Gated Clock Routing for Low Power Microprocessor Design

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. Gated clock trees include 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 the switched capacitance of the clock tree. We construct a clock tree topology based on the locations and the activation frequencies of the modules, while the locations of the internal nodes of the clock tree (and hence the masking gates) are determined using a dynamic programming approach followed by a gate reduction heuristic. This work assumes that the gates are turned on/off by a centralized controller. Therefore, the additional power and routing area incurred by the controller and the gate control signal routing are examined. Various tradeoffs between power and area for different design options and module activities are discussed and detailed experimental results are presented. Finally, good design practices for implementing the gated clocks are suggested.

Keywords: gated clock, low power, clock routing, VLSI

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. Recently, the system-on-a-chip (SOC) design has become popular. A typical SOC design consists of one or more CPUs, RAM banks, bus interface units, I/O and memory controllers, floating point coprocessor, etc. By shutting down the idle units, one can prevent the circuit from 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 wasteful switching in the clock tree and saves power in the clock tree in addition to saving power in the functional units, which are fed by the clock.

In general, processor instructions not executed with the same occurrence frequency, that is, some instructions are executed more frequently than others. This fact motivated the development of Reduced Instruction Set Computer architecture. Our work is also guided by this same principle: gated clock routing for low power is driven by the occurrence frequency of different instructions.

In this paper, we use the term *circuit module*, or simply *module,* to refer to a circuit block that has a single clock entry point for driving all of the clock pins of the registers in the block. The modules are leaf nodes of the clock tree.

We assume that the modules are sequential machines with latches at the inputs (see Figure 1) or a cluster of such sequential machines. This assumption is made because in such a machine, there is no switching activity once the clock is shut off. In fact, this type of machine is commonly used in

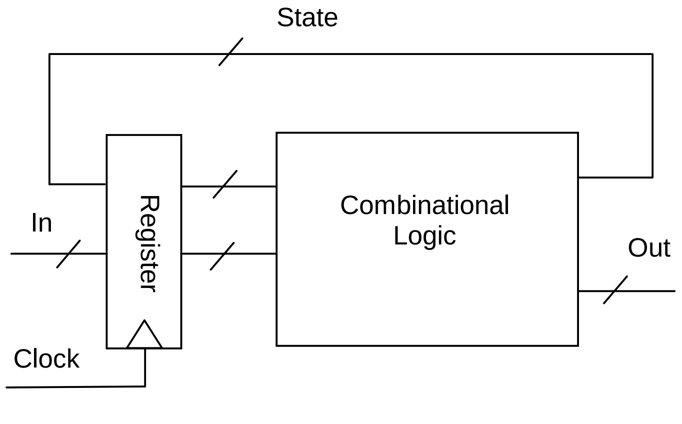

Figure 1: Model of a sequential machine

industrial designs for its simplicity and ease of timing verification [1]. The basic idea of clock gating is to mask off the clock to the registers of idle modules. This keeps the inputs of the combinational logic block steady, preventing any switching in the block. Therefore, dynamic power dissipation of the circuit is reduced.

In this paper, we address a particular form 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 example of a gated clock tree is shown in Figure 2, where the sink nodes correspond to the locations of modules and the internal nodes refer to the Steiner points of the clock routing.

