

Synthesis of Skewed Logic Circuits

AIQUN CAO

Synopsys, Inc.

NARAN SIRISANTANA

Intel Corporation

and

CHENG-KOK KOH and KAUSHIK ROY

Purdue University

Skewed logic circuits belong to a noise-tolerant high-performance static circuit family. Skewed logic circuits can achieve performance comparable to that of Domino logic circuits but with much lower power consumption. Two factors contribute to the reduction in power. First, by exploiting the static nature of skewed logic circuits, we can alleviate the cost of logic duplication which is typically required to overcome the logic reconvergence problem in both Domino logic and skewed logic circuits. Second, a selective clocking scheme can be applied to a skewed logic circuit to reduce the clock load and hence, clock power. In this article, we propose a two-step synthesis scheme of skewed logic circuits. In the first step, an integer linear programming-based approach is presented to overcome the logic reconvergence problem in skewed logic circuits with minimal logic duplication cost. In the second step, a dynamic programming-based heuristic is applied to achieve an optimal selective clocking scheme. Experimental results show that the average power saving of skewed logic circuits over Domino logic circuits is 41.1%.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Automatic synthesis, Optimization*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Advanced technologies, Standard cells, VLSI*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Skewed logic, synthesis, optimization, power

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

Preliminary versions of this article appeared in the Proceedings of the International Symposium on Low Power Electronics and Design, 2001 [Sirisantana et al. 2001], the Proceedings of the International Symposium on Quality Electronic Design, 2002 [Cao et al. 2002], and the Proceedings of Asia South Pacific Design Automation Conference, 2003 [Cao et al. 2003].

Authors' addresses: A. Cao, Synopsys, Inc., 700 East Middlefield Rd., Mountain View, CA 94043; N. Sirisantana, Intel Corporation, M/S RA3-256, 2501 NW 229th Ave., Hillsboro, OR 97124; C.-K. Koh, and K. Roy, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.

© 2005 ACM 1084-4309/05/0400-0205 \$5.00

1. INTRODUCTION

Domino logic has been a popular choice for designs that demand very high performance [Krambeck et al. 1982; Goncalves and Man 1983]. However, two inherent drawbacks of Domino logic limit its further use as technology continues to scale. First, the noise margin of Domino logic circuits is very small because it is dependent on the threshold voltages of transistors. That makes Domino logic circuits extremely susceptible to failures due to threshold voltage variation and noise injection. Moreover, the threshold voltage scaling increases subthreshold current exponentially, making Domino logic even more susceptible to noise. In other words, *Domino logic circuits do not scale well with the technology*. Second, Domino logic dissipates much more power than static circuits; it does not suit low power operation well.

To overcome the drawbacks of Domino logic, a new noise-immune high-performance logic style, called skewed logic [Somasekhar 1999; Solomatnikov et al. 2000] or Monotonic Static (MS) CMOS logic [Thorp et al. 1999b; Thorp et al. 1999a], has been proposed. Skewed logic circuits are fully complementary static CMOS logic, with the size of pull-down network (PDN) decreased and that of pull-up network (PUN) increased, or vice versa, for fast low-to-high or fast high-to-low transition, respectively. Sizing the PDN and PUN to favor one-direction transition is referred to as *skewing* [Somasekhar 1999; Solomatnikov et al. 2000].

Similar to Domino logic, skewed logic is operated in precharging-evaluation fashion for high performance. Fast transition is used for evaluation, while slow transition is used for precharging. Thus, skewed logic is comparable in speed to Domino logic. At the same time, skewed logic has better noise immunity than Domino logic due to its static nature. However, skewed logic suffers from the same logic reconvergence problem that plagues Domino logic. In Thorp et al. [1999a], the synthesis of skewed logic is performed by employing logic duplication to transform the network into a unate representation as in Domino logic synthesis. Moreover, each gate is clocked for precharging-evaluation operations as in Domino logic. Consequently, the synthesized skewed logic in [Thorp et al. 1999a] does not have a power advantage over Domino logic.

