

# Synthesis of Selectively Clocked Skewed Logic Circuits

Aiqun Cao, Naran Sirisantana, Cheng-Kok Koh and Kaushik Roy

*School of Electrical and Computer Engineering*

*Purdue University, West Lafayette, IN.*

{caoa, sirisant, chengkok, kaushik}@ecn.purdue.edu

## Abstract

*Skewed logic circuits with selective clocking scheme have performance comparable to that of Domino logic, but consume much lower power. Unlike Domino, the reconvergent path problem in skewed logic circuits may be overcome without logic duplication due to the static nature of skewed logic. In this paper, we propose a novel approach that alleviates the need for logic duplication when dealing with reconvergent paths in skewed logic circuits. We also propose a dynamic programming-based heuristic to determine a low-power clocking scheme for skewed logic circuits. Experimental results show that 32% of gates in a skewed logic circuit are duplicated, whereas 69% of gates in a Domino logic circuit are duplicated. The total power saving of skewed logic over Domino logic is 32.6% on average.*

## 1. Introduction

Dynamic circuits, such as Domino logic, are commonly found in circuitry that requires very high performance. However, dynamic circuits dissipate a much higher level of power than static circuits. Moreover, the noise margin of dynamic circuits is very small, as it is dependent on the threshold voltages of transistors. That makes dynamic circuits extremely susceptible to failures due to threshold voltage variations and noise injection; they do not scale well with the technology.

To overcome the drawbacks of dynamic circuits, a new logic style, called skewed logic [2,3] or Monotonic Static (MS) CMOS logic [1], was proposed. A skewed logic gate has the same circuit topology as a classical static CMOS gate but the size of pull-down network (PDN) is decreased and the size of pull-up network (PUN) is increased or vice versa for fast low-to-high or fast high-to-low transition, respectively. Sizing the PUN and PDN to favor one transition direction is referred to as skewing. The structures of a skewed-down NAND gate (for fast high-to-low transition) and a skewed-up NOR gate (for fast low-to-high transition) are shown in Figures 1(a) and (b), respectively.

Operating as a precharge-evaluate circuitry, skewed logic circuits can achieve performance comparable to that of dynamic circuits. A skewed gate is precharged to the logic value that allows only fast transition in the evaluation phase. For fast evaluation, skewed-down gates are followed by skewed-up gates, and vice versa. Precharging can be accomplished either by clocked skewed logic gates, which precharge just like Domino gates (see Fig. 1(c)), or by the propagation of precharged logic values through the logic chain originating from a clocked gate. For the second option, it is important the precharging of skewed logic gates between two clocked gates does not exceed the precharge phase of the clock period. Ref. [1] considered clocking of all skewed gates, whereas Ref. [3] explored the second option by clocking skewed gates *selectively*. (see Fig. 2). Precharging through propagation allows a smaller number of clocked gates. That leads to a lower clock load and hence potentially lower clock power consumption than Domino circuits.

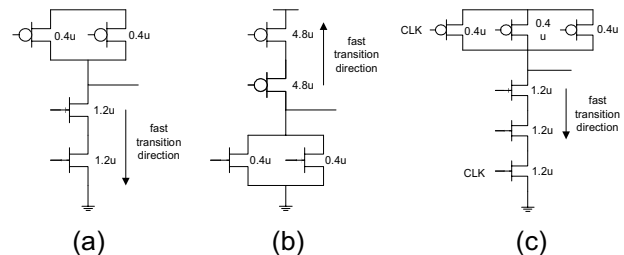


Fig.1 Two-input (a) NAND, (b) NOR, and (c) clocked NAND skewed logic gates.

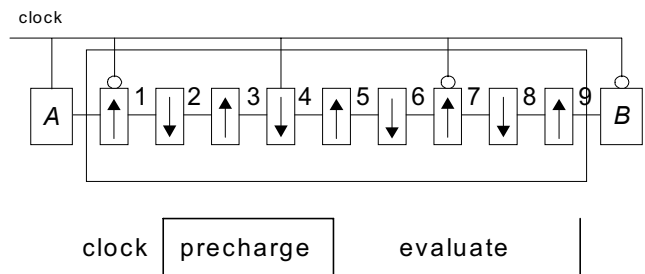


Fig.2 A chain of skewed logic gates between two latches. The skew direction is indicated by the arrow.

