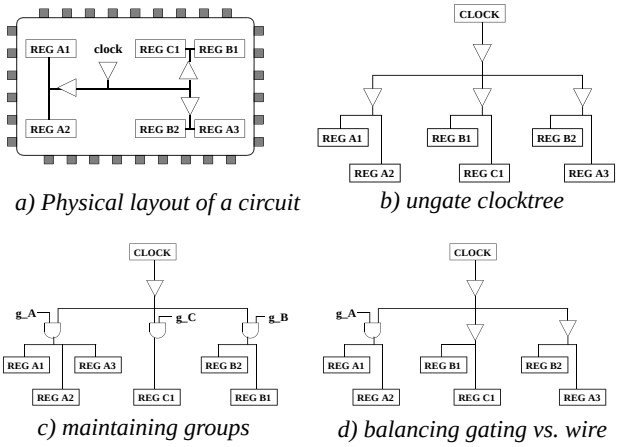
While gating the clock can provide enormous gains in the power efficiency of a design, the addition of clockgating complicates the clock signal distribution. Gated groups are developed from the logical model of the circuit, yet blindly applying gating at the logic level ignores the physical design and location of the group members. An effective low power gated clocktree tool must balance aspects of the logical and physical design in order to reach the best solution.

In this paper we look at the impact of the physical design on a hierarchical gated clocktree and its power dissipation. The gating is hierarchical because the clock sinks in the tree are gated individually, and we clockgate the higher levels of the tree to reduced power. Gated clock distribution in principle consumes less power due to the reduced switching, but depending on the physical placement and routing, the wiring can increase more then the savings from gating the clock. We present simple examples to show when hierarchical clockgating is effective within the clock distribution tree, and when it becomes prohibitively expensive to use. We modify the traditional zero-skew clock tree pairing algorithms to include consideration of the gating groups and to ultimately develop the best topology for the lowest power dissipation. These methods are applied to realistic placement, group and activity data generated from a low power ASIC microprocessor to show up to 24% reduction in switched capacitance for a clocktree in one design example.

2 CLOCKGATED DISTRIBUTION

Most of the published research in clockgating design does not address the influence of physical design on the effectiveness of gating the clock. When all of the members of the gating groups are tightly clustered, gating the clock will not substantially increase the wiring, but when one considers that gating groups may not be clustered in areas of a chip, the wiring overhead must be taken into consideration. One solution proposed in the literature is to use the gating activities to influence the floorplanning of physical design [3], but this may not always be feasible considering interconnect and critical timing constraints.

An example of the danger in ignoring the physical design is shown in Fig. 1, where Fig. 1a represents the physical placement of several groups of registers in a circuit design. Fig. 1b shows the most efficient, ungated clocktree that provides a balanced clock distribution. If gating is applied blindly, as in Fig. 1c, registers A1, A2, and A3 will be fed from the same signal, although A3 is located on the opposite end of the die. Fig. 1d shows a possible solution where A1 and A2 registers are gated, but the remaining registers are fed with an ungated clock because it provides the most power efficient method of



a) Physical layout of a circuit *b) ungate clocktree*

c) maintaining groups *d) balancing gating vs. wire*

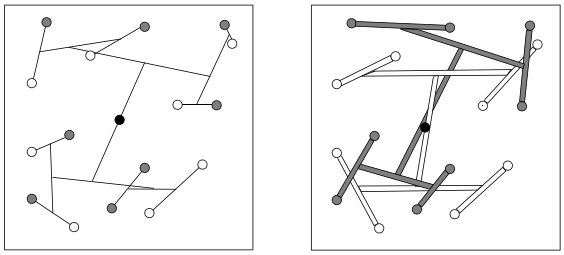
Fig. 1. a) shows the physical placement of several registers in a circuit. b) represents a clean, ungated clocktree. c) demostrates a possible reorganization of bottom level drivers to strictly maintain clockgating groups. d) shows a possible solution that balances both total wire while maintaining some gating.

distributing the clock. In an ASIC design methodology where the logic is synthesized and the cell placement is automated, multiple gating domains at the logic level can overlap in the same physical area making this a very real problem, and due to critical path constraints in the logic it is not always possible to relocate modules according to the logic gating structure.