In this article, we show that by exploiting the static nature of skewed logic circuits, we can achieve a substantial reduction in the amount of logic that must be duplicated for overcoming the logic reconvergence problem. Specifically, we propose a two-step synthesis scheme for skewed logic circuits with the objective of minimizing total power consumption. In the first step, an integer linear programming-based approach is presented to overcome the logic reconvergence problem in skewed logic circuits with minimal logic duplication cost. In the second step, a dynamic programming-based heuristic is applied to achieve an optimal selective clocking scheme to reduce the clocking power. Experimental results show that the power saving of skewed logic circuits over Domino logic circuits is 41.1% on average.

The rest of the article is organized as follows. Skewed logic circuits are described in Section 2. Section 3 formulates the synthesis problem of skewed logic circuits. The two-step synthesis scheme is presented in Section 4

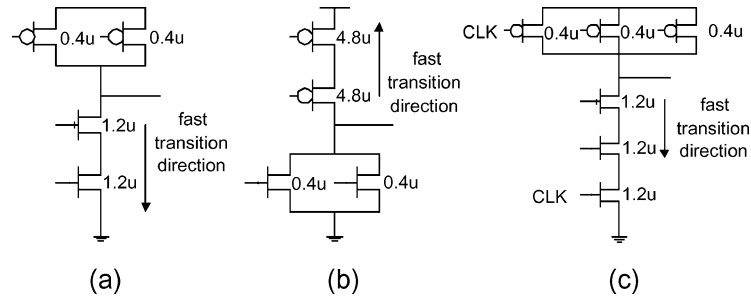


Fig. 1. Two-input (a) NAND, (b) NOR, and (c) clocked NAND skewed logic gates.

and 5. Section 6 provides experimental results and Section 7 concludes the article.

2. SKEWED LOGIC CIRCUITS

2.1 Topology

A skewed logic gate has the same circuit topology as a classical static CMOS gate, except that the sizes of PMOS and NMOS transistors are properly adjusted such that the gate favors a particular transition direction. Changing the ratio between PMOS and NMOS transistors of the gate from its original value $R = W_p/W_n$ (W_p and W_n being the channel width of PMOS and NMOS transistors, respectively) to a new value $R' = W'_p/W'_n$ to favor one transition direction is referred to as *skewing*. R'/R is referred to as *skew value* of the gate. Note that *the overall gate size (transistor width) and gate capacitance are invariant under skewing* [Somasekhar 1999]. 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 Figure 1(a) and (b), respectively.

2.2 Operation

Operating as precharging-evaluation circuitry, skewed logic circuits can achieve performance comparable to that of Domino logic circuits. A skewed gate is precharged to the logic value that allows only fast transitions 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 Figure 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 that the precharging of skewed logic gates between two clocked gates does not exceed the precharge phase of the clock period. Thorp et al. [1999a] considered clocking of all skewed gates, whereas Sirisantana et al. [2001] and Solomatnikov et al. [2000] explored the second option by clocking skewed gates selectively (see Figure 2). Precharging through propagation allows a smaller number of clocked gates. It leads to a lower clock load and hence potentially lower clock power consumption than Domino circuits. Determining the selective clocking of skewed gates for low

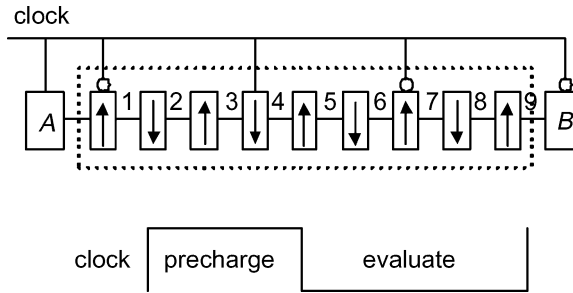


Fig. 2. A chain of skewed logic gates between two latches. The skew direction is indicated by the arrow.