In this paper, we consider the problem of synthesizing a selectively clocked skewed logic (SCSL) circuit: *Given a logic network to be implemented by skewed logic, and the clock period (possibly with uneven duty cycles), determine the skew direction and skew value of each gate, and whether the gate is clocked, such that the total power consumption is minimized.*

We propose a two-step approach for the synthesis of SCSL circuits: First, we determine the skew direction of each gate. Second, we apply a dynamic-programming-based technique to determine the skew value of each gate, and whether the gate is clocked. The task of assigning skew directions, has also been considered in [1]. The main difficulty of the problem is due to the existence of reconvergent paths, which may render the assignment of alternating skew directions impossible (see Fig. 3). Treating skewed logic like Domino logic, [1] applied the technique of logic duplication to obtain a unate function, which ensures that an assignment of alternating skew directions is always possible. However, logic duplication is expensive. In this paper, we overcome this problem by observing that under certain conditions, assigning non-alternating skew directions to skewed logic gates does not impede the performance of the circuit. Experiment results show that with this method, the cost of duplication in skewed logic circuits is less than half of that in Domino on average, and the average power saving over Domino is 32.6%.

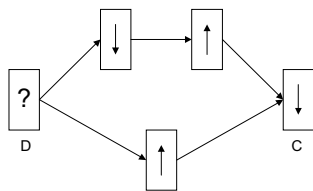


Fig.3 Logic reconvergence in skewed logic.

## 2 Assignment of Skew Directions

### 2.1 Reconvergent Paths

In Fig. 3, the reconvergent paths cause a problem in the assignment of alternating skew directions because the difference of the logic depths of the two paths is an odd number. In the following discourse, we consider only reconvergent paths of that nature. Given some reconvergent paths, we refer to the node from which the reconvergent paths depart as the *divergent node*, and the node at which the paths converge the *convergent node*. All the other nodes in the reconvergent paths are called *intermediate nodes*.

Consider the reconvergent paths in Fig. 3. At first glance, the problem can be easily resolved by duplicating the fan-in cone that feeds the divergent node and the divergent node itself. The duplicates can then drive the

top path and bottom path separately (Fig. 4). At most, the number of gates in the whole circuit doubles. That is an established approach for synthesizing Domino circuits [4, 5].

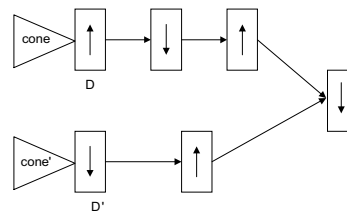


Fig.4 Logic Duplication.

However, a careful analysis of the *static* skewed logic circuit reveals that we can make use of the NOR gate *B* in Fig. 5(a) to *avoid logic duplication* as follows: We make both *B* and the divergent node, *D*, skewed-up gates. The other fan-in to *B*, gate *A*, and the intermediate node after *B*, gate *F*, are both skewed down, as shown in Fig. 5(b). The skew directions of gates *D* and *B* are non-alternating.

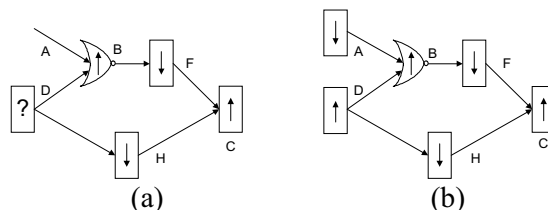


Fig.5 (a) Reconvergent paths with a NOR gate. (b) Non-alternating skew directions.

Although *D* and *B* are skewed in the same direction, *B* still properly precharges (to logic 0) because of gate *A*, which precharges to logic 1 as shown in Fig. 6 (The precharge value is shown after each gate in the figure). In other words, we make use of the fact that gate *B* is a static CMOS circuit to accommodate non-alternating skew directions. As skewed logic achieves its performance by allowing only fast transition during the evaluation phase, such a scheme requires that if gate *D* switches, it switches earlier than gate *A*. Otherwise, a glitch will appear at the output of *B*, and that will affect the performance of the circuit.

