Gated Clock Routing Minimizing the Switched Capacitance*

Jaewon Oh and Massoud Pedram

Dept. of Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089
Tel: (213) 740-4480
e-mail: [joh, massoud]@zugros.usc.edu

Abstract

This paper presents a zero-skew gated clock routing technique for VLSI circuits. The gated clock tree has masking gates at the internal nodes of the clock tree, which are selectively turned on and off by the gate control signals during the active and idle times of the circuit modules to reduce switched capacitance of the clock tree. This work extends the work of [4] so as to account for the switched capacitance and the area of the gate control signal routing. Various tradeoffs between power and area for different design options and module activities are discussed and detailed experimental results are presented

1. Introduction

Clock gating is an effective way of reducing power dissipation in digital circuits. In a typical synchronous circuit, e.g. a general purpose microprocessor, only a portion of the circuit is active at any given time. By shutting down the idle modules in the circuit, we can prevent the circuit consuming unnecessary power. In addition, we can shut down a portion of the clock tree by masking off the clock at the internal node of the tree using an AND-gate. This prevents unnecessary switching in the clock tree and saves power in the clock tree in addition to the power savings in the modules.

In this paper, we address an instance of the *gated clock routing* problem. In our gated clock tree, we insert gates immediately after every internal node of the clock tree to minimize the dynamic power consumption. These gates also serve as buffers and can be sized to adjust the phase delay of the clock signal. They are turned on and off by the control signals generated from a centralized gate controller. An instance of the gated clock tree is shown in Figure 1, where the sinks correspond to the locations of modules and the Steiners are the internal nodes of the clock tree.

A gate in the clock tree must be enabled (i.e. the control signal is true) whenever any of its descendant gates are enabled. This suggests that the control signal of a gate is the OR function of the control signals of its descendant gates.

* This research was funded in part by DARPA under contract no. F33615-95-C1627 and by NSF under contract no. MIP-9628999.

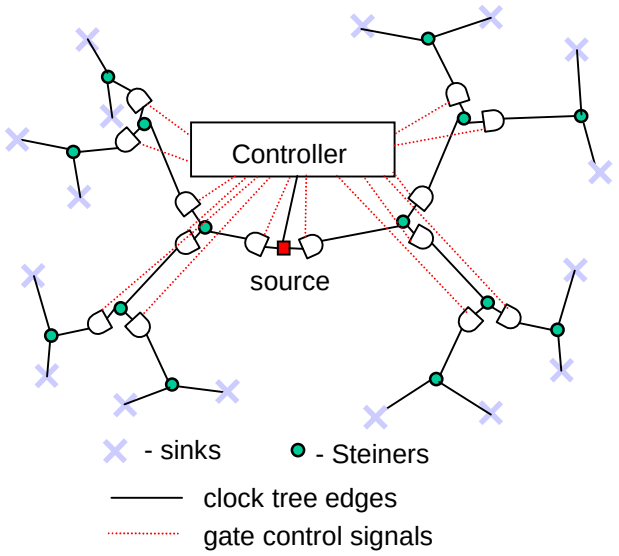

Figure 1: Gated clock tree

In [5], a gated clock tree topology construction based on module activity patterns was suggested. The authors used high-level synthesis information to extract the activity patterns. However, the routing of the clock tree and the control signals, the actual power dissipation and the area of them were not considered. In contrast, our method considers all of these. In addition, we propose a method for clock tree construction based on the instruction statistics of the processor. This can be extracted from instruction level simulation of the processor with a number of benchmark programs. The instruction statistics are used to extract the activities of the nodes and the switching activities of the control signals as will be discussed in the following sections. More precisely, we will investigate how the probabilistic information (instruction statistics) and the geometrical information (sink locations) are used to guide the low power clock routing.

The remainder of this paper is organized as follows. Section 2 gives the terminology and the precise problem statement. Section 3 describes how the probabilities of the gate control signals are calculated. Section 4 presents the clock tree construction algorithm. Sections 5 and 6 show our experimental results and conclusions.