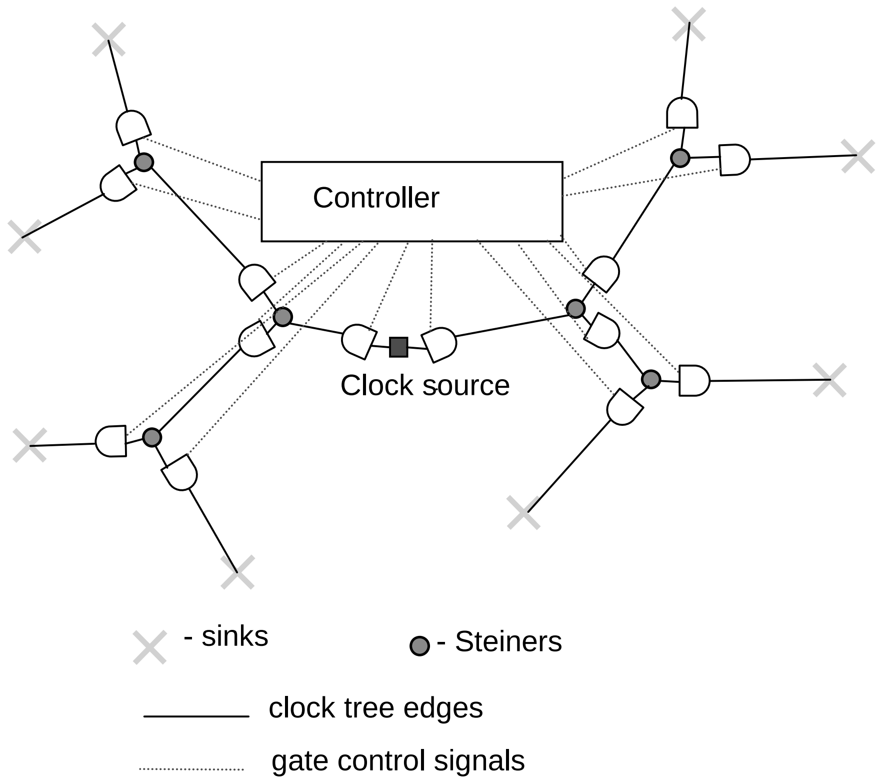

Figure 2: Model of a gated clock tree

A gate in the clock tree must be enabled (i.e., given a true control signal) 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.

In [6], an algorithm for constructing the topology of a gated clock tree was suggested. This algorithm takes advantage of the activity patterns of the modules in the circuit to design the clock tree topology. The authors use high-level design information to extract these activity patterns. However, the routing overhead and power dissipation cost of the control signals for the masking gates in the clock tree were not modeled.

In this paper, we propose a method for clock tree construction, which is based on detailed statistics about instruction frequencies and module activation per instruction. This information is extracted from instruction level simulation of the processor with a large number of benchmark programs and is compiled into Instruction Frequency and Instruction Transition-Module Activation Tables. Using these two tables, we are able to calculate the power dissipation of a candidate gated clock tree routing solution efficiently and accurately. In addition, we consider the control signal overhead in constructing the clock tree routing. We therefore show how probabilistic information (instruction usage and module activation statistics) and geometrical information (locations of the clock tree sinks) can be used to generate a low power clock routing solution.

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.

Preliminary work contributing to this paper has been published in [4] and [5].

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 (i.e., the depth of the 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 that e_i connects v_i to its parent in T. Let $| e_i |$ be the length of edge e_i. An example of the clock tree topology is shown in Figure 3.

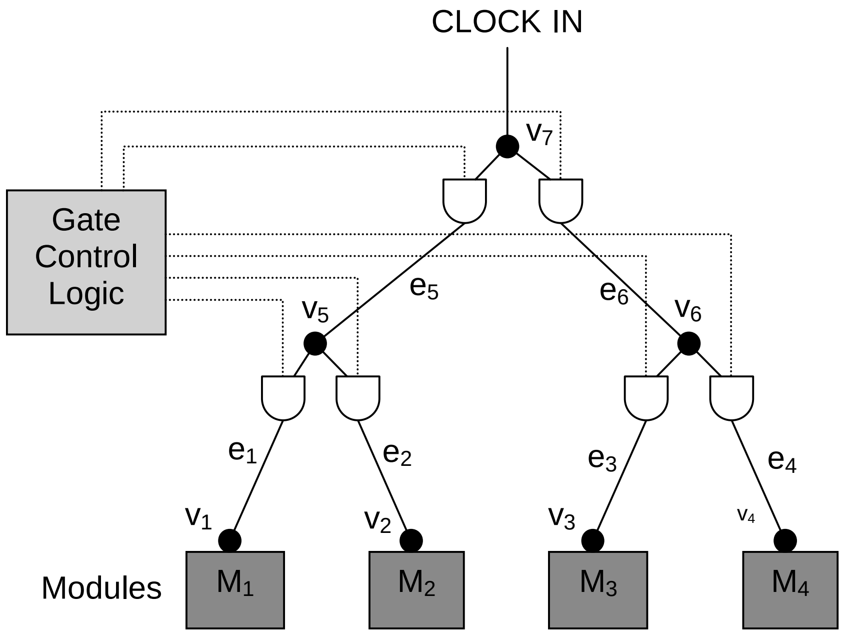

Figure 3: Topology of the gated clock tree

We assume that the controller is located at the center of the chip. The routing used for the control signals is *Star (shortest path) routing* as shown in Figure 2. 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 in the clock tree

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

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

where c_0, α, f, and 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

$$power(e_i) = c_0 \left| e_i \right| f\, V_{dd}^2$$

When the masking gates are present, power is dissipated only when the control signal is on. Thus the power dissipation in edge e_i is

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

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 defined as

$$w(e_i) = c_0 \left| e_i \right| P(EN_i)$$

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 becomes

$$w(e_i) = (c_0 \left| e_i \right| + C_i)\, P(EN_i)$$

The total switched capacitance in the clock tree is therefore

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

2.2 Switched capacitance in the controller tree

Similar to Equation (1), we can write the switched capacitance of edge EN_i as

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

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

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

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

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

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 node v_i has modules M_1, M_2,…,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

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

where $P(M_i \cup M_j)$ means the probability that either M_i or M_j is active.