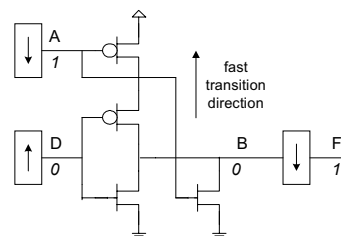


Fig.6 Transition of a NOR gate.

Note that for any NOR gate that precedes a skewed-down gate, we can always skew up the NOR gate and its faster fan-in gate as in Fig. 6 such that only fast transitions occur during the evaluation phase. When the NOR gate has more than two fan-in gates, in general, non-alternating skew directions can be assigned as long as the skewed-up fan-in gates switch, if they do switch, earlier than fan-in gates that are skewed down. A similar analysis will reveal that the skew direction assignment shown in Fig. 7(b) with the NAND gate as the middle gate also achieves the same effect. Similar configurations can also be obtained for AND and OR gates.

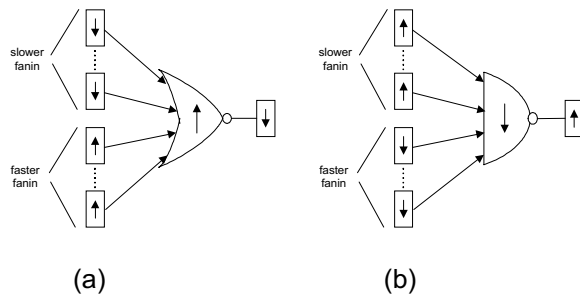


Fig.7 Skew directions for NOR and NAND gates.

The key limitation here is that when non-alternating skew directions are considered, the only skew direction feasible for the fan-out gate of a NOR (NAND) gate is skewed down (up). Consider the case of a NOR gate. If the fan-out gate is skewed-up, it has to precharge to logic 0, which indicates that the NOR gate has to precharge to logic 1, and therefore all fan-ins to NOR have to precharge to logic 0.

The implication is that depending on the types of intermediate nodes and convergent nodes in the reconvergent paths, the number of skew directions for the divergent and convergent nodes may be restricted; that determines the flexibility that we have in avoiding logic duplication. In Fig. 5(a), for example, if gates  $F$  and  $H$  are inverters, and the convergent node  $C$  is a NOR gate, then the only possible skew direction assignment to avoid gate duplication is to skew up the convergent node  $C$ . We refer to the gates that allow non-alternating skew directions as *candidate gates*. For a gate to be considered a candidate gate, it is important that the earliest switching fan-in gate is along the reconvergent paths. Since until this step, no exact timing information can be obtained without transistor size of each gate, it is necessary to estimate the path delay of each gate in the circuit. Several delay estimation methods are available: we may use the logic level of the gate (from the primary inputs (PIs)) to approximate the path delay of the gate, or we may use the nominal delay of each gate and its number of fan-outs to approximate gate delay and hence, path delay. After considering the timing information, the divergent node

would have only one skew direction if the convergent node is the only candidate.

There are two other possible scenarios for the skew direction assignment of the convergent and divergent nodes:

- (a) There exist candidate nodes that make assignment of either skew direction possible.
- (b) There exist no candidate nodes. Therefore, either skew direction is possible, but logic duplication of the fan-in cone and the divergent node is absolutely necessary.

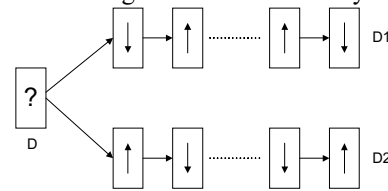


Fig.8 Conflict of divergent nodes  $D1$  and  $D2$ .