2. Problem Definition

We assume that the topology of the clock tree is full binary, that is, every non-leaf node has exactly two children. However, the tree is not necessarily a balanced tree (depth of leaf nodes may not be the same). Let T be the rooted clock tree topology. Let $\{M_1, M_2,\ldots,M_N\}$ be the modules. If there are N modules, there are N–1 internal nodes. Let $\{v_1, v_2,\ldots,v_{2N-1}\}$ be the nodes of the clock tree where $\{v_1, v_2,\ldots,v_N\}$ are leaves and the rest are internal nodes of the tree. Let $\{e_1, e_2,\ldots,e_{2N-2}\}$ be the edges of the tree. We identify each point v_i, except the root, of the rooted topology T with edge e_i, so e_i connects v_i to its parent in T. Let $|e_i|$ be the length of edge e_i.

We assume that the controller is located at the center of the chip. The control signal routing is a *star routing* as shown in Figure 1. We denote the controller tree as S. We label each edge in the controller tree as EN_i. Edge EN_i controls the gate on edge e_i of the clock tree. The signal probability of EN_i (probability that EN_i is 1) is denoted as $P(EN_i)$ and the transition probability of EN_i (probability that EN_i changes logic value per cycle) is denoted as $P_{tr}(EN_i)$. Let $|EN_i|$ be the length of edge EN_i.

2.1 Switched Capacitance of the clock tree

Consider a clock tree without gates. For a particular edge e_i, the power dissipation on the edge e_i is given by

$$power(e_i) = \frac{1}{2} c_0 |e_i| \alpha f V_{dd}^2 \qquad (1)$$

where c_0, α, f, V_{dd} are the unit wire capacitance, transition probability of the clock net, the clock frequency, and the supply voltage, respectively. For the clock net, $\alpha = 2$ since there is one rising and one falling edge in every clock cycle. So the above equation becomes

$$power(e_i) = c_0 |e_i| f V_{dd}^2$$

With the masking gates, this power is dissipated only when the control signal is on. Thus the power dissipation in edge e_i is

$$power(e_i) = c_0 |e_i| P(EN_i) f V_{dd}^2$$

During the layout synthesis step, V_{dd} and f are fixed parameters, hence we can use the switched capacitance as an exact measure of the power dissipation. The switched capacitance $w(e_i)$ of an edge e_i is given by

$$w(e_i) = c_0 |e_i| P(EN_i).$$

There may be a load capacitance associated with each node of the clock tree. Including the node capacitance C_i at node v_i, the switched capacitance of e_i is given by

$$w(e_i) = (c_0 |e_i| + C_i) P(EN_i).$$

The total switched capacitance in the clock tree is therefore

$$W(T) = \sum_{\forall e_i} (c_0 |e_i| + C_i) P(EN_i).$$

2.2 Switched capacitance in the controller tree

Similarly, from Equation (1), we can see the switched capacitance of edge EN_i is

$$w(EN_i) = \frac{1}{2} (c_0 |EN_i| + C_g) P_{tr}(EN_i)$$

where C_g is the input capacitance of the AND gate. The total switched capacitance in the controller tree is

$$W(S) = \frac{1}{2} \sum_{\forall EN_i} (c_0 |EN_i| + C_g) P_{tr}(EN_i)$$

The objective of our gated clock routing is to find trees T and S so as to minimize

$$W = W(T) + W(S)$$

subject to *zero skew* constraints. Notice that the signal probability of EN_i determines the switched capacitance in the clock tree whereas its transition probability determines the switched capacitance in the controller tree.

3. Computation of $P(EN_i)$ and $P_{tr}(EN_i)$

To calculate $W(T)$ and $W(S)$, we need to compute the signal probabilities $P(EN_i)$ and the transition probabilities $P_{tr}(EN_i)$. Let $P(M_i)$ be the probability that M_i is active (i.e. M_i receives the clock signal). Suppose v_i has modules $M_1, M_2,\ldots,M_l$ at the leaves. If any of these modules are active, then EN_i must be turned on. Thus $P(EN_i)$ is given by

$$P(EN_i) = P(M_1 \cup M_2 \cup \cdots \cup M_l) \qquad (2)$$

To find $P_{tr}(EN_i)$, we need the module activation statistic over consecutive clock cycles. Let $AT(M_i)$ be a two-bit *activation tag* which represents the module activities in two consecutive clock cycles. For example, $AT(M_i) = 01$ means that M_i is idle in the current clock cycle and becomes active in the next clock cycle. $AT(M_i)$ can have four possible values and their corresponding logic value transitions of EN_i are shown below.