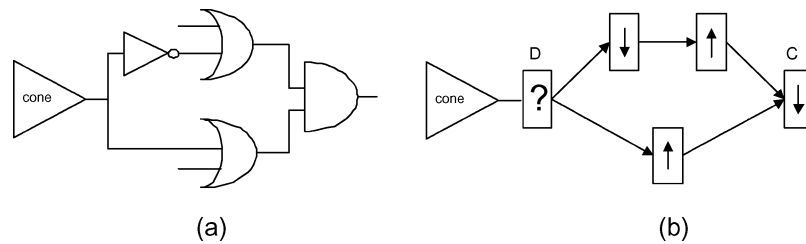


Fig. 3. Logic reconvergence problem in (a) a Domino logic circuit, (b) a skewed logic circuit.

power operation is an integral step of the synthesis of skewed logic circuits. We will present a dynamic-programming based technique that determines the appropriate skew values and the optimal clocking scheme in Section 5.

2.3 Logic Reconvergence Problem

To ensure fast evaluation, alternating skew directions should be assigned to successive logic gates, that is, skewed-down gates are followed by skewed-up gates and vice versa. Consequently, skewed logic encounters the same problem as Domino logic: logic reconvergence. The conventional synthesis paradigm for Domino logic requires that a unate representation from a logic network be generated. That is accomplished by pushing inverters to the primary inputs. With the presence of reconvergent paths in the logic network, an inverter may be trapped in the reconvergent paths as shown in Figure 3(a). Similarly, the reconvergent paths in a skewed logic network can render the assignment of alternating skew directions impossible, as shown in Figure 3(b). For the synthesis of Domino logic, this problem can be overcome by duplicating the fan-in cone of the reconvergent paths [Prasad et al. 1997], i.e., transforming a binate logic network to a unate one. Thorp et al. [1999a] applied this method for synthesizing skewed logic circuits. This approach will at most double the size of the circuit [Reddy 1973]. The duplication cost can be reduced by proper output phase assignment [Puri et al. 1996; Zhao and Sapatnekar 2000] but the achievable reduction is limited. Both the logic duplication technique and the output phase assignment technique for Domino logic synthesis can be applied to the synthesis of skewed logic.

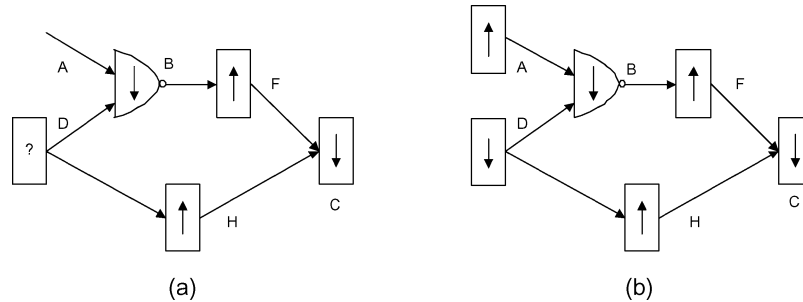


Fig. 4. (a) Reconvergent paths with a NAND gate. (b) Nonalternating skew directions.

Sirisantana et al. [2001] made use of pass transistors to alleviate the logic reconvergence problem. However, it did not consider the glitch caused by the pass transistors which could affect the performance of circuits. In Cao et al. [2002], that was corrected and the static nature of skewed logic was exploited to reduce the logic duplication penalty. The key is that, under certain conditions, assigning nonalternating skew directions to skewed logic gates does not impede the performance of circuit. In other words, the reconvergence problem can be overcome by assigning nonalternating skew directions. In Cao et al. [2002], each pair of reconvergent paths was solved independently; such an approach restricted the solution space since there might be many reconvergent paths overlapping with each other in real circuits. The solution space is further reduced by the restriction that only one gate is permitted to have nonalternating skew direction in each pair of reconvergent paths. In this article, we eliminate those two restrictions to achieve even more improvement on logic duplication and thus power dissipation. We consider the overlapping of reconvergent paths and allow more than one gate to have nonalternating skew direction for each pair of reconvergent paths to overcome the logic reconvergence problem.