When a divergent node  $D1$  has only one possible direction (to avoid duplication), that direction may conflict with the only skew direction allowed for another divergent node  $D2$  (see Fig. 8). To resolve such conflicts, the fan-in cone of node  $D$  has to be duplicated. Similar conflict may exist for two convergent nodes that belong to the fan-in cone of some node. Therefore, we can state the problem of skew direction assignment as follows: *Given a logic network, determine the skew directions of the gates such that the amount of logic duplication is minimized.*

## 2.2 Pseudo-Reconvergent Paths

In fact, incompatibility exists not only among reconvergent paths. Consider Fig. 8 again, but replace  $D1$  and  $D2$  with two primary outputs, PO1 and PO2. If skew directions of PO1 and PO2 are not properly assigned, the skew direction of node  $D$  cannot be assigned in a consistent manner. We refer to paths that lead to the abovementioned problem as pseudo-reconvergent paths. These pseudo-reconvergent paths have one divergent node and multiple "convergent" nodes. Similarly, pseudo-reconvergent paths with one convergent node and multiple "divergent" nodes can be caused due to inappropriately assigned PIs.

When the problem is due to the assignment of the PIs or POs as in Fig. 8, the problems may be eliminated by choosing appropriate skew directions for them. However, the assignment problem can be complicated by pseudo-reconvergent paths that intertwine with each other as shown in Fig. 9. The skew direction assignment for the pseudo-reconvergent paths originating from gate  $A$  is incompatible with that for the pseudo-reconvergent paths originating from gate  $B$ , unless there exist candidate nodes that make non-alternating skew directions feasible. In other words, the fan-in cone of gate  $A$  or  $B$  must be duplicated to resolve the issue.

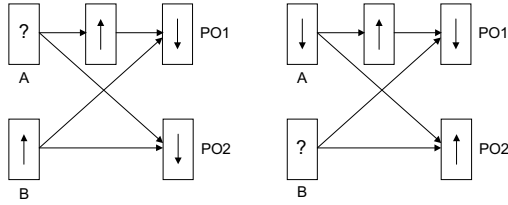


Fig.9 Intertwined Pseudo-Reconvergent paths.

### 2.3 Skew Direction Assignment Algorithm

Therefore, the problem of skew direction assignment is equivalent to that of resolving the conflicts that arise due to the incompatibility among the reconvergent paths and pseudo-reconvergent paths. The goal is to minimize logic duplication. The equivalent problem can be formulated by an incompatibility graph  $G(V,E)$  as follows: a vertex in  $V$  corresponds to either reconvergent paths or pseudo-reconvergent paths. There is an edge in  $E$  connecting two vertices if the two corresponding reconvergent paths or pseudo-reconvergent paths are incompatible. Each vertex is assigned a weight corresponding to the duplication cost of the fan-in cone feeding to the divergent node of the corresponding reconvergent paths or pseudo-reconvergent paths.

A solution to the skew direction assignment problem is to find a least cost vertex cover such that every edge in  $G$  connects at least one vertex in the cover. For every vertex in the cover, duplication of the corresponding fan-in cone renders it compatible with those it was originally incompatible with. We take the approach proposed by [8, 6] to solve the vertex cover problem, which is NP-hard: The optimal solution can be obtained by using a branch and bound algorithm on a binary decision diagram (BDD) derived from the incompatibility constraint [6]. Alternatively, the heuristic proposed by [8] can be used. In this paper, we use the branch and bound algorithm.

Subsequently, the skew direction of every gate can be obtained by propagating from all single-direction divergent nodes to the rest of the circuit.

### 3 Selective Clocking Scheme

Now that we have determined the skew direction of each gate, the next step is to determine the skew value, i.e., the ratio of the PMOS and NMOS transistor sizes, of each gate. We assume that a library of skewed logic gates are given. Every logic gate has a few implementations corresponding to different skew values in the library. Different skew values result in different precharge and evaluation delays, denoted respectively by  $t_{pc}$  and  $t_{ev}$ , and thus, different clocking schemes. In this paper, we present a dynamic programming based heuristic algorithm to find the clocking solution. The goal is to find the skew values of the gates such that the total number of clocked gates is minimized, while satisfying the precharge and evaluation time constraints. First, we consider the special case when

the netlist has a tree structure with the primary output as a root. Second, we consider the case when the netlist is a directed acyclic graph.

