Power Reduction in Microprocessor Chips by Gated Clock Routing*

Jaewon Oh and Massoud Pedram

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. The clock tree topology is constructed based on the locations and the activation frequencies of the modules and whereas 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.

1. INTRODUCTION

Clock gating methods recently gained attention as a way of reducing power dissipation in digital circuits. In a typical synchronous circuit, especially in a general purpose microprocessor, only a portion of the circuit is active at any given time. The remaining parts of the circuit are idle but may experience unnecessary switching, thereby dissipating power. In addition, instructions of the processor are not executed at the same rate. Some instructions are more frequently executed than others. This fact motivated the development of Reduced Instruction Set Computer (RISC) architecture. Our work is also motivated by this same fact, that is, the instruction frequencies are used for power optimized clock routing.

We assume that the modules are Moore-type sequential machines (see Figure 1) or a cluster of such sequential machines. This assumption is made because in a Moore machine, there is no switching activity once the clock is shut off. In contrast, in a Mealy machine, activity in the external input lines can cause a switching activity in the combinational logic even when the clock is shut off. The basic idea of clock gating is to mask off the clock to registers of the modules that are idle. This keeps the inputs of the combinational logic block steady, preventing any switching, and hence dynamic power dissipation in the circuit.

In this paper, we address the *gated clock routing* problem. We insert gates immediately after every internal node of the clock tree to minimize the dynamic power consumption (see Figure 2). These gates can also serve as buffers and can be sized for fine tuning the phase delay of the clock signal. A gate in the clock tree should be enabled (the control signal is true) whenever any of its descendant gates are enabled. This suggests that the control signal of a gate is the successive OR function of the control signals of its descendant gates. One should design the control circuit carefully in order to avoid hazards to the operation of the entire circuit. Some of the difficulties in designing the control circuit and their solutions are discussed in section 4.4.

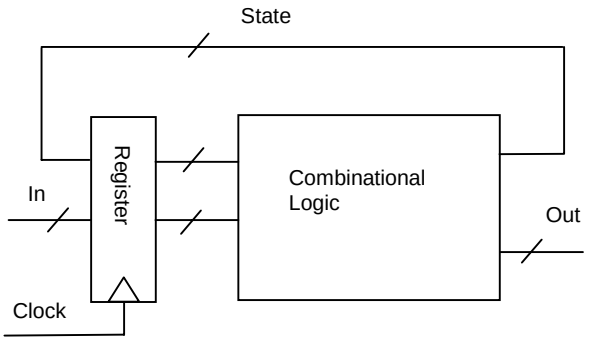

Figure 1: Modules are moore-type machines

In [7], a gated clock tree topology construction was suggested for DSP chips. The authors used high-level synthesis information to determine the tree topology. However, geometric locations of the registers were not considered. In contrast, our method applies to microprocessor chips and considers the placement of the registers. In addition, we propose a method for clock tree construction based on the instruction frequencies of the processor. Instruction frequency refers to the average percentage of the time an instruction will be executed in the real programs. This can be extracted from instruction level simulation of the processor with a number of benchmark programs. The instruction frequencies are used to extract the module activities as will be discussed in the following sections. We will investigate how the probabilistic information (instruction frequencies) 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 some terminology and the precise problem statement. Section 3 describes how activities of the clock tree nodes are calculated. Section 4 describes the clock tree construction based on activities and sinks' geometry. Sections 5 and 6 show our experimental results and conclusions.

* This research was funded in part by DARPA under contract no. F33615-95-C1627 and by NSF NYI under contratct no. MIP-9457392.