To find $P_{tr}(EN_i)$, we need the module activation statistics over two 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 EN_i logic value transitions 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 EN_i transition, so we only need to consider cases 2 and 3 in 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 a 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 in Table 1, and these two instructions occur 15 times in the stream. Therefore $P(M_1) = 15/20 = 0.75$. 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. Since I_1 and I_3 are such instructions and they appear 11 times in the stream, $P(EN_i) = P(M_5 \cup M_6) = 11/20 = 0.55$.

The transition probability of this EN_i can be also found by scanning the instruction stream. We examine every two consecutive instructions during the scanning and count the number of EN_i transitions (a 01 transition occurs when both M_5 and M_6 are idle in the current clock cycle and any of M_5 or M_6 are active in the next clock cycle and a 10 transition is the inverse of this). We can see that $P_{tr}(EN_i)$ = 10/19= 0.526 (there are ten transitions out of 19 in the stream).

However, the instruction stream can be very long. Because some instructions are rarely executed, the instruction stream should be very long in order to get a reasonable probability value for the rare instructions. To get accurate instruction statistics, we may need some millions of 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 one scan of the instruction stream.

3.3 Table-driven probability computation

Instruction Frequency Table (IFT) enlists the probability that each instruction is executed in any cycle. 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 (IFT)

To find $P(M_5 \cup M_6)$, we simply add the two probabilities of I_1 and I_3 in Table 2, obtaining 0.55. Any signal probability $P(EN_i)$ can be found using Table 1 and Table 2 without rescanning the instruction stream. 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), as will be shown next.

Let IFT(I_i) be the instruction frequency of I_i (Table 2) and RTL(I_i) be the set of modules used during the execution of I_i (Table 1), and let $M(v_k)$ be the leaves of v_k. Then $P(EN_i)$ can be found by the procedure given below.

```
PROCEDURE ComputeSignalProb ($EN_k$)
Begin
  $P \leftarrow 0$;
  For each instruction $I_i$
    If RTL($I_i$) $\cap M(v_k) \neq \phi$ then
      $P \leftarrow P$ + IFT($I_i$);
  end for
  Return $P$;
End PROCEDURE
```

Let K be the total number of instructions. Also let $L = \max(|M(I_i)|)$ for all i. For the above procedure, we can implement disjoint set data structures on modules in which *FIND_SET*(M_i) and *UNION*(M_i) can be done in $O(1)$ and $O(N)$ time respectively. The **If** statement can be done by performing *FIND_SET*(M_i) for every used module in the instruction, which takes at most $O(L)$ time. With K iterations of the **For** loop, the above procedure takes $O(KL)$ time.

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

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 the $AT(M_k)$s, for $k=1,\ldots,l$ 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 EN_i is $O(K^2N)$, where K is the total number of instructions and N is the total number of modules.

The ITMAT can be constructed as a by-product of the instruction-level simulation of the processor. Creating the tables is very fast since it takes linear time in the length of the instruction stream. A question arises as to how long the instruction stream should be in order to obtain a reliable and dependable table. Certainly, the longer the instruction stream, the higher the accuracy of the table. The validation of any processor design involves simulation of long instruction streams. Consequently, processor designers end up generating a rich pool of benchmark data (instruction traces) during the course of their design, which can provide us with the data trace that is long enough to provide good accuracy for ITMAT entry calculation.

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 [7] 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 a 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 4).