#### 3.1 Netlist with a Tree Structure

With a tree structure, it is natural to apply a dynamic programming based approach to assign skew values of skewed gates and to determine the location of clocked gates. We apply the dynamic programming approach in a bottom-up fashion, i.e., from the first level logic gates, which should be clocked (as seen in Fig. 2).

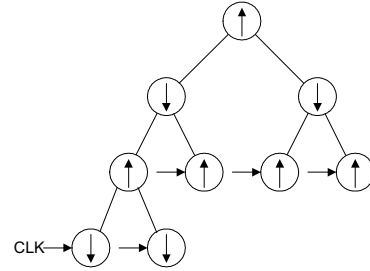


Fig.10 Tree with leaf nodes connected to clock.

Before we present the algorithm, we first define precharge slack and evaluation slack. Consider a skewed gate  $N$ . Let  $CFI$  be the set of clocked gates that are in the fan-in cone of  $N$  (including  $N$ ) such that there exist no unclocked gates along the paths from  $N' \in CFI$  to  $N$ . The precharge propagation delay is the sum of the longest precharge delay from gates in  $CFI$  to  $N$  and the precharge delay of  $N$ . The precharge slack at  $N$  is obtained by subtracting the precharge propagation delay from the given duration of precharge period. The evaluation propagation delay is the sum of the longest evaluation delay from the primary inputs to  $N$  and the evaluation delay of  $N$ . The evaluation slack at  $N$  is obtained by subtracting the evaluation propagation delay from the given duration of evaluation period.

Given a tree  $T$ , we associate each node in  $T$  with a list of triples:

$$\{(p_1, e_1, 0), \dots, (p_k, e_k, 0), (p_{k+1}, e_{k+1}, 1), \dots, (p_m, e_m, 1)\}$$

For the triple  $(p_i, e_i, b_i)$ ,  $p_i$  is the precharge slack,  $e_i$  is the evaluation slack, and  $b_i$  is a binary value: '0' means that the gate is not clocked, and '1' means that the gate is clocked. We construct the list in a bottom up fashion as follows.

Let  $N$  be a leaf node in  $T$ . Let  $T_{pc}$  and  $T_{ev}$  be the given durations of precharge and evaluation periods, respectively. For every skew value of  $N$ , we create a tuple  $(p, e, 1)$ , where  $p = T_{pc} - t_{pc}$ , where  $t_{pc}$  is the precharge delay of  $N$ , and  $e = T_{ev} - t_{ev}$ , where  $t_{ev}$  is the evaluation

delay of  $N$ . The third term in the triple is always '1' as  $N$  is a primary input gate.

Consider an internal node  $N$  in  $T$ , we construct the list of triples of  $N$  from its child nodes. For simplicity, we assume that  $N$  has two child nodes with the following two lists of triples:

$$\{(p_1', e_1', b_1'), (p_2', e_2', b_2'), \dots, (p_m', e_m', b_m')\},$$

$$\{(p_1'', e_1'', b_1''), (p_2'', e_2'', b_2''), \dots, (p_n'', e_n'', b_n'')\}.$$

For a particular skew value of  $N$ , let  $t_{pc}$  and  $t_{ev}$  be the precharge and evaluation delay if  $N$  is not clocked, and let  $t'_{pc}$  and  $t'_{ev}$  be the precharge and evaluation delay if  $N$  is clocked. Therefore, the two triples obtained when we combine  $(p_i', e_i', b_i')$  with  $(p_j'', e_j'', b_j'')$  are:

$$(\min(p_i', p_j'') - t_{pc}, \min(e_i', e_j'') - t_{ev}, 0), \text{ and}$$

$$(T_{pc} - t'_{pc}, \min(e_i', e_j'') - t'_{ev}, 1).$$

There are in total  $O(m*n*s)$  combinations for  $N$ , where  $s$  is the total number of skew value choices for  $N$ . As a result, there are exponential number of combinations for the entire tree.