Hierarchical gating in the clocktree creates new timing constraints that must be considered. One problem is that the gating signal must arrive in time to enable the clock in the upper stages of the tree, but in fact the gating signal control edge is most likely generated from registers on the lowest level in the clocktree. Thus the control signal has less than a full clock cycle in order to stabilize and provide gating signals for signals higher in the tree. A careful timing analysis must be performed on gating signals in order to guarantee timing constraints, and if a gating signal is along the critical path, it may not be feasible to gate the clock higher in the tree.

2.1 Clock routing algorithms

There are many algorithms available to generate zero-skew clocktrees for arbitrary placements in a plane. The original method of means and medians (MMM) by Jackson, Srinivasan, and Kuh [6] demonstrated a top-down approach by recursively partitioning the tree into groups until reaching the clock sinks. A very different approach was taken by Cong, Kahng, and Robins (KCR) who developed a bottom-up matching approach in building the balanced clock tree [7]. Their method matches all the sinks in the plane into pairs with a minimum of total wiring, and then uses the balanced skew points between pairs as the connection points for the next level of matching. A recent advance in the design of zero-skew trees is the Deferred-Merge-Embedding (DME) algorithm, a clocktree algorithm that starts with a bottom-up matching of clock sinks (using any of the developed matching heuristics), and then determines the exact zero-skew



n=16 m=1 *a) single tree*

n=16 m=2 *b) two clock groups*

Fig. 2. Two clock-routing strategies with clockgated groups. In a), the clocktree is routed with the same ungated signal to all sinks regardless of groups. In b), the sinks are partitioned into two independent groups and their gated clocksignals are routed independently but with an increase in total wiring.

point for each pair with a top-down approach [8].

There have been many significant improvements based on these algorithms for zero-skew tree design:

- Planar routing, where none of the routing wires cross, thus enabling a signal layer routing with no vias [9][10].
- Variable wire-size widths to control delay [11].
- Buffer insertion in the clocktree to minimize power [3][12][13].

2.2 Increase in clockgated wiring

One problem with gating as previously formulated is that it forces a top-down approach to the design at the logic level that does not take into account the physical design of the low-level sinks. Consider the case of a zero-skew tree in Fig. 2, where two clockgated domains are placed together (the shading in the circles denotes the two groups). Routing them in one large group as in Fig. 2a creates a minimum size tree, but the tree must use ungated clock signals. Fig. 2b shows that by enforcing independent trees for each group there is an increase in the total wiring, but the clock signals are gated. The essential problem with gated clocktree routing is to balance additional wiring with reduced signal activity.

In order to further illustrate the wiring overhead with gated groups, we consider a random distribution of several gated groups in a physical layout, which represents the case where clockgated groups are intermingled within the physical layout. The total wirelength to route a zero-skew tree for n randomly placed modules in a l_1 by l_2 area (Fig. 2a shows such an example tree) is $O(\sqrt{l_1 l_2}\sqrt{n})$ [7].

Theorem 1: Partitioning n modules with a uniform random distribution in a unit square into m independent groups will increase the average wiring for a binary tree by $\sqrt{m}$ in which each group is strictly maintained, as opposed to wiring n modules in a single binary tree.

Proof: Consider a unit routing area with n modules. The total wirelength will be proportional to $\sqrt{n}$. When the mod-

ules are partitioned into m independent groups, each subgroup will require a tree of wirelength $\sqrt{n/m}$. When the wirelength of all the subtrees and summed together, the wiring is $\sqrt{nm}$ for an increase of $\sqrt{m}$ over an ungated tree (1).

$$\frac{wiring(gated)}{wiring(ungated)} = \frac{\sum_{m} \sqrt{\frac{n}{m}}}{\sqrt{n}} = m\sqrt{\frac{1}{m}} = \sqrt{m} \quad (1)$$

Theorem 1 is only a rough estimate considering that is based on a order estimate. In addition, at some point the independent trees must all be connected, adding some additional wiring. The main point to realize is that as more gated groups are mixed in the same area of a circuit, more wiring overhead will be created in order to route those signals independently.