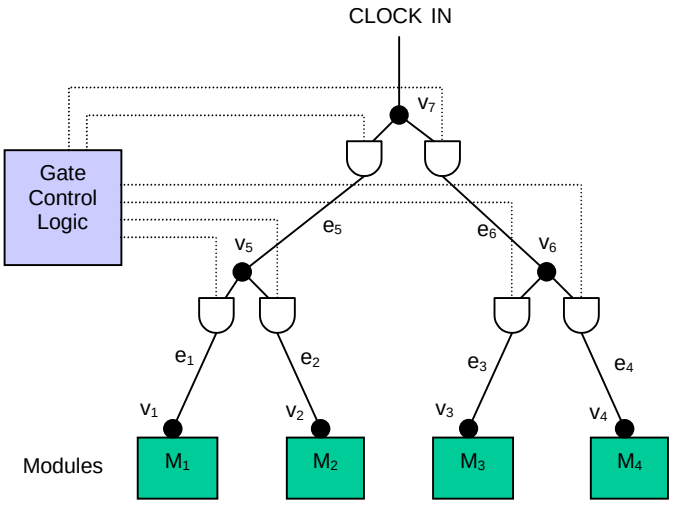

Figure 2. Gated clock tree

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, e_i connects v_i to its parent in T (see Figure 2). Let $|e_i|$ be the length of edge e_i.

2.1 Switched Capacitance

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

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

where c_0, α, f, V_{dd} are the unit wire capacitance, switching activity 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

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

If we can block the clock signal from reaching the edge e without causing any hazard to the entire circuit, the power dissipation in the clock tree can be reduced. We define *node activity* as the percentage of the time a tree node v is connected to the root of the clock tree through a series of enabled gates from the root to the node and denote it as $\delta(v)$. Since there is one edge for every node, this may be also called edge activity. With activity control, the power dissipation in edge e_i is

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

where the e_i, v_i relation was defined earlier.

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

$$w(e_i) = c_0 |e_i| \delta(v_i)$$

There can be a load capacitance associated with each node. Including the node capacitance C_i at v_i, the switched capacitance is given by

$$w(e_i) = (c_0 |e_i| + C_i) \delta(v_i)$$

The objective of our low power clock routing is to minimize

$$W(T) = \sum_{\forall e_i} (c_0 |e_i| + C_i) \delta(v_i)$$

subject to *zero skew* constraints.

3. ACTIVITY COMPUTATION

To get $W(T)$, we need to compute $\delta(v_i)$ for all the nodes. Let $P(M_i)$ be the probability that M_i is active (i.e. M_i receives the clock signal). By default, $\delta(v_i) = P(M_i)$ for any leaf node v_i. Suppose a non-leaf node v_k merges two leaf nodes v_i and v_j. Then the activity of v_k is given by

$$\delta(v_k) = P(M_i \cup M_j)$$

In general, for any internal node v_k,

$$\delta(v_k) = P(M_1 \cup M_2 \cup \cdots \cup M_l) \qquad (1)$$

where M_1, M_2,…,M_l correspond to the leaf nodes of the subtree rooted at v_k.

If a Register Transfer Level (RTL) simulation is used to find the probabilities in Equation (1), a huge number of clock-by-clock module usages have to recorded. Certainly, the time complexity will be very large. So we propose a method for computing activities using more efficient *instruction level simulation* of the processor and knowledge about RTL description of the processor.

3.1 Instruction Frequencies

By simulating the processor at the instruction level with a number of benchmark programs, we can find out how often each instruction is executed on the average. This information is often vital to processor optimization at the architectural level. Furthermore the RTL description of each instruction tells us what modules are used to execute each instruction. For example, we may have the following probabilities shown in Table 1. For each instruction, the probability of its occurrence is shown along with modules that are needed to execute the instruction.

Instruction	Used modules	Probability(%)
I_1	M_1, M_3, M_6	15
I_2	M_1, M_6	7
I_3	M_2, M_3, M_5	12
I_4	M_4, M_{10}	20
I_5	M_5, M_7, M_9, M_{12}	10
I_6	M_8, M_{13}, M_{14}	18
I_7	M_1, M_6, M_9, M_{11}, M_{13}	9
I_8	M_5	2
I_9	M_7	3
I_{10}	M_2, M_4, M_{15}	4
		total 100 %

Table 1: Instruction frequencies and module usages

3.2 Activity Computation

Suppose v_k has leaves$\{M_1, M_3, M_5, M_{11}\}$. Notice that if any module in $\{M_1, M_3, M_5, M_{11}\}$ is active at any time, v_k must be connected to the root. That is, for each instruction, if any of its used modules appear in $\{M_1, M_3, M_5, M_{11}\}$, it should be added up to the activity of v_k. Such instruction list includes $\{I_1, I_2, I_3, I_5, I_7, I_8\}$ and hence the activity of v_k is 55(%).