Fortunately, various pruning criteria can be deployed to simplify the search process. Whenever a triple has a negative slack, for example, it constitutes an infeasible solution and should be discarded. Moreover, consider two triples of a node:  $(p_i, e_i, b_i)$  and  $(p_j, e_j, b_j)$ . We observe that if  $b_i = b_j$ ,  $p_i \leq p_j$ , and  $e_i \leq e_j$ , then  $(p_i, e_i, b_i)$  is an inferior solution; it can be eliminated from the list. Not only can the total number of combinations for  $N$  be reduced to  $O((m+n)*s)$ , the runtime of the algorithm can also be reduced accordingly [9].

After traversing the whole tree, we select among the triples at the root node one that requires the least number of clock number. The skew values for the remaining nodes can be obtained by a top-down traversal of the tree.

### 3.2 General Logic Network

When the logic network is a directed acyclic graph, it is very difficult to apply dynamic programming to find appropriate skew values and clocking scheme; c.f. sizing of Domino gates [7]. A common technique, which we adopt here as well, is to first decompose the netlist into trees. Several ways of decomposing a netlist into trees are available. In this paper, we first compute the logic depth of all primary outputs. We find the fan-in cone of the primary output with the largest logic depth, and use that to define a tree rooted at the primary output. We remove the nodes in the tree from the netlist, and proceed to extract trees from the remaining netlist in a similar fashion.

Clearly, not all the leaf nodes of the extracted trees are primary inputs. Our implementation ensures that the

leaf nodes of the extracted trees are initialized with appropriate precharge and evaluation slacks.

### 3.3 Further Reduction of Clocked Gates

If we assume a 50-50 duty cycle for the clock, there is a further step to cut down the number of gates connected to clock. After obtaining the clocking scheme as outlined above, we can always eliminate the clock connected to the gates in the *CFI* of the primary outputs.

As Fig. 11 shows, the clock connected to gate 7 in Fig. 2 can be deleted without affecting the correct operation of the circuit. After removing the clock, if gates 7, 8 and 9 finally keep the precharge value, they still have the entire duration of the evaluation period to perform the precharge properly. Otherwise, they would still evaluate correctly. If the duty cycles are uneven, only some clock gates in the *CFI* of primary outputs may be unlocked.

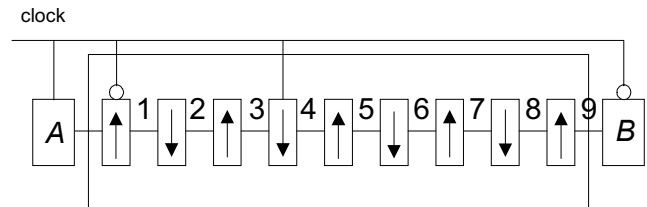


Fig.11 Eliminating the last stage clock.

## 4. Experimental Results and Conclusion

We have implemented the synthesis algorithms presented in Sections 2 and 3 in C language under the Berkeley SIS environment. For comparison, we consider nine ISCAS benchmark circuits implemented as Domino circuits and skewed logic circuits. For Domino circuits, we limit the library of gates to contain only INVERTER, 2-input NAND and up to 6-input NOR gates. Note that logic duplication is used to resolve the issue of logic reconvergence. For skewed logic circuits, the library contains INVERTER, up to 4-input NAND and NOR gates, with six different skew values each (three for skewing up and three for skewing down).

For each benchmark circuit, we compare the total power consumption of the two implementations running at the same clock frequency (as dictated by how fast the Domino circuit is). Circuit simulation using PowerMill is performed using the 0.35 $\mu$ m CMOS technology at a supply voltage of 3.3V. The effective channel widths for PMOS and NMOS transistors when unskewed (or when skew value = 1) are 4.5 $\mu$ m and 1.8 $\mu$ m, respectively, for all benchmark circuits. The results are summarized in Table 1.

From Table 1, we observe that the skewed logic implementation significantly reduces the amount of logic duplication required when compared with that required by the Domino logic implementation (32% versus 69%).

That contributes to substantial power savings. Furthermore, clocked gates account for only 23% of the total number of gates in skewed logic. That leads to a large amount reduction in clock power compared to Domino circuits. The average power saving of skewed logic over Domino is 32.6%.