The ultimate goal of clockgating is to reduce the overall power dissipation in a circuit design, but just considering the increase in wiring capacitance does not provide the complete picture. The power dissipation of a gated wire is proportional to the *switched capacitance,* calculated as the product of total capacitance and the signal activity: $P \propto \sum \alpha C$.

2.3 Additional power overhead

The switched capacitance equation is a simplistic view of the power dissipation for a gated clocktree, because it is based purely on the wiring. In reality, clockgating within a tree creates other overhead components, such as the logic required to disable the clock signal for each group in the tree which leads to:

- the use of a logical gate to replace a buffer in the tree,
- the generation of a gating control signal,
- the routing of the control signals to each gated buffer.

In a buffered tree, the logic gates can replace the buffers with minimal impact on power dissipation, because most of the gate capacitance will be switched as well. When using hierarchical gating the lowest level registers already use enable signals, thus the gating signals are already available. The hierarchical gating signals on higher levels are then generated with a logical OR of the subgroup signals, which adds extra logic to the design.

A possible significant power overhead is the routing of the gating signals to the specific points in the clocktree. Our assumption for gating signals is that they are well-behaved, in the sense that in order to maintain a given activity α for the clock, the gating signal has a much lower frequency than the clock signal. This means that the gating signal gates the signal for relatively long periods, as opposed to gating and ungating the clock every other cycle.

The overhead becomes minimal with the well-behaved clockgate assumption, so we will make this assumption for the following simulations on a real clock routing problem from an ASIC core microprocessor.

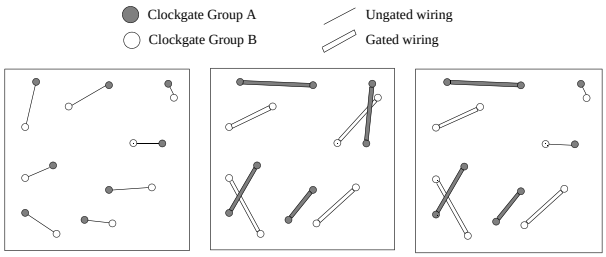


Fig. 3. Examples of gated pairing algorithms. a) NO GROUPS minimizes total wiring b) HARD GROUPS also minimizes total wiring, but pairings only allowed between similar clockgated nodes. c) SOFT GROUPS measures pairing cost as switched distance, total distance multiplied by the signal activity.

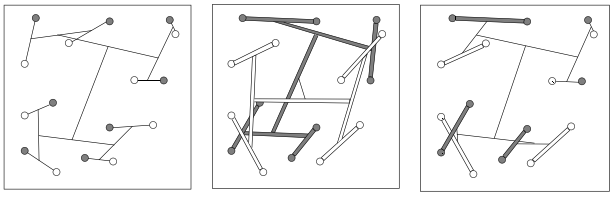
3 GATED CLOCKTREE ALGORITHMS

3.1 Methods of pairing

The examples in the previous sections have shown that gating groups can sometimes cause problems in the physical design of the tree, the key being in the distribution of the underlying nodes. Highly clustered members of the same gating group will not show the increase in wiring that completely random distributions have, thus algorithms to develop gating clocktrees must be modified to consider these factors. In order to explore the relationship further, the following modifications to the KCR greedy pairing algorithms were developed to balance gating groups with keeping a minimal length of wiring. The algorithms are illustrated in Fig. 3.

- *NO-GROUPS* (Fig. 3a) - This is the traditional matching algorithm, where the only determining factor is minimizing the overall wiring. If any two gated group members happen to be paired together, then they can be gated. This looks at purely physical placement.
- *HARD-GROUPS* (Fig. 3b) - This strictly enforces the matching among gating groups that are “alike”. For example with four clockgating groups, each of the four independent groups are paired using the *NO-GROUPS* algorithm. When no more group members can be paired, the logic for gating is placed at the head of each tree, and the grouping restriction is removed. This looks only at the logical gating information.
- *SOFT-GROUPS* (Fig. 3c) - This variant is similar to the *NO-GROUPS* algorithm but it tries to minimize *switched* capacitance. The cost function for pairing two nodes is not based purely on distance, but on the switched distance. This will penalize pairings between unlike members, and reward connections of the same groups, but only if they are physically close. This algorithm looks at both physical and logical information.