Let $P(I_i)$ be the instruction frequency of I_i, and let $\mathcal{M}(I_i)$ and $\mathcal{M}(v_k)$ be the set of used modules of I_i and the set of leaf nodes of v_k respectively. Then $\delta(v_k)$ is found by the procedure given below.

PROCEDURE ComputeActivity(v_k)
begin
  $\delta(v_k) \leftarrow 0$;
  for each instruction I_i
    if $\mathcal{M}(I_i) \cap \mathcal{M}(v_k) \neq \phi$ **then**
      $\delta(v_k) \leftarrow \delta(v_k) + P(I_i)$;
  end for
  return $\delta(v_k)$;
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 with which *FIND_SET*(M_i) and *UNION*(M_i) can be done in $O(1)$ and $O(N)$ respectively. The **if** statement can be done by performing *FIND_SET*(M_i) for every used modules in the instruction, which takes at most $O(L)$. Thus the above procedure takes $O(KL)$.

4. CLOCK TREE CONSTRUCTION

4.1 Delay modeling

To estimate the phase delay of the clock tree, we used Elmore delay modeling which was used in [8] for zero-skew clock routing. Our zero-skew clock routing method is the same as [8] except that we have gates at the internal nodes of the tree. Inserting gates reduces the subtree capacitance in the Elmore delay computation, thereby reduces 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 [4], merging sector is a line segment with slope ±1, which represents the possible locations of Steiner node where its two subtrees are merged, and these merging sectors are found in bottom-up. The actual locations of nodes in the merging sectors are determined in top-down fashion (see an example in Figure 3). The nearest-neighbor heuristic of [5] greedily merges two nodes when the geometric distance between the two corresponding merging sectors is minimum. Our method is also greedy, but the merging sequence is determined by the switched capacitance.

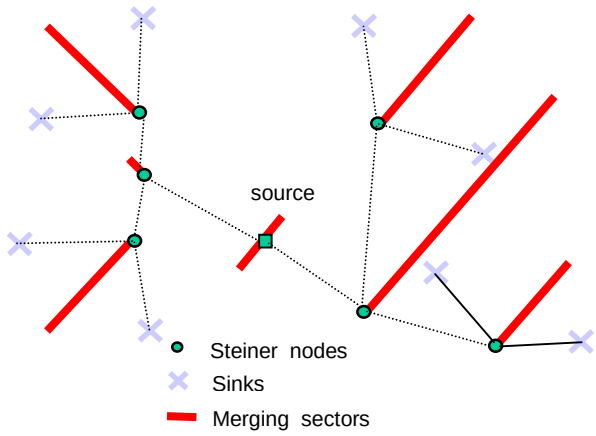

Figure 3: An example of bottom-up merging sequence

Let $ms(v_i)$ be the merging sector of v_i. Suppose we try to 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. Then the switched capacitance SC after the merge of ($ms(v_i)$, $ms(v_j)$) is

$$SC(v_i, v_j) = (c_0|e_i| + C_i)\delta(v_i) + (c_0|e_j| + C_j)\delta(v_j) \qquad (2)$$

When we merge subtrees bottom-up, we merge sectors that result in the smallest switched capacitance as given in Equation (2). Without $\delta(v_i)$ and the load capacitance C_i, C_j,

the problem is identical to that in [5]. Our entire algorithm is outlined below.

PROCEDURE GatedClockRouting
Input: Instruction Frequency,
Used modules for each instruction,
Sink locations
Output: Clock Tree Layout with gates

begin
// initially, every sink location is a merging sector itself
for each pair of $ms(v_i)$, $ms(v_j)$
determine $|e_i|$, $|e_j|$ satisfying zero-skew;
compute SC between v_i, v_j (Equation (2));
end for

// bottom-up merge
repeat
pick the pair $ms(v_x)$, $ms(v_y)$ whose SC is minimum
create new node v_k;
$\mathcal{M}(v_k) \leftarrow \mathcal{M}(v_x) \cup \mathcal{M}(v_y)$;
$\delta(v_k) \leftarrow$ ComputeActivity(v_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 (Equation (2));
end for
until only the root is left

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

The repeat loop iterates N times and within each iteration, the dominating complexity is either ComputeActivity which takes $O(KL)$ or the for loop which takes $O(N)$. So the overall complexity of our algorithm is $O(N\ (N + KL))$ (N, K, L are defined previously).