2.4 Nonalternating Skew Directions

A careful analysis of the static skewed logic circuit reveals that we can make use of the NAND gate *B* in Figure 4(a) to avoid logic duplication as follows [Cao et al. 2002; Kim et al. 2000]: we make both gate *B* and gate *D* skewed-down gates. The other fan-in to gate *B*, gate *A*, and the fan-out of gate *B*, gate *F*, are both skewed up, as shown in Figure 4(b). The skew directions of gates *D* and *B* are nonalternating.

Although gates *D* and *B* are skewed in the same direction, gate *B* still properly precharges (to logic 1) because of gate *A*, which precharges to logic 0 as shown in Figure 5 (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 nonalternating skew directions. As skewed logic achieves its performance by allowing only fast transitions 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 gate *B*, and that will affect the performance of the circuit.

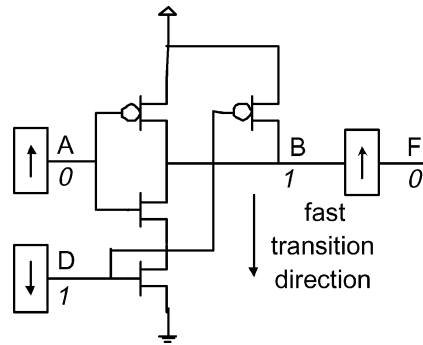


Fig. 5. Transition of a NAND gate.

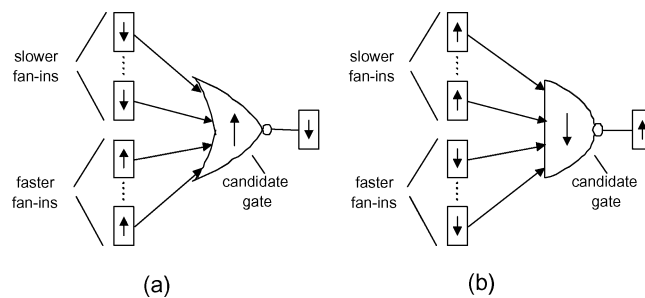


Fig. 6. Skew directions for (a) candidate NOR gate, (b) candidate NAND gate.

Note that for any NAND gate that precedes a skewed-up gate, we can always skew down the NAND gate and its faster fan-in gate as in Figure 5 so that only fast transitions occur during the evaluation phase. When the NAND gate has more than two fan-in gates, nonalternating skew directions can be assigned as long as the skewed-down fan-in gates switch earlier than the fan-in gates that are skewed up if they do switch. A similar analysis will reveal that the skew direction assignment shown in Figure 6(a) with the NOR gate as the middle gate also achieves the same effect. Similar configurations can also be obtained for AND and OR gates. We refer to the gates that allow nonalternating skew directions as candidate gates.

The key limitation here is that when nonalternating skew directions are considered, the only skew direction feasible for the fan-out gate of a NAND (NOR) gate is skewed up (down). 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 the NOR gate have to precharge to logic 0.

It is obvious that the nonalternating skew direction scheme cannot eliminate all of the logic reconvergence problem in any circuit due to such a limitation. Logic duplication is still needed for the unresolved logic reconvergence problem. In Section 4, we formulate the problem of skew direction assignment for the minimization of logic duplication cost as an integer linear program.

3. SYNTHESIS PROBLEM OF SKEWED LOGIC CIRCUIT

The skewed logic circuit synthesis problem can be stated as follows: *Given a mapped logic network, implement it as skewed logic by determining the skew direction and skew value of each gate, and an optimal clocking scheme, such that the timing constraint is satisfied and the total power consumption is minimized.* Here, we assume that the generic cell library would contain NAND, NOR, and inverters for mapping the logic network.