## Acknowledgment

This work was supported in part by NSF (CCR-9984553), and SRC Hewlett-Packard Research Fellowship.

## 5. References

[1] T. Thorp, G. Yee, and C. Sechen, *Design and Synthesis of Monotonic circuits*, International Conference on Computer Design, 1999.

[2] D. Somasekhar, *Power and dynamic noise considerations in high performance CMOS VLSI design*, Ph.D. Thesis, Purdue University, August 1999.

[3] A. Solomatnikov, D. Somasekhar, K. Roy, C. K. Koh, *Skewed CMOS: Noise-Immune High-Performance Low-Power Static Circuit Family*, International Conference on Computer Design, 2000, pp. 241-246.

[4] S. M. Reddy, *Complete Test Sets for Logic Functions*, IEEE Transaction on computers, C-22(11): 1016-1020, November 1973.

[5] M. R. Prasad, D. Kirkpatrick, and R. K. Brayton, *Domino Logic Synthesis and Technology Mapping*, International Workshop on Logic Synthesis, 1997.

[6] R. Puri, A. Bjorksten, and T. E. Rosser, *Logic Optimizaion by Output Phase Assignment in Dynamic Logic Synthesis*, ICCAD, 1996.

[7] M. Zhao and S. S. Sapatnekar, *Technology Mapping for Domino Logic*, Proceedings of the IEEE International Conference on Computer-Aided Design, pp. 248 - 251, 1998.

[8] M. Zhao and S. S. Sapatnekar, *Dual-Monotonic Domino Gate Mapping and Optimal Output Phase Assignment of Domino Logic*, Proceedings of the IEEE International Symposium on Circuits and Systems, 2000.

[9] L. Stockmeyer, *Optimal orientations of cells in slicing floorplan designs*. Information and Control, 57(2/3):91-101, May/June 1983.

Table 1. Total power comparison between SCSL circuits and Domino circuits.

Circuit	Type	# gates	# transistors	Total width ( $\mu\text{m}$ )	# clocked gates	Circuit power (mW)	% reduction	Clock power (mW)	% reduction	Total power (mW)	% reduction
C432	SCSL	387	1 489	4 526	73	33.45	30.7	4.76	57.7	38.21	35.8
	Domino	510	3 096	9 245	510	48.28		11.25		59.53	
C499	SCSL	895	4 254	13 386	179	79.62	30.3	19.81	45.3	99.43	33.9
	Domino	1 097	6 846	20 221	1 097	114.21		36.23		150.44	
C880	SCSL	583	2 669	8 132	141	86.35	21.7	16.45	39.2	102.80	25.1
	Domino	695	4 183	12 533	695	110.24		27.06		137.30	
C1355	SCSL	842	4 578	15 021	203	89.42	28.5	20.94	41.1	110.36	31.3
	Domino	1 126	6 960	20 635	1 126	125.01		35.54		160.55	
C1908	SCSL	1 190	5 988	18 965	235	97.54	24.5	20.28	37.3	117.82	27.1
	Domino	1 416	8 934	26 276	1 416	129.16		32.37		161.53	
C2670	SCSL	1 505	8 227	25 963	486	165.73	31.4	34.90	45.0	200.63	34.3
	Domino	1 990	12 231	36 344	1 990	241.72		63.46		305.18	
C3540	SCSL	2 177	10 892	34 221	506	164.20	34.7	31.38	47.6	195.58	37.2
	Domino	2 787	17 903	52 293	2 787	251.36		59.86		311.22	
C5315	SCSL	3 229	14 586	49 288	640	229.40	29.9	41.75	46.4	271.15	33.1
	Domino	4 114	25 926	76 287	4 114	327.21		77.88		405.09	
C7552	SCSL	4 982	20 025	62 416	705	399.88	32.3	72.32	47.7	472.20	35.2
	Domino	6 425	41 435	120 838	6 425	590.50		138.24		728.74	
<b>Average</b>							<b>29.3</b>		<b>45.3</b>		<b>32.6</b>