1. $AT(M_i) = 00$ (EN_i stays at 0)
2. $AT(M_i) = 01$ (EN_i makes a 0 to 1 transition)
3. $AT(M_i) = 10$ (EN_i makes a 1 to 0 transition)
4. $AT(M_i) = 11$ (EN_i stays at 1)

Cases 1 and 4 do not cause transition of EN_i, so we just need to consider cases 2 and 3 for the computation of the transition probability.

If Register Transfer Level (RTL) simulation is used to find these probabilities, a huge number of clock-by-clock module usages have to be recorded. Certainly, the time complexity will be very large, especially for general-purpose microprocessors. So we propose a method for computing activities using more efficient *instruction level simulation* of the processor and knowledge about the RTL description of the processor.

3.1 RTL description

For simplicity, we assume that the microprocessor has four instructions and six modules throughout the rest of this section. The RTL description of each instruction tells us what modules are used to execute each instruction. For example, we may have the following RTL description of the instructions.

Instruction	Used Modules
I_1	M_1, M_2, M_3, M_5
I_2	M_1, M_4
I_3	M_2, M_5, M_6
I_4	M_3, M_4

Table 1: RTL description of instructions

3.2 Instruction stream

By simulating the processor at the instruction level with a number of benchmark programs, we can trace the instructions that are executed. For example, our instruction stream for 20 clock cycles may be given as follows.

$I_1\ I_2\ I_4\ I_1\ I_3\ I_2\ I_2\ I_1\ I_2\ I_1\ I_3\ I_2\ I_1\ I_3\ I_1\ I_2\ I_1\ I_1\ I_4\ I_2$

From this, we can find any probabilities by scanning the instruction stream. For example, M_1 appears in I_1 and I_2, and these two instructions occur 15 times in the stream, so $P(M_1) = 15/20 = 0.75$. If a node v_i in the clock tree has two leaf nodes M_5 and M_6, any instructions that use either of these modules should contribute to the signal probability of EN_i. I_1 and I_3 are such instructions, so $P(EN_i) = P(M_5 \cup M_6) = 11/20 = 0.55$. The transition probability of this EN_i is also found by scanning the instruction stream. We examine every two consecutive instructions during the scanning and count the number of transitions of EN_i (01 transition when both M_5 and M_6 are idle in the current clock cycle and any of M_5 and M_6 are active in the next clock cycle; 10 transition is the reverse of this). We can see that $P_{tr}(EN_i) = 10/19 = 0.526$ (there are 19 transitions).

However, the instruction stream can be very long. To get accurate instruction statistics, we may need some millions of instructions. Because some instructions are rarely executed, the instruction stream should be very long to get reasonable probability value for the rare instructions. Therefore, the above brute-force method is very expensive.

To overcome this problem, we propose a method that computes all the necessary probabilities from the tables that can be generated by scanning the instruction stream just once.

3.3 Table-driven probability computation

Instruction Frequency Table (IFT) enlists the probability that each instruction is executed on the average. By scanning the previous instruction stream, we have the IFT in Table 2.

Instruction	**I_1**	**I_2**	**I_3**	**I_4**
Probability	0.4	0.35	0.15	0.1

Table 2: Instruction Frequency Table

To find $P(M_5 \cup M_6)$, we simply add the two probabilities of I_1 and I_3 in Table 2, which is 0.55. Any signal probability $P(EN_i)$ can be found using Table 1 and Table 2 without rescanning the instruction stream. It was shown in [4] that the time complexity of computing this probability is $O(KL)$, where K is the total number of instructions and L is the maximum number of used modules for any instructions ($K = L = 4$ in our example).

Instruction Transition-Module Activation Table (IMATT) enlists $AT(M_i)$ for every possible combination of two consecutive instructions. In addition, IMATT keeps the probability that the two instructions occur in two consecutive clock cycles. By scanning the previous instruction stream, we have IMATT in Table 3.