Each of the three pairing algorithms have strengths and weaknesses depending on the distribution of gating groups. If one considers clustering of gating group members (the



a) NO GROUPS *b) HARD GROUPS* *c) SOFT GROUPS*

Fig. 4. Examples of the resulting trees when the various pairings algorithms are used recursively on the balance points.

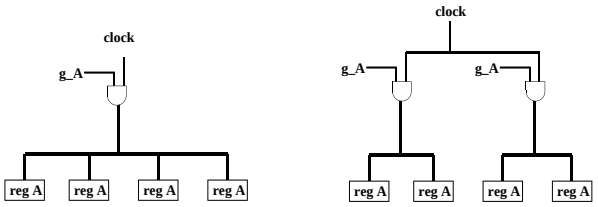
typically assumption in previous works for clockgating,) all of the pairing algorithms work well because the closest nodes for the minimum solution are already members of the same gating group. The other extreme of gating distribution is when the gating groups are randomly dispersed. *NO-GROUPS* finds the minimum wiring solution and essentially ignores gating possibilities unless two nodes just happen to be near. *HARD-GROUPS* on the other hand will ignore close nodes of different groups in order to pair nodes that potentially could be far apart. While *HARD-GROUPS* enables gating of nets, it may increase the wiring so much as to negate the power savings depending on the activity. The problem is that *HARD-GROUPS* ignores the switching, but it is the switching that determines power dissipation of the net. *SOFT-GROUPS* performs better in this situation as it maintains minimum wiring in the pairings, but it can pair nodes from the same group that are further apart if the switching can compensate for the wiring.

Fig. 4 shows examples of the trees that would be generated after recursively calling the algorithms on balance points between pairs. The *NO-GROUPS* and *HARD-GROUPS* trees are similar to their counterparts in the example in Fig. 2, and the selection between the two depends solely on the net switching activity. The SOFT-GROUPS tree provides a compromise between the two extremes of fully gated versus fully ungated trees, by gating on the first levels and then using an efficient ungated clock to feed the higher levels of the tree.

The *SOFT-GROUPS* pairing does have problems in that as soon as a group is paired with a node from a different group, no more gating can be performed at the higher level. Thus with SOFT-GROUPS, the pairing works well on the first level, but matching groups becomes harder on the next levels. SOFT-GROUPS can get trapped in a local minimum on the first couple of levels, and eliminates gating in the higher levels that could have resulted in further power savings.

3.2 UNGATE operation

A fourth heuristic was developed to exploit partial gating in the clocktree as with *SOFT-GROUPS*, but to also avoid the local minimum problem at the lower levels. The new algorithm, called *UNGATE*, performs a top-down analysis on the tree to decide whether to keep the gating groups. The difference is that *UNGATE* operates on a fully gated tree built with *HARD-GROUPS*, and then recursively traverses the tree to break apart the gating when desired. By making the gating



a) before UNGATE *b) after UNGATE*

Fig. 5. Example of the UNGATE operation where a grouped tree is partitioned into two independent groups fed by an ungated clocksignal.

the default, and forcing the algorithm to actively break gating sections, it tends to keep gating higher in the tree. Fig. 5 shows an example operation where a single gated tree can be broken into two separate gated trees, which then can be reconnected with an ungated signal.