Unfortunately, it is extremely difficult to determine the skew direction and value of each gate at the same time. The reason is that the network topology may change due to logic duplication when we assign skew directions. Without a fixed network topology, it is difficult to find the skew value for each gate as the loads of the gates are not fixed. In this article, we apply a two-step approach to the synthesis of skewed logic, that is, the synthesis problem is divided into two subproblems: *skew direction assignment* followed by *skew value and clocking optimization*. The problem of skew direction assignment is that of determining the skew direction of each gate such that the logic reconvergence problem is overcome with minimal logic duplication. The problem of skew value and clocking optimization is that of determining the skew value of each gate and an optimal clocking scheme such that the given timing constraint is satisfied, and the total circuit power dissipation is minimized. Although the two-step approach may degrade the overall quality of the solution, experimental results indicate that significant power reductions can be achieved. We present the details of the solutions to the two subproblems in the next two sections.

4. SKEW DIRECTION ASSIGNMENT WITH INTEGER LINEAR PROGRAM (ILP)-BASED APPROACH

We state the problem of skew direction assignment as follows: *Given a logic network, determine the skew direction of each gate by using both alternating and nonalternating schemes such that the amount of logic duplication is minimized.* Note that the output phase assignment problem in Puri et al. [1996] and Zhao and Sapatnekar [2000], which is NP-hard, is a special case of this problem.

In this section, we transform the alternating and nonalternating schemes into integer linear constraints among the edges of the network so that an edge-based ILP formulation is generated. The ILP is solved with the objective of minimizing the logic duplication cost under those constraints. A simplification technique to reduce the number of constraints is presented. Afterwards, we present heuristics to solve the ILP efficiently. Methods to further reduce the logic duplication cost are discussed.

4.1 Definitions

Given any pair of 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. For simplicity, we say that a pair of reconvergent paths forms a *reconvergent cycle*, or simply a *cycle*, in the rest of this article, although there are no cyclic paths in the directed acyclic graph defined by the

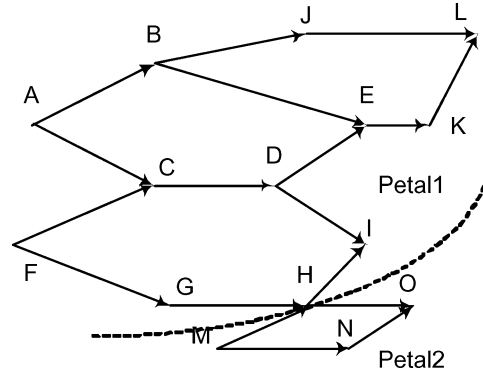


Fig. 7. Two petals.

logic network. We denote the cycle with divergent node A and convergent node Z as C_{AZ} . We denote the set of edges in cycle C as $E(C)$.

Given two cycles C and C' , if $E(C) \cap E(C') \neq \emptyset$, we say that C and C' belong to a *petal* [Dey et al. 1990]. Furthermore, given three cycles C , C' , and C'' , if $E(C) \cap E(C') \neq \emptyset$, and $E(C) \cap E(C'') \neq \emptyset$, C , C' , and C'' belong to the same petal. In other words, a petal consists of all the cycles that share edges among them. Figure 7 shows two petals, one formed by many cycles, and the other by a single cycle.

The cycles in Figure 7 also illustrate the fact that some cycles are the union of two or more other cycles (after removing the shared edges in these cycles). For example, the union of C_{AE} and C_{BL} forms the cycle C_{AL} (after removing the shared edge e_{BE}). We call C_{AL} a composite cycle and C_{AE} and C_{BL} simple cycles, in that they are not the result of the union of other cycles.

Given an edge e , we denote the head (destination) and tail (source) of the edge as H_e and T_e , respectively. Edges with the same head are sibling edges. If H_e is a candidate gate, and T_e is not the slowest among all the fan-ins of H_e , edge e is a *candidate edge*.

4.2 An Edge-Based Formulation

Here, we use the assignment of skew directions to gates in a cycle to illustrate the proposed edge-based integer linear program (ILP) approach. Along the two paths of the cycle, there may be more than one candidate gate where nonalternating skew directions can be applied.

In our formulation, candidate edges are (binary) variables in the ILP. Edge e is assigned value 0 if H_e is in the same skew direction as T_e , and value 1 otherwise. All noncandidate edges implicitly have a value of 1. P is defined as the *parity* of an edge or a path. For an edge, its parity is the same as the value it is assigned; for a path, its parity is 0 if the sum of the values of all the edges along the path is an even number, and 1 if the sum is an odd number.