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 [5]. 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 nodes will be constantly higher because activities increase monotonically as we go up the tree. This means that we want to bring the high activity nodes to the tree as late as possible so that the overall activity in the tree will be reduced. Our method is similar to the tree construction of Huffman encoding [3] except that our method considers geometry of nodes in addition to the probability of nodes.

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. 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.

4.4 Design Issues in Gated Clock Routing

In this subsection, we discuss some of the issues in designing the control logic. Assume that all the registers are triggered by the clock rising edge or the clock `high'. 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 pipeline; instruction decoding stage determines the instruction type and modules to use, and then the control logic enables gates necessary to deliver clock signal from the root to the specific modules in the execution stage.

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 for processor.

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. In practice however, designing glitch-free circuit is difficult. [1] suggested to use a latch in addition to the gate to filter out glitches while the clock is high. However, we do not need to place latches at every internal nodes of the clock tree. It is sufficient to use latches only at the last stages of the tree (the gates immediately before the modules). The timing diagram of an enable signal is shown in Figure 4.

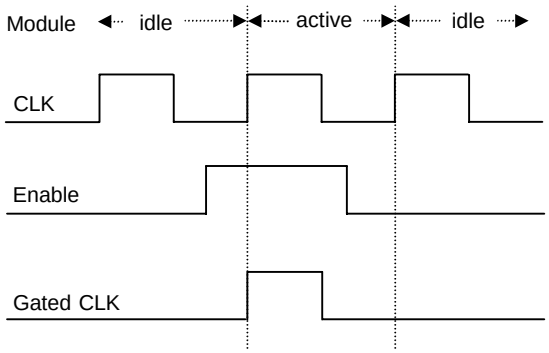

Figure 4: Timing requirements of the gate enable signals

Suppose a module is active for a number of consecutive clock cycles. It is a waste of energy if enabling signals go on/off when the module is active for a number of consecutive cycles. 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.

5. EXPERIMENTAL RESULTS

We implemented our algorithm in C++ for Sun Sparc 20 workstations. For sink locations (module locations) and the sink load capacitance, we used the benchmark r1-r5 from [8]. The instruction frequencies and the used modules for each instruction are generated according to a probabilistic model of the CPU when it executes typical programs. An instruction can have a number of different operand addressing modes. If an instruction has *k* different addressing modes, we consider it as *k* different instructions because each addressing mode will use different modules. The benchmark characteristics are shown in Table 2. The number of instructions in the table is a reasonable guess for a RISC machine when addressing modes are considered. The average number of used modules per instruction is made to be 40% for all the benchmarks (this can be seen in the column labeled Ave($\mathcal{M}(I_i)$)). 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.

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

Table 2: Benchmark characteristics for gated clock routing

Our program takes from 1 second to 2 minutes on a Sparc 20 workstation for the above benchmarks. Detailed CPU times are shown in the tables. We used 20 different random seeds to generate module usage of the instructions for each benchmark. The averages are shown in Table 3. In this experiment, buffers/gates are inserted at every node of the clock tree. The buffered clock tree is a commonly used method in current clock routing. The buffered clock tree is constructed with buffers whose size is half the size of AND-gates, which is a reasonable assumption.

Gated clock trees are constructed using two heuristics, the nearest neighbor heuristic and our proposed minimum switched capacitance heuristic. The table shows that the switched capacitance of the gated clock tree is much less than the buffered clock tree with only marginal increase in wiring costs over buffered clock tree. The power savings are almost 40% on the average, that is, the gated clock rooting consume only 60% of the power consumed by the buffered clock trees. Besides, our minimum switched capacitance heuristic further reduces switched capacitance about 9% over the nearest neighbor heuristic.