The *UNGATE* algorithm ungates a node when it determines that a better low power solution would be to use an ungated clock. The basic idea is to look at a particular gated subgroup and estimate the wiring for both the gated and ungated version of the tree for all the surrounding nodes (not just members of that gating group). The estimate is calculated by recursively calling the *NO-GROUPS* and *HARD-GROUPS* algorithms described in the previous sections on the nodes in the surrounding area. After each gated group has been analyzed with *UNGATE*, the entire tree is rebuilt with the *NO-GROUPS* algorithm on the remaining nodes and the heads of the gating groups left by the *UNGATE* operation.:

```
function ungate ( GatedTree )
{
  nodes = find_enclosed_nodes( GatedTree )
  ungated_ power = treerouter (nodes,no_groups);
  gated_power = treerouter(nodes,hard_groups);
  // if ungated power is less, then break up the group
  if ( ungated_power < gated_power ) {
    ungate( GatedTree->left() )
    ungate( GatedTree->Right() )
  }
  // when the gated version prevails, add the entire
  // node to the list of remaining nodes to route
  else {
    NewClockSinks.Add ( GatedTree )
  }
```

Fig. 6. UNGATE Algorithm Pseudocode

This heuristic looks at the physical design of the circuit to make better decisions on gating nets. At the higher levels of the tree, chances are that many groups will be mixed together, and the tree will use an ungated clock signal. As the *UNGATE* algorithm proceeds down the tree, it ungates signals where it estimates the grouping is detrimental to power dissipation, and keeps gating in sections of the tree where many of the same nodes are clustered together. By working

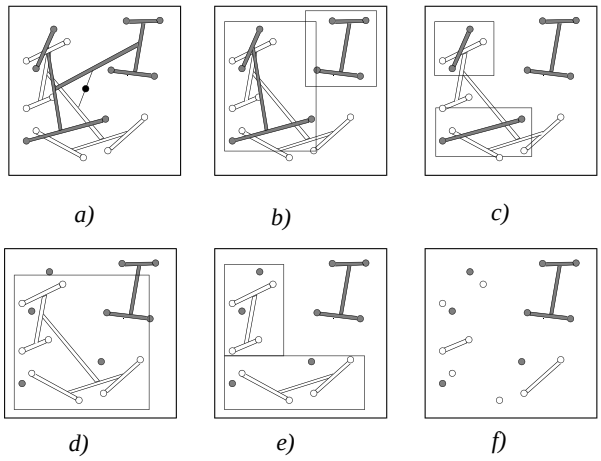


Fig. 7. An example of the UNGATE operation performed on a HARD GROUP tree. The algorithm encloses a group starting at the top and determines if the groups should be broken. When broken, the algorithm then examines the two underneath children for the same operation. a), b) and c) show steps to ungate the first group, and d), e), and f) show the steps for the second

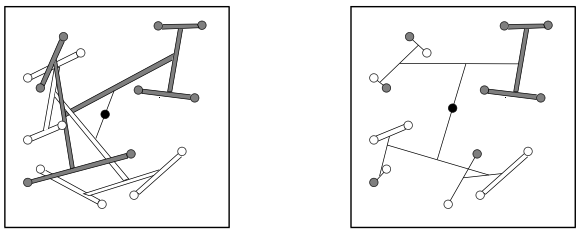


Fig. 8. Shows the resulting tree of the UNGATE operation. a) is the orginial HARD-GROUPS tree and b) shows the tree after it was UNGATING and rerouted with NO-GROUPS.

on a top-down approach, *UNGATE* is less likely to get trapped into a local minimum, but instead it starts with a fully gated tree, and determines where to disable the gating. Fig. 7 shows an example *UNGATE* operation on a gated tree from the *HARD-GROUPS* pairing example. After the ungate algorithm has completed, the entire tree is rebuilt using the *NO-GROUPS* pairing as seen in Fig. 8.