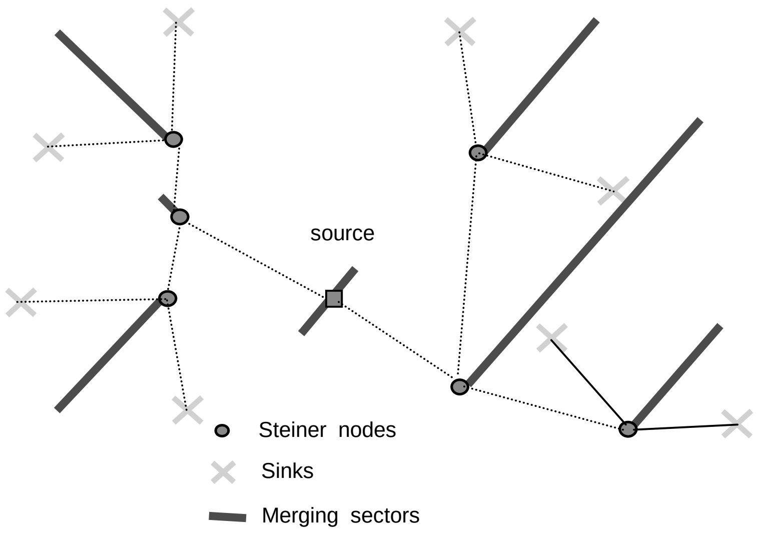

Figure 4: An example of bottom-up merging and top-down placement

The nearest-neighbor heuristic of [3] greedily merges two nodes when the geometric distance between the corresponding merging sectors is at a minimum. Our method is also greedy, but the merging sequence is determined by the switched capacitance.

Let $ms(v_i)$ be the merging sector of v_i. Suppose we try to merge $ms(v_i)$ and $ms(v_j)$ 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 merging $ms(v_i)$ and $ms(v_j)$ is