Prob.	Instr. Trns.	M_1	M_2	M_3	M_4	M_5	M_6
0.057	$I_1 \rightarrow I_1$	11	11	11	00	11	00
0.158	$I_1 \rightarrow I_2$	11	10	10	01	10	00
0.158	$I_1 \rightarrow I_3$	10	11	10	00	11	01
0.057	$I_1 \rightarrow I_4$	10	10	11	01	10	00
0.184	$I_2 \rightarrow I_1$	11	01	01	10	01	00
0.057	$I_2 \rightarrow I_2$	11	00	00	11	00	00
….	…	…	…	…	…	…	…
0.000	$I_4 \rightarrow I_4$	00	00	11	11	00	00

Table 3: Instruction Transition-Module Activation Table

For example, $I_1 \rightarrow I_3$ occurs three times in the instruction stream. So its probability is 3/19 = 0.158.

Assume that v_i has leaf nodes $M_1, M_2, \ldots, M_l$. In order for EN_i to make a 01 transition, at least one of $AT(M_k)$, $k=1,\ldots,l$ should be 01 while the remaining $AT(M_k)$s should be either 00 or 01. Likewise, in order for EN_i to make a 10 transition, at least one of $AT(M_k)$, $k=1,\ldots,l$ should be 10 while the remaining $AT(M_k)$s should be either 00 or 10. All other cases force EN_i to remain at 0 or 1. Alternatively, we perform logical OR over all the $AT(M_k)$s and if its result is 10 or 01, we have transitions on EN_i. For example, if v_i has leaf nodes M_2, M_3, M_5 and M_6, $I_1 \rightarrow I_2$ causes EN_i to make a 10 transition since the OR of the four corresponding table entries is 10. For each row in Table 3, if the corresponding modules' activation tags cause EN_i to

make a transition, the probability on that row should be added to the transition probability of EN_i. The time complexity of computing the transition probability of *ENi* is $O(K^2N)$, where K is the total number of instructions and N is the total number of modules.

4. Clock Tree Construction

4.1 Delay modeling

To estimate the phase delay of the clock tree, we use the Elmore delay model as was used in [6] for zero-skew clock routing. Inserting gates reduces the subtree capacitance in the Elmore delay computation, thereby reducing the phase delay.

4.2 Minimum switched capacitance heuristic

Bottom-up merging followed by top-down placement method is commonly used in clock routing. In [2], the merging sector is a line segment with slope ±1, which represents the possible locations of the Steiner node where its two subtrees are merged. These merging sectors are found in bottom-up fashion. The actual locations of the Steiner nodes are determined in top-down fashion (see an example in Figure 2). The nearest-neighbor heuristic of [3] greedily merges two nodes when the geometric distance between the corresponding merging sectors is minimum. Our method is also greedy, but the merging sequence is determined by the switched capacitance.

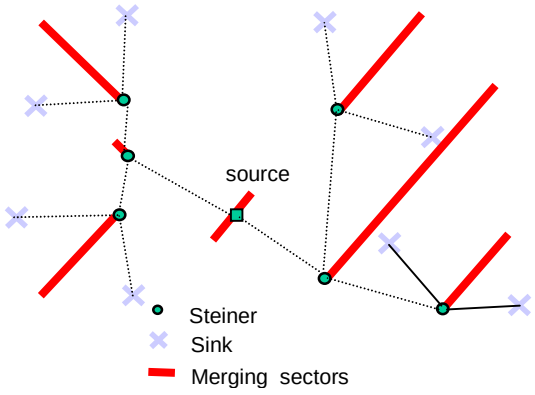

Figure 2: An example of bottom-up merging sequence

Let $ms(v_i)$ be the merging sector of v_i. Suppose we try to merge ($ms(v_i)$, $ms(v_j)$) and the root of the merged tree is v_k. We can uniquely determine $|e_i|$, $|e_j|$ such that the zero skew constraint is satisfied. As mentioned before, we assume that the gate controller is located at the center of the chip. Let this center be CP. To compute the switched capacitance in an edge of the controller tree, we need to estimate the distance between the gate location (location of the Steiner node) and the *CP*. Since we do not know the exact locations of the Steiner nodes during the bottom-up phase, we approximate the edge length of the controller tree as the distance between the *CP* and the middle point of the merging sector. Let this distance be $dist(CP, mid(ms(v_j)))$. Then the switched capacitance SC after the merge of ($ms(v_i)$, $ms(v_j)$) is