3.3 Results of Design Example

In order to evaluate the potential advantages of these algorithm changes, they were tested on a set of clock sink locations, gating groups, and activities generated from two versions of an ASIC core microprocessor design. The gating signals for the lowest clock sinks were taken from the register enable signals, and the hierarchical gating groups were selected based on a correlation threshold of similar signals. Thus the higher the threshold, the more individual groups there were, but each of the groups were smaller. For a lower correlation threshold, the groups were bigger, but the signals were less similar, thus reducing the effective signal activity

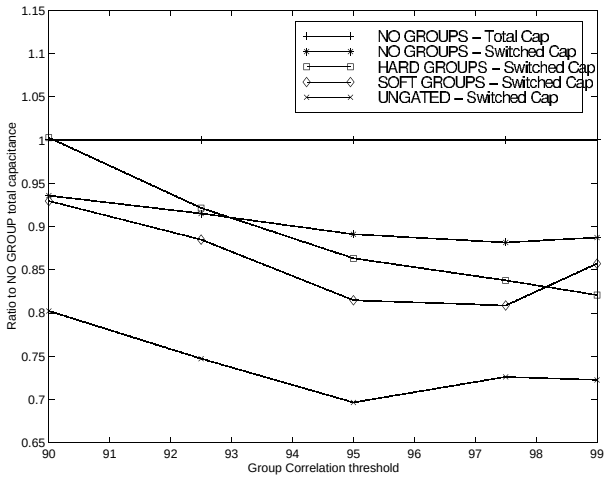


Fig. 9. Shows the ratio of switched wiring for each of the gated clocktree algorithms for Design #1.

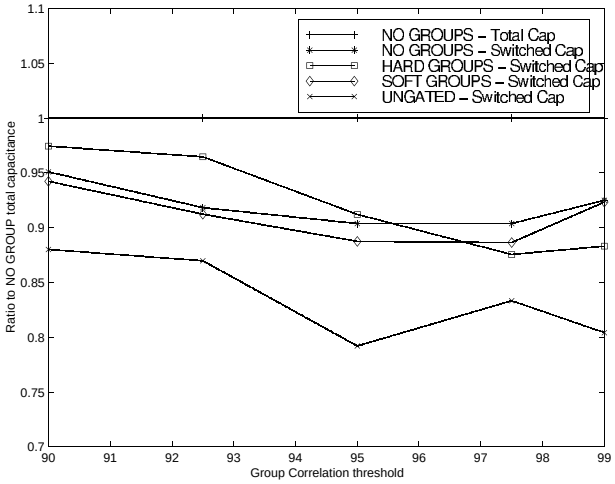


Fig. 10. Shows the ratio of switched wiring for each of the gated clocktree algorithms for Design #2.

(the gating signal must be the least common denominator between the members).

Our first three pairing heuristics were used with the KCR G+H+E algorithm to build a clocktree. The *NO-GROUPS* solution showed an average total wiring value of 12.15 and 7.61 units for the two design examples across all the different groups assignments. With *HARD-GROUPS* the average total wiring was almost 31% and 17% larger for the two design examples, but the switched capacitance was down by 11% and 8% respectively. The *SOFT-GROUPS* variant showed a slight increase in total wiring, but the switched capacitance was decreased by 14% and 9% respectively. By far the best gain was achieved with the *UNGATE* operation with 26% and 16% reduction in average switched capacitance relative to the ungated tree.. .

It is important to note the total wiring for the *UNGATE* with

	NO GROUPS		HARD GROUPS		SOFT GROUPS		UNGATE	
# GROUPS	total wire	switched wire	total wire	switched wire	total wire	switched wire	total wire	switched wire
25	12.19	10.81	15.37	10.00	12.58	10.44	12.70	8.80
19	12.11	10.68	15.68	10.14	12.50	9.79	12.66	8.79
13	12.18	10.85	16.07	10.51	12.32	9.92	12.84	8.48
12	12.18	11.14	16.20	11.22	12.32	10.78	11.56	9.10
10	12.08	11.30	15.98	12.11	12.23	11.22	11.15	9.69

Table 1.Gated clocktree routing algorithm wiring results for Design #1 (237 nodes)

	NO GROUPS		HARD GROUPS		SOFT GROUPS		UNGATE	
# GROUPS	total wire	switched wire	total wire	switched wire	total wire	switched wire	total wire	switched wire
16	7.61	7.04	8.92	6.72	7.76	7.03	7.64	6.12
11	7.64	6.90	8.95	6.69	7.66	6.77	7.87	6.37
9	7.64	6.90	8.97	6.97	7.62	6.78	7.57	6.05
7	7.62	6.99	9.11	7.35	7.64	6.95	7.91	6.62
4	7.53	7.16	8.37	7.33	7.48	7.09	7.17	6.62