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

When we merge subtrees bottom-up, we merge sectors that result in the smallest switched capacitance (SC) as given in Equation (3). We have the algorithm outlined below.

```
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 creating the tables takes $O(B)$ time, where B is the length of the stream. The second and third statements take $O(N(KL + K^2N))$ and $O(N^2)$ time 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 $O(K^2N)$ time. So the overall time complexity is $O(B + K^2N^2)$.

Small switched capacitance means that

- the distance between v_i, v_j is short
- the activity of v_k will be small.

It has been shown that merging the nearest neighbors is effective in reducing the total wire length [3]. Also, if we merge small activities, the resultant activity will be also small. That is, if we merge nodes with higher activities first, in the merging sequences that follow, the activity of the nodes will be higher because activities increase monotonically as we go up the tree. This means that we want to bring the high activity nodes into the tree as late as possible so that the overall activity in the tree will be reduced. This is illustrated in Figure 5. Our method is similar to the tree construction of Huffman encoding except that our method considers the geometry of the nodes in addition to the probability of the nodes.

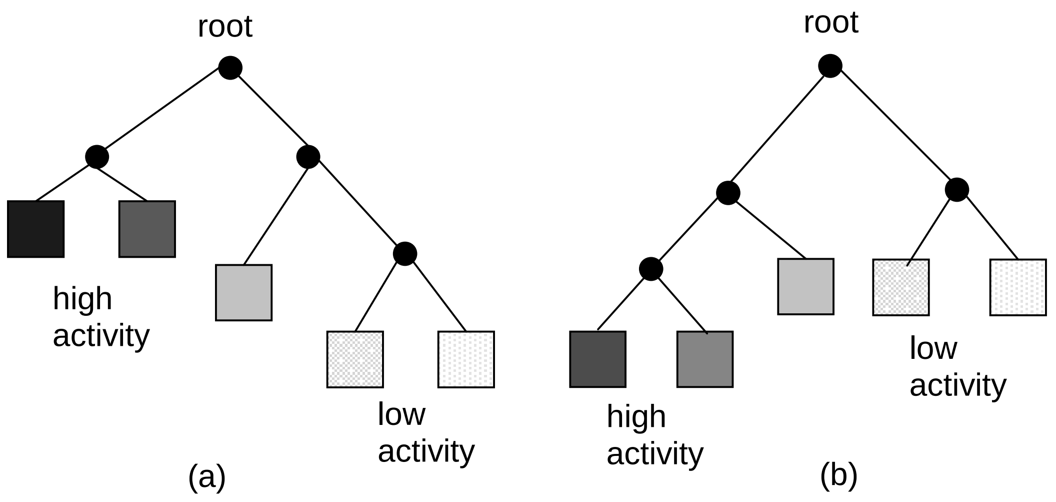

Figure 5: The two clock tree topologies dissipate different power. In (a), low activity nodes are merged first and the switching activities decrease as we go up the tree. In (b) however, high activity nodes are merged first, and the switching activities increase, thereby dissipating more power than (a).

We do not claim any optimality or near-optimality in our clock tree design. Our method is intended for use on top of the designers' clock layout methodology, such as a DME or DME-like methodology, to provide a means of reducing the power consumption in the circuit. Choosing the middle point CP as the distance measure may cause overestimation of the switched capacitance in some places and underestimation in other places. However, when summed up, these over/under estimations tend to cancel each other out when there are enough nodes. Choosing the middle point is a reasonable choice since we do not know where the Steiner nodes will be placed.

4.3 Reducing the Number of Gates

Inserting gates at every node of the clock tree may result in a large area and increase the complexity of the control circuit and the routing of the enable signals. This is especially significant because the routing of enable signals is Star-like, and therefore its area can be larger than that of 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 the activity of the node.

Case (1) is obvious since there is no timeframe 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 in which both nodes have gates.

The gate removal schemes may remove so many gates in the tree that the phase delay of the clock signal may increase rapidly. So we include a rule for enforcing a gate insertion regardless of these three rules as follows: if the subtree capacitance of the node reaches, for example $20C_g$, where C_g is the input capacitance of a gate, we insert a gate. However, when we override the gate reduction rule to in fact insert a gate, we insert a buffer instead since the purpose of this insertion is simply to improve the delay and not mask off the clock. Obviously, the gate control signal will be removed if a buffer is inserted instead of a masking gate. Using buffers and removing control signals contributed to further reduction of the switched capacitance.

5. Results