There are three types of constraints that we have to capture in the edge-based ILP formulation.

(i) *Gate constraints* originate from the definition of candidate gates: the nonalternating fan-ins of a candidate gate must be faster than the alternating

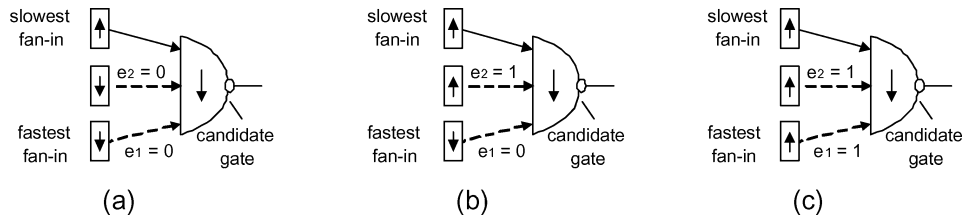


Fig. 8. Three feasible assignments of edge variables.

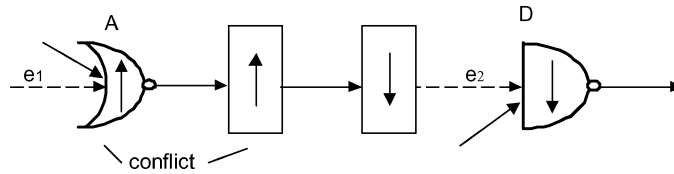


Fig. 9. Path constraints.

fan-ins. Therefore, when a candidate edge is determined to be of value 1, all the slower sibling candidate edges must also have value 1. Figure 8 shows an example of the gate constraint where e_1 and e_2 are two candidate edges feeding into a candidate gate, and e_1 is faster than e_2 . Three possible cases for the value of two candidate edges e_1 and e_2 are enumerated in Figure 8(a)–(c). A single inequality can capture the gate constraint between e_1 and e_2 as follows:

$$e_2 - e_1 \geq 0.$$

(ii) *Path constraints* stem from the fact that there is only one nonalternating skew direction for each candidate gate. For example, the only nonalternating skew direction for a candidate NOR gate is to skew up.

Consider the simple example in Figure 9, where e_1 and e_2 are the only two candidate edges (dashed line) along a path. Gate A and D are two candidate gates. Gate A (D) can only be skewed-up (skewed-down) for the nonalternating scheme since it is a NOR (NAND) gate. However, if nonalternating skew directions are applied to both of them, that would result in a conflict as shown in Figure 9. In other words, e_1 and e_2 cannot both be assigned 0. That constraint is captured by the following inequality:

$$e_1 + e_2 \geq 1.$$

Consider two successive candidate edges between which there are no other candidate edges. (However, there might be noncandidate edges between the two successive candidate edges.) In general, if the head gates of these two successive candidate edges are of different types (NAND and NOR), the number of noncandidate edges between them must be *odd* in order to avoid conflicts. If the candidate gates are of the same type (NAND or NOR), there should be an *even* number of noncandidate edges between them in order to avoid conflicts.

Unfortunately, not only do conflicts exist between two successive candidate edges, they also exist among all candidate edges along one path. Consider three successive candidate edges, e_1 , e_2 , and e_3 in Figure 10. The two constraints for

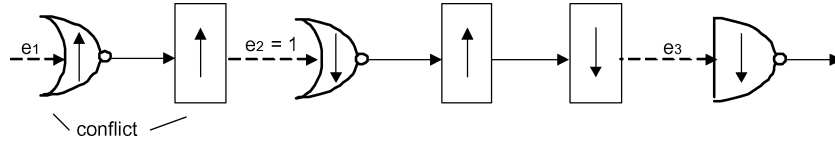


Fig. 10. Conflicts among three successive candidate edges.