Table 2.Gated clocktree routing algorithm wiring results for Design #2 (146 nodes)

10 and 12 groups on the first design example and 4 groups for the second example because in these cases, the total wiring was actually less then the *NO-GROUPS* alternative. The explanation for this is that all heuristics are non-optimal and can get trapped into local minima. Even considering using those results as the ungated baseline, the best *UNGATE* solutions of 8.48 units and 6.05 units are still 24% and 16% lower for the two design examples respectively.

4 CONCLUSIONS

The clock tree is a good target for power reduction in processor designs because it switches all the time, consuming power every cycle. Gating the clock is an important technique that can shutdown portions of the network and prevent power dissipation, but it must be carefully applied and it must consider the physical design aspects. The all or nothing approach to gating may lead to problems, while the optimal solution seems to point to a partial approach, one that balances efficient wiring with the power savings due to gating. The clock trees presented here demonstrate some the basic principles of selectively gating higher in the clocktree. The *UNGATE* operation gives an algorithmic solution to combining information for the logic and physical design aspects to build an effective gated clocktree.

ACKNOWLEDGEMENTS

This work is supported by a research grant from IBM Microelectronics Division, Burlington, Vermont and through NSF Career grant MIP-9703440. The authors would like to thank Dr. Gabriel Robins for help in understanding the balanced-wiring tree algorithms, and Mark Weir and John Chickanosky for their review of the paper.

5 REFERENCES

[1] E. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems*, IEEE Press: New Jersey. 1995.

[2] J. Oh, M. Pedram, "Gated Clock Routing Minimizing the Switched Capacitance," *Proc. of Design and Test in Europe*, Feb 1998, pp 692-697.

[3] M. Sarrfzadeh, A. Farrahi, G. Tellez, "Activity-Driven Clock Design," *International Conference on Computer-Aided Design* 1995.

[4] Q. Wu, M. Pedram, and X. Wu, "Clock-Gating and Its Application to Low Power Design of Sequential Circuits," *IEEE Custom Integrated Circuits Conference*, 1997, pp. 479-482.

[5] G. Yeap, "Low-Power Design Methodology," *Kluwer Academic Publishers*, 1998, p. 140-141.

[6] M. Jackson, A. Srinivasan, and E. Kuh, "Clock Routing for high-performance IC's," in *Proceeding 27th Design Automation Conference*, 1990, pp. 573-579.

[7] J. Cong, A. Kahng, G. Robins, "Matching-Based Methods for High-Performance Clock Routing", *IEEE Transactions on Computer Aided Design*, vol. 12., no. 8, Aug 1993, pp. 1157-1169.

[8] T. Chao et al, "Zero-skew clock routing trees with minimum wirelength," *IEEE Transactions on Circuits and Systems*, Nov. 1992, p. 799-814.

[9] Q. Zhu, W. Dai, "Planar Clock Routing for High Performance Chip and Package Co-Design," *IEEE Transactions on VLSI Systems*, vol. 4, no. 2, June 1996, pp. 210-226.

[10] A. Kahng, C. Tsao, "Planar-DME: A Single-Layer Zero-Skew Clock Tree Router," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 15, no. 1, January 1996, pp. 8-19.

[11] R. Tsay, "Delay Minimization for Zero-Skew Routing." *IEEE Transactions on Computer-Aided Design*, February 1993.

[12] A. Vittal, M. Marek-Sadowska, "Low-Power Buffered Clock Tree Design," *IEEE Transactions on Computer-Aided Design on Integrated Circuits and Systems*, vol. 16, no. 9, Sept 1997, pp. 965-975.

[13] K. Carrig, et al, "A New Direction in ASIC High-Performance Clock Methodology," *IEEE Custom Integrated Circuits Custom 1998*, p. 593-596.