We implemented our algorithm in C++ on a Sun Sparc 20 workstation. For sink locations (module locations) and the sink load capacitance, we used benchmarks r1-r5 from [7]. The instruction stream and the used modules for each instruction were generated randomly. However, to simulate that some instructions are more frequently executed than the others, 10% of the instructions were made to appear 50% of the time in the instruction stream. The remaining 90% of the instructions made up the remaining 50% of the instruction stream. The benchmark characteristics are shown in Table 4. The length of the instruction stream was 100 thousands of instructions in all the benchmarks. The program ran for two minutes for the biggest benchmark r5 and ran in less time for other benchmarks.

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

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

Obtaining the right test bench set was the biggest difficulty in writing this paper. The test bench requires almost all the information about a processor, e.g., the instruction set, the RTL description of the design, the physical layout, and the technology information. Industry does not provide us with such detailed information about any of their products. So we had to rely on the design data for an artificially constructed processor. However we believe that the assumptions we made and the setup that we used to generate the test bench for our gated clock routing technique, are generally valid and industrial processors will exhibit nearly the same kind of behavior and statistics.

5.1 Buffered clock tree vs. gated clock tree

The clock tree buffering is a commonly used method in clock routing. The buffered clock tree is constructed using the nearest neighbor heuristic [3], and the size of a buffer is assumed to be half the size of an AND gate. The comparison between a buffered clock tree, a gated clock tree, and a gated clock tree with gate reduction heuristics is shown in Figure 6.

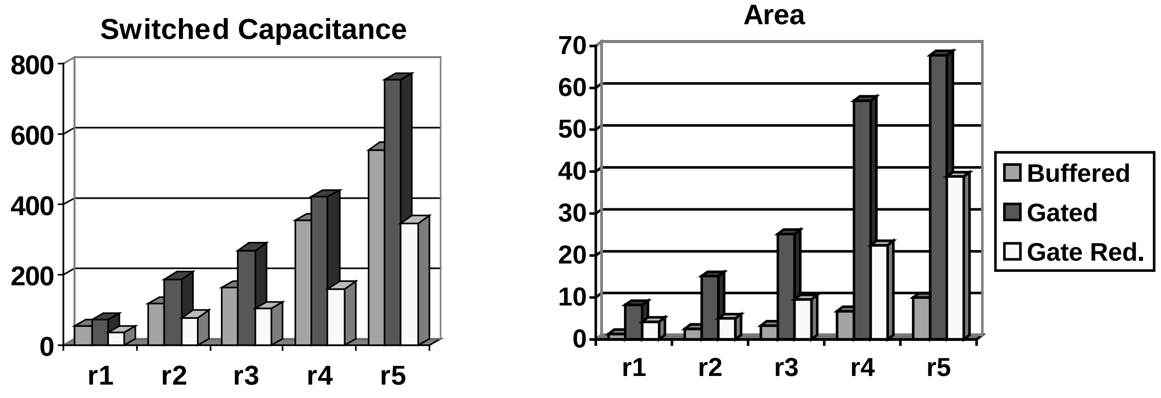

Figure 6: Comparison among different clock routing methods. Switched capacitance in pF, area in 10^6 units

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 the switched capacitance and the area comes from the Star-like routing scheme assumed for the control signals. After the gate reduction, it consumes about 30% less power than the buffered clock routing. However, there is still 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 7. 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.

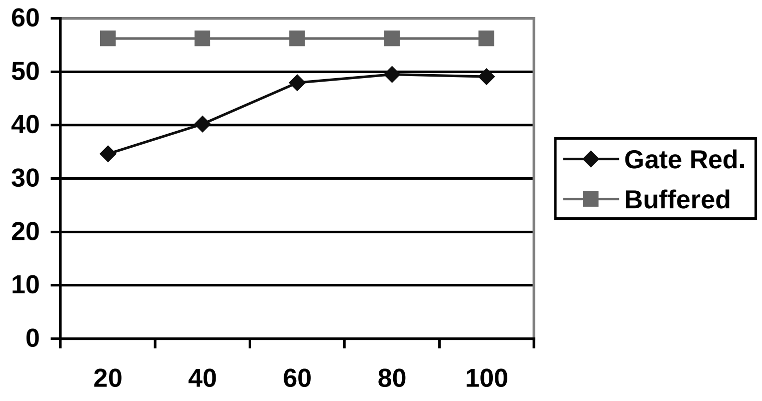

Figure 7: Average module activity (x-axis) vs. switched capacitance (y-axis) for benchmark r1

5.3 Optimum number of gates

The proportion of power dissipation between the clock tree and the controller tree is shown in Figure 8. We picked only one benchmark to show this proportion since other benchmarks showed similar graphs.

If there are a lot of masking gates in the clock tree, then the switched capacitance in the clock tree will be reduced, but the switched capacitance and the area of the controller tree will be increased. On the contrary, if there are too few gates, the switched capacitance in the clock tree will be increased. Intuitively, there is an optimum number of masking gates that minimizes the total switched capacitance. This optimum value is clearly observed in Figure 8.

We control 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 Figure 8, the optimum gate reduction to achive the minimum power dissipation is at 55%.

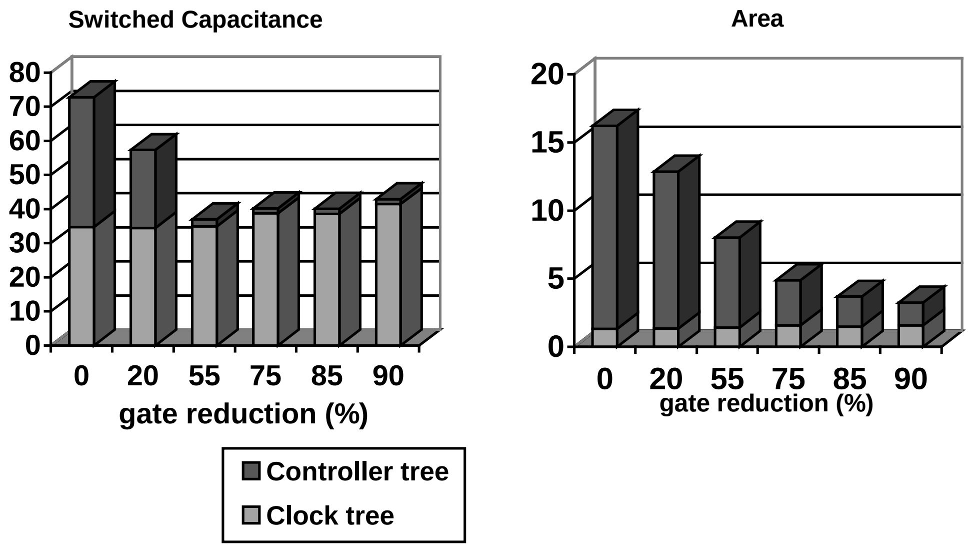

Figure 8: Gate reduction vs. switched capacitance and area for benchmark r1

6. Conclusion

We presented a gated clock routing that 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 the gated clock routing significantly reduces power dissipation over buffered clock routing with a marginal increase in the routing area and the number of gates. Furthermore, our experimental results showed that there is an optimum number of gates in order to achieve the lowest switched capacitance. This helps a designer choose a more effective trade-off between the power, area, and routing complexity.