e_1 and e_2 , and e_2 and e_3 are:

$$e_1 + e_2 \geq 1, \quad (1)$$

$$e_2 + e_3 \geq 1, \quad (2)$$

respectively. If e_2 is assigned value 0, the constraints among the three candidate edges can be captured by the preceding two constraints. If e_2 is of value 1, however, it acts like a noncandidate edge, and the circuit degenerates to the case in Figure 9. Therefore, we would require a new constraint for e_1 and e_3 :

$$e_1 + e_3 \geq 1 \text{ if } e_2 = 1, \quad (3)$$

which is equivalent to

$$e_1 + e_3 - e_2 \geq 0. \quad (4)$$

So, there are three constraints in total for these three candidate edges. It turns out that the three inequality constraints among e_1 , e_2 , and e_3 , Equations (1), (2), and (4) can be adequately captured by a single inequality constraint as follows:

$$e_1 + e_2 + e_3 \geq 2, \quad (5)$$

since at most one of the three candidate edges can have value 0. Extending to n successive candidate edges along one path, there are $O(n^2)$ number of inequalities among them. However, they can be replaced and captured by one single inequality:

$$e_n + e_{n-1} + \cdots + e_1 \geq n - 1,$$

which implies that only one candidate edge among those n edges can have value 0.

In general, given any three candidate edges (not necessarily successive) with head gates A , B , C (in that logic order), there are four possible scenarios:

- (1) A conflicts with B , B conflicts with C ; hence, A conflicts with C . Here, A conflicts with B implies that A and B cannot both be assigned nonalternating skew directions.
- (2) A does not conflict with B , B does not conflict with C ; hence, A conflicts with C .
- (3) A conflicts with B , B does not conflict with C ; hence, A does not conflict with C .
- (4) A does not conflict with B , B conflicts with C ; hence, A does not conflict with C .

We have shown how the conflicts that arise in the first scenario can be efficiently captured by one single inequality constraint. In fact, given any path

in a simple cycle, one can always partition the candidate edges into at most two groups with edges in a group conflicting with each other but not with edges in the other group. Let j be the number of pair-wise conflicting candidate edges in a group. We further partition the group into k subgroups such that every subgroup contains only successive candidate edges. Based on the simplification technique presented earlier, the conflicts within every subgroup can be captured by one inequality constraint. The number of additional inequality constraints to capture the relations between these k subgroups is $\binom{k}{2} = \frac{k(k-1)}{2}$. We provide the proof in the Appendix. Without the simplification, the number of inequality constraints among these j candidate edges would be $\binom{j}{2}$. The experiments conducted in this article show that k is about a third of j .

(iii) *Cycle constraints* concern candidate edges on two different paths in a cycle. In general, we can generate cycle constraints by treating them as path constraints. We define a path, via the divergent node, between the two candidate edges in the cycle. Now, we can express the cycle constraints as the path constraints, and similar simplification technique applies. Moreover, we have another constraint: the original two paths of the cycle must have the same parity, that is, the number of alternating skew directions along the two paths must satisfy:

$$e_1 + \cdots + e_m + c - e'_1 - \cdots - e'_n - c' = 2i, \quad (6)$$

where

$$c - c' - n \leq 2i \leq c - c' + m, \text{ and } i \text{ is integer.}$$

Here, e_j is the candidate edge along one reconvergent path in a cycle, and e'_k is the candidate edge along the other reconvergent path in the same cycle. Constants c and c' are the numbers of noncandidate edges along the two paths, respectively. i is an integer variable to be solved in the ILP. Equation (6) arises from the logic reconvergence problem that we try to solve, whereas other constraints arise from the limitation of the nonalternating skew direction scheme.