$$SC(v_i, v_j) = (c_0|e_i| + C_i)P(EN_i) + (c_0|e_j| + C_j)P(EN_j) + \frac{1}{2}(c_0\, dist(CP, mid(ms(v_i)) + C_g)P_{tr}(EN_i) + \frac{1}{2}(c_0\, dist(CP, mid(ms(v_j)) + C_g)P_{tr}(EN_j) \quad (3)$$

When we merge subtrees bottom-up, we merge sectors that result in the smallest switched capacitance as given in Equation (3). Detailed algorithm for the clock tree construction is similar to [4]. We summarize the algorithm outline below (SC is short for switched capacitance).

PROCEDURE GatedClockRouting
Input: Instruction Stream,
RTL description of each instruction,
Sink locations;
Output: Clock Tree Layout with gates

begin
scan the instruction stream and create IFT and ITMAT;
find $P(EN_i)$ and $P_{tr}(EN_i)$ for every sink;
compute SC between every pair of sinks;

// bottom-up merge
repeat
pick the pair $ms(v_x)$, $ms(v_y)$ whose SC is minimum
create new node v_k;
compute $P(EN_k)$ and $P_{tr}(EN_k)$;
find $ms(v_k)$;
remove node v_x, v_y;
for each remaining node v_n
determine $|e_k|$, $|e_n|$ satisfying zero-skew;
compute SC between v_k, v_n;
end for
until only the root is left

// top-down placement
place internal nodes v_k within each $ms(v_k)$;
end PROCEDURE

Scanning the instruction stream and the creating the tables take $O(B)$, where B is the length of the stream. The second and the third statements take $O(N\ (KL + K^2N))$ and $O(N^2)$ respectively. Since $L < N$, the combined complexity is $O(K^2N^2)$. The **repeat** loop iterates N times and within each iteration, the dominating complexity is the probability computation which takes $O(K^2N)$. So the overall complexity is $O(B + K^2N^2)$.

4.3 Reduction of Gates

Inserting gates at every node of the clock tree may result in large area and increase complexity of the control circuit and

the routing of the enable signals. Especially, since the routing of enable signals is a star routing, its area can be bigger than the clock tree routing if we do not reduce the number of gates. There are cases when inserting gates hardly reduces switched capacitance. We can think of three cases when a node does not need a gate.

1. Activity of the node is close to 1,
2. Switched capacitance of the node is very small,
3. Activity of the parent node is almost the same as activity of the node.

Case (1) is obvious since there is no time frame during which the node can be shut off. In case (2), the node's switched capacitance is so small that having a gate can only reduce switched capacitance marginally. In case (3), there is very little increase in activity when we go up from the node to its parent. In this case, it is not necessary for both the node and its parent to have gates. Only the parent will have a gate, and the resulting switched capacitance is at most slightly higher than the case that both nodes have gates.

However, these gate removal schemes may remove so many gates in the tree that the phase delay of the clock signal may increase rapidly. So we included a rule for enforcing a gate insertion regardless of those three schemes whenever the subtree capacitance of the node reaches, say $20C_g$, where C_g is the input capacitance of a gate.

5. Experimental Results

We implemented our algorithm in C++ on a Sun Sparc 20 workstations. For sink locations (module locations) and the sink load capacitance, we used the benchmarks r1-r5 from [6]. The instruction stream and the used modules for each instruction are generated according to a probabilistic model of the CPU when it executes typical programs. The benchmark characteristics are shown in Table 4. The length of the instruction stream was 100 thousands for all the benchmarks. The average number of used modules per instruction is about 40% for all the benchmarks (this can be seen in the column labeled Ave($M(I_i)$)). That is, about 40% of the modules are active at any given time on the average.