We assumed a binary tree topology in order to take advantage of the efficient bottom-up merging (know as DME, Deferred Merge & Embedding) technique for clock routing. Notice however that the clock tree is binary only in its topology. In fact, zero skew merging may result in a tree where some edge length becomes zero (this indeed occurred in our test benches). In this case, two nodes of the tree collapse into one, resulting in a non-binary tree in physical layout.

The goal of this paper was to propose a power reduction technique based on gated clock routing, not to propose a new zero-skew or bounded-skew clock routing algorithm. Our basic model of the power consumption in a gated clock tree can be used with any other clock routing algorithm (e.g., a bounded-skew clock routing algorithm) to produce power savings.

We understand that the Elmore delay is no longer suitable for deep submicron design, but we used the Elmore delay in this paper as the basis for our delay calculator only for the sake of presentation. The Elmore delay model can indeed be replaced with other delay calculation equation. Furthermore, because in clock routing we are interested in the delay difference, the choice of the delay calculator is not as critical as an application in which the absolute delay accuracy is required.

In our power estimation of the clock tree, we have not included the short circuit power, the power associated with the additional control logic. However, also not included is the power saving on modules, which may be a more significant saving than the power saving in the clock tree itself. We believe that the power saving in the clock tree and the modules is large enough to compensate the additional power consumption due to the control logic and the enable signal routing. Furthermore, a module in this paper refers to (from coarse to fine grain), a chip, a large functional unit, or a flip-flop. If the granularity is too coarse, the benefit of power saving is less. On the contrary, if it is too fine, the complexity of gate control logic and the routing of the enable signal is too high. A simulation of different design styles may be needed to get optimum module granularity.

References

[1] Luca Benini and Giovanni De Micheli, "Automatic Synthesis of Low-Power Gated-Clock Finite-State Machines," *IEEE Transactions on Computer-Aided Design*, vol. 15, no. 6, pp. 630-643, June 1996.

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

[3] M. Edahiro, "Minimum Path-Length Equi-Distance Routing," *Proc. of 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," *Proc. of Asia South Pacific Design Automation Conference*, pp. 313-318, 1998.

[5] Jaewon Oh and Massoud Pedram, "Gated Clock Routing Minimizing the Switched Capacitance," *Proc. of Design Automation and Test in Europe*, pp. 692-697, 1998.

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

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

Appendix: Design issues in gated clock routing

In this appendix, we discuss some of the issues in designing the gated clock tree.

Timing

Assume that the clock rising edge or the clock 'high' triggers all the registers. The gated clock should be designed so that every sink must see the clock pulse as if the clock signal is never gated. Both the timing of the clock rising edge and the clock pulse duration should be preserved for correct operation of the entire circuit. To pass the correct clock pulses, all the gates from the root to the active module must be enabled before the clock edge comes in. That is, if a module is to be active in the current clock cycle, this fact must be known in the previous clock cycle. This can be done, for example, in a microprocessor with a pipeline. The instruction-decode stage determines the instruction type and