Another experiment shown in Table 4 uses gate reduction scheme presented in section 4.3. Buffers are inserted using the same scheme except that activities are disregarded. It can be seen that reductions in switched capacitance are still significant, that justifies the use of the gate reduction method. Due to the reduced number of gates, switching reductions of the gated clock trees are less than the previous experiment. However, the number of gates is about 15 - 20% of the previous experiment, which is a significant reduction in the area occupied by the gates.

6. CONCLUSION

We presented a gated clock routing which has significantly 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 that takes advantage of instruction level simulation to find the node activities of the clock.

Our experimental results showed that the gated clock routing significantly reduces power dissipation over buffered clock routing with marginal increase in routing area and the number of gates. Furthermore, our proposed minimum switched capacitance heuristic further reduced power dissipation over the nearest neighbor heuristic.

In our power estimation of the clock tree, we have not included the short circuit power, the power associated with the additional control logic and with the routing of the enable signals. However, also not included is the power saving on modules, which is 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 are 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 (PCB, MCM) or a large functional unit (Floating-Point Arithmetic) or any sequential elements in a circuit. In case the module has several clock sinks, the sink location of the module should be approximated as the geometric center of the module's original sinks. A designer should consider trade-off among the power, area and the complexity of the gate control logic. 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] 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] Luca Benini, 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.

[3] T. Cormen, C. Leiserson and R. Rivest, "Introduction to Algorithms," *The MIT Press*, pp. 337-343, 1990.

[4] 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.

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

[6] David Patterson and John Hennessy, "Computer Architecture: A Quantitative Approach," *Morgan Kaufmann Publishers, Inc.*, 2nd Edition, 1996.

[7] 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.

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

Bench		r1	r2	r3	r4	r5
Buffered Tree * (NN)	total wiring	1297790	2550399	3288430	6755468	9971453
	switched cap [1]	56236	119395	164935	354116	555859
	CPU time(sec)	0.36	1.58	3.34	15.67	41.34
Gated Clock Tree (NN)	total wiring	1297790	2550399	3288430	6755468	9971453
	switched cap [2]	30699	78710	96465	212117	306425
	CPU time(sec)	0.41	1.8	3.76	17.42	45.97
	([1]-[2])/[1] (%)	45.4	34.1	41.5	40.1	44.8
Gated Clock Tree (MSC)	total wiring	1537978	3187166	4064631	8353696	12164903
	switched cap [3]	27878	72763	88016	194129	277833
	CPU time(sec)	0.64	3.23	6.73	33.91	88.95
	([2]-[3])/[2] (%)	9.2	7.6	8.8	8.5	9.3

Table 3: Routing costs and switched capacitances of buffered clock tree vs gated clock tree. Buffers/gates are placed at every node. Note: * - buffer size = 1/2 gate size, NN - Nearest Neighbor heuristic, MSC - Minimum Switched Capacitance heuristic, wiring cost in λ, capacitance in femto Farad(fF).

Bench		r1	r2	r3	r4	r5
Buffered tree (NN)	total wiring	1320155	2664806	3389032	6842366	10247289
	switched cap [1]	43784	92464	124646	261174	405375
	no. of buffers	102	220	312	648	1000
	CPU time(sec)	0.37	1.59	3.30	15.73	41.34
Gated Clock Tree (NN)	total wiring	1321959	2662414	3381709	6829282	10198714
	switched cap [2]	25936	57045	65393	155272	256748
	no. of gates	118	248	348	732	1108
	CPU time(sec)	0.41	2.06	3.77	17.62	46.35
	([2]-[1])/[1] (%)	40.7	38.3	47.5	40.5	36.7
Gated Clock Tree (MSC)	total wiring	1702113	3489634	4643665	9812025	14101057
	switched cap [3]	22767	49930	61076	148869	235117
	no. of gates	176	348	456	1006	1444
	CPU time(sec)	0.77	6.97	8.07	40.85	107.92
	([2]-[3])/[2] (%)	12.2	12.5	6.6	4.1	8.4

Table 4: Routing costs and switched capacitances of buffered clock tree vs. gated clock tree with gate reduction scheme. Buffers are inserted when a node's subtree capacitance is more than $20C_g$.