Bench	No. of sinks	No. of instr	Ave($M(I_i)$)
r1	267	64	107
r2	598	89	240
r3	862	108	345
r4	1093	120	438
r5	3101	160	1240

Table 4: Benchmark characteristics for gated clock routing

Note that the power consumption of the gated clock tree will be at least 40% of the ungated clock tree as a result.

5.1 Buffered clock tree vs. gated clock tree

Buffered clock tree is a commonly used method in current clock routing. The buffered clock tree is constructed using the nearest neighbor heuristic and the size of a buffer is assumed to be half the size of AND-gates. The comparison among buffered clock tree, gated clock tree and gated clock tree with gate reduction heuristic is shown in Figure 3.

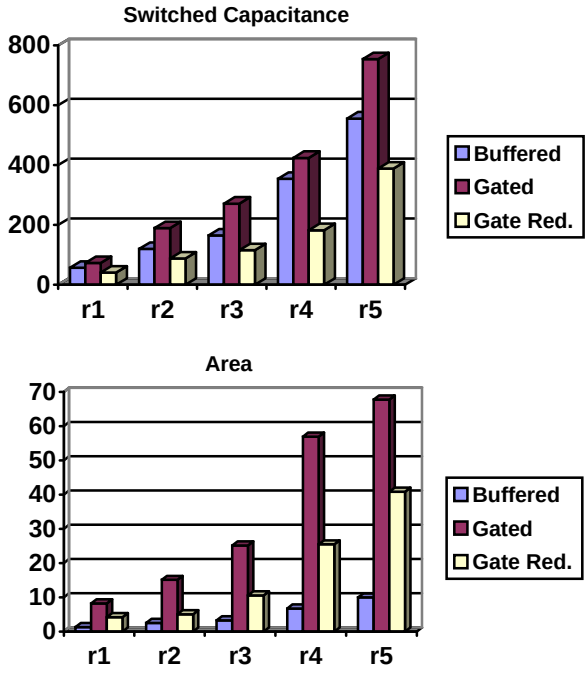

Figure 3: Comparison among different clock routing methods. Switched capacitance in pF, area in $10^6 \lambda$

As can be seen from the figure, if the gate reduction heuristic is not applied, the gated clock routing is worse than the conventional buffered clock routing. The major overhead in switched capacitance and the area comes from the star routing. After the gate reduction, it consumes about 30% less power than the buffered clock routing. There is still however an area overhead.

5.2 Impact of average module activity

If the average activity is too high, then there is little room for power savings. The average module activity vs. switched capacitance is shown in Figure 4. As the average module activity increases, the power consumption difference between the two routing methods diminishes. Thus the gated clock routing is more effective when the module activity is low.

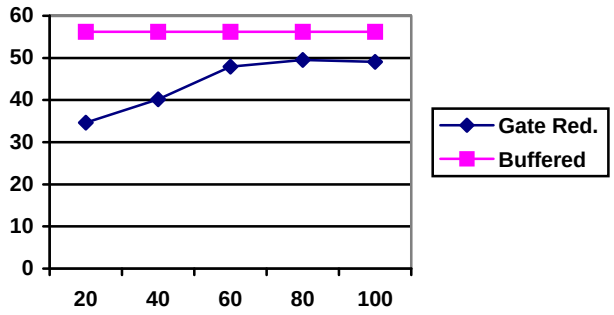

Figure 4: Average module activity (x-axis) vs. switched capacitance (y-axis) for benchmark r1

5.3 Optimum number of gates

If there are a lot of gates, then the switched capacitance in the clock tree is reduced, but the switched capacitance and the area of the controller tree is increased. On the contrary, if there are too few gates, the switched capacitance in the clock tree will be increased. Intuitively, there will be an optimum number of gates that minimizes the total switched capacitance. This is shown in Figure 5.