modules to use, and then the control logic enables gates necessary to deliver the clock signal from the root to the specific modules in the execution stage. In that sense, the gate controller itself is a FSM since it needs to store a state at the instruction decoding stage and send out appropriate gate-control signals at the execution stage, which are not back-to-back clock cycle operations. The gate controller will need uninterrupted clocks all the time, and there may be some other critical modules that it will also need uninterrupted clocks. This will reduce the amount of power savings in the clock gating methodology.

If the gate enable signal comes while the clock is high, the sink may see unwanted transitions. Thus, the enable signal should be on/off only when the clock is low. Besides, the enable signals should not have glitches while the clock is high because this may introduce extra clock pulse. The timing diagram of an enable signal is shown in Figure 9.

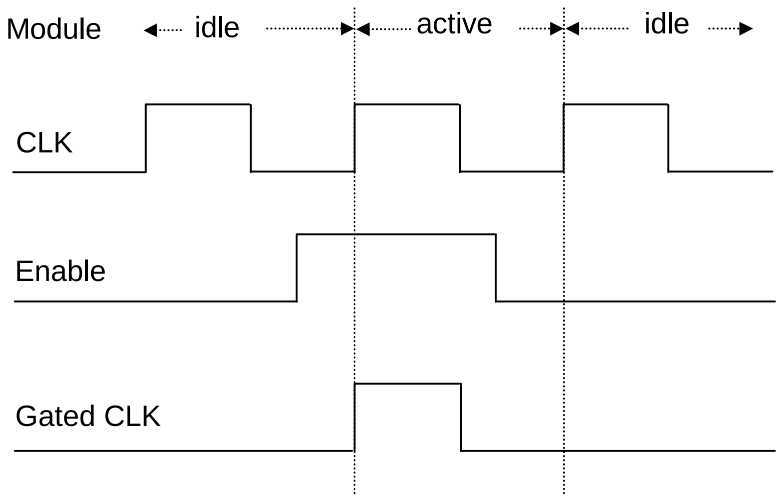

Figure 9: Timing requirements of the gate enable signals

In practice, however, designing a glitch-free circuit is difficult. Reference [1] suggests the use of a latch in addition to the gate to filter out glitches while the clock is high. Their method is illustrated in Figure 10. F_i is the signal to stop the clock when it is low, and L is a latch transparent when the clock CLK is low. When CLK is low, F_i pass through the latch to create gate enable signal EN_i. When CLK is high, EN_i retains its value regardless the value of F_i. This circuit ensures that no glitch in F_i is propagated while the clock is high.

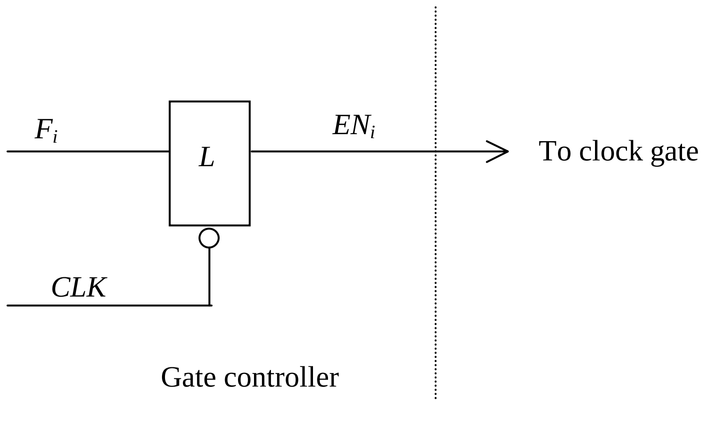

Figure 10: Generating a glitch-free gate enable signal

Distributed Gate Controller

Processors generate data path control signals for enabling tri-state buffers or for addressing MUX outputs to feed the data from the registers to specific combinational circuits. Naturally, these data path control signals have similar timing as the gate control signals. Some gate control signals may even be shared with existing data path control signals. Therefore gated clock control logic can be easily integrated with the existing control circuitry in the processor.

For simplicity in this paper, we assumed a centralized controller. However, it is possible to have a distributed controller. In a typical microprocessor, the modules are control block, data-path block, or memory block. Usually a data-path block or a memory block accompanies one or more control blocks nearby to control how they run. These control blocks can serve as the clock gate controllers since they have information about the active/idle states of the modules they are controlling. In addition, the clock masking gates that provide the clock signal to the modules they control, are apt to be in the neighborhood of the control block and the controlled block. The result of the distributed controller is a reduction in the length of a Star-like routing. This is illustrated in Figure 11.

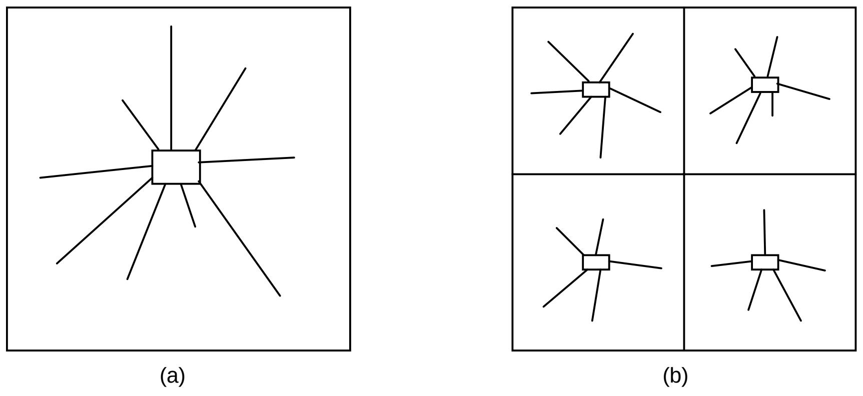

Figure 11: (a) one centralized controller vs. (b) four distributed controller

Assume that the chip is square and its side is of length *D*. The longest edge length of the star tree is *D*/2 (assuming Manhattan length). 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 a 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 $\sqrt{k}$.

Suppose a module is active for a number of consecutive clock cycles. This results in a waste of energy if enabling signals go on/off during this time. To prevent this, the gate enable signal may be designed to remain high for one or two clock cycles even after the module is gone idle. This prevents unnecessary switching of the enable signal between the consecutive active cycles of the module. Note that it is harmless to feed the clock during the idle time of the module.