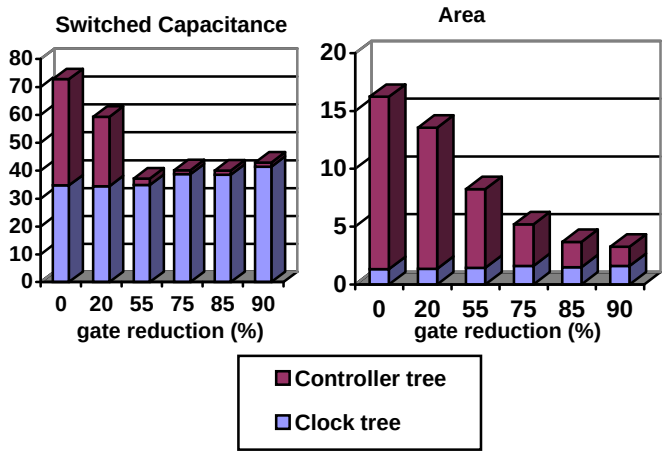

Figure 5: Gate reduction vs. switched capacitance and area for benchmark r1

We controlled the number of gates by giving different parameters in the gate reduction heuristic. When there are many gates, the controller tree dominates the switched capacitance and the area. As the number of gates is reduced, the switched capacitance in the controller tree is reduced but that of the clock tree is increased. In the figure, the optimum gate reduction for lowest power is at 55%.

6. Conclusion

We presented a gated clock routing which has lower switched capacitance over buffered clock trees. We presented a clock topology generation heuristic based on the module activities and the sink locations. We proposed a method to find the signal probability and the transition probability of the gate control signals from the tables generated from the instruction stream.

Our experimental results showed that there is an optimum number of gates for lowest switched capacitance. Our results help a designer choose trade-off among the power, area and the complexity of the routing.

In this paper, we assumed a centralized gate controller. However we are also investigating a distributed controller for reduced star routing area. This is illustrated in Figure 6.

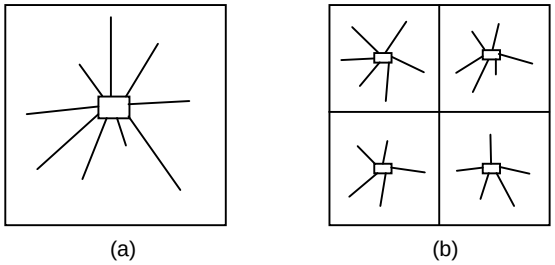

Figure 6: (a) one centralized controller vs. (b) four distributed controllers

Assume that the chip is square and its side is of length D. Then the longest edge length of the star tree is D/2. Assuming the average edge length is half of this (D/4), the total routing area is GD/4 where G is the number of gates. If we divide the chip into k equal sized partitions (where k is power of two), then each partition has G/k gates and the average edge length is $D/4\sqrt{k}$. Therefore the total routing area is

$$k\frac{G}{k}\frac{D}{4\sqrt{k}} = \frac{GD}{4\sqrt{k}}$$

As the number of partition increases, the star routing area is reduced by a factor of $\sqrt{k}$. Feasibility of the distributed controllers and their impact on the design complexity of the controller logic in currently under investigation.

References

[1] Mazhar Alidina, José Monteiro, Srinivas Devadas, Abhijit Ghosh, Marios Papaefthymiou, "Precomputation-Based Sequential Logic Optimization for Low Power," *IEEE Transactions on VLSI Systems*, vol. 2, no. 4, pp. 426-436, December, 1994.

[2] Kenneth D. Boese and Adrew B. Kahng, "Zero-Skew Clock Routing Trees With Minimum Wirelength," *Proc. IEEE International Conference on ASIC*, pp. 1.1.1-1.1.5, 1992.

[3] M. Edahiro, "Minimum Path-Length Equi-Distance Routing," *Proc. IEEE Asia-Pacific Conf. on Circuits and Systems*, pp. 41-46, 1992.

[4] Jaewon Oh and Massoud Pedram, "Power Reduction in Microprocessor Chips by Gated Clock Routing," to appear in Asia and South Pacific Design Automation Conference, 1998.

[5] Gustavo E. Téllez, Amir Farrahi, Majid Sarrafzadeh, "Activity Driven Clock Design for Low Power Circuits," *Proc. International Conference on Computer-Aided Design*, pp. 62-65, 1995.

[6] R-S Tsay, "Exact zero skew," *International Conference on Computer-Aided Design*, pp. 336-339, 1991.