The above three types of constraints are applicable to both simple cycles and composite cycles. Fortunately, we have to consider only simple cycles. In the following, we argue that the reconvergence problems of all composite cycles are solved when those of simple cycles are solved. The reconvergence problem of a cycle is solved if the two paths of the cycle have the same parity. Now we apply induction on the number of simple cycles that a composite cycle contains. We can prove that the two paths of the composite cycle have the same parity if that is true for all the simple cycles that it contains. Take the cycles C_{AE} and C_{BL} in Figure 7, for example. Once the constraints specified by cycles C_{AE} and C_{BL} are satisfied, the reconvergence problems of them are solved directly. In other words, the paths $A - C - D - E$ and $A - B - E$, $B - E - K - L$ and $B - J - L$ have the same parity:

$$\begin{aligned} P_{ACDE} &= P_{ABE}, \\ P_{BEKL} &= P_{BJL}. \end{aligned}$$

For the two paths $A - B - J - L$ and $A - C - D - E - K - L$ of the composite cycle C_{AL} , we have:

$$P_{ABJL} = P_{ABE} + P_{BJL} - P_{BE} = P_{ACDE} + P_{BEKL} - P_{BE} = P_{ACDEKL},$$

which means that the reconvergence problem of the composite cycle $_{AL}$ is solved as well.

4.3 Skew Direction Assignment

First, we consider the skew direction assignment within a petal. Clearly, we can formulate the skew direction assignment problem as an ILP in which the variables are the candidate edges, and the constraints are the gate, path, and cycle constraints formed by simple cycles within the petal.

However, there is one compelling reason to abandon the preceding suggested approach. The suggested approach will produce a “YES/NO” answer, of which the “NO” answer is of no value as it does not provide a partial result to our synthesis problem. Rather, we would like to identify in a petal as many cycles as possible whose reconvergence problems can be resolved by nonalternating skew direction assignment. The other cycles would have their fan-in cones duplicated to overcome the logic reconvergence problems. To achieve this, we use the following heuristic that incrementally solves a one-cycle ILP formulation at a time. Suppose we already have a feasible solution for n cycles in the petal. We use the assignment specified in the feasible solution to determine the constraints for the $(n+1)^{th}$ cycle. For example, if the n -cycle solution assigns value 1 to a candidate edge in the $(n+1)^{th}$ cycle then this candidate edge becomes a noncandidate edge in the ILP formulation for the $(n+1)^{th}$ cycle. Otherwise, it imposes constraints on the problem of skew direction assignment of the $(n+1)^{th}$ cycle even though it is no longer a variable.

If the ILP formulation for the $(n+1)^{th}$ cycle produces a feasible solution, we proceed to the $(n+2)^{th}$ cycle. Otherwise, we solve a large ILP that considers all $(n+1)$ cycles simultaneously. Here, the constraints of the large ILP are the union of the constraints of those $(n+1)$ cycles. If the ILP produces a feasible solution, we proceed to the $(n+2)^{th}$ cycle. Otherwise, we duplicate the fan-in cone of the $(n+1)^{th}$ cycle and proceed to the $(n+2)^{th}$ cycle. Note that when the fan-in cone of the $(n+1)^{th}$ cycle is duplicated, all reconvergence problems in the fan-in cone are also resolved.

In the incremental ILP formulation, we optimize the following objective in each ILP:

$$\min \sum_i c_i \times e_i,$$

where e_i is the candidate edge, and c_i is the number of different simple cycles sharing the candidate edge e_i . The idea here is that we want to reduce the number of constraints in the ILP of later cycles. The objective function maximizes the number of candidate edges assigned with value 1 (i.e., alternating skew directions), especially for candidate edges shared by many cycles.

Now, we consider the assignment of skew directions to the gates not in any petals. Some of these gates may lie along the path connecting two petals, as

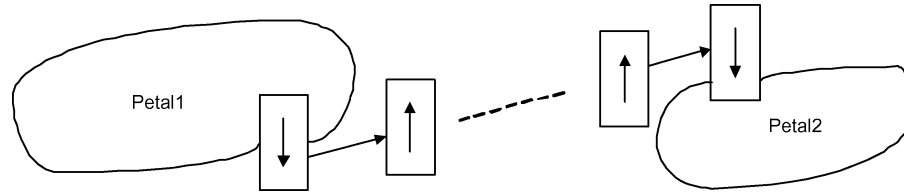


Fig. 11. Path connecting 2 petals.

shown in Figure 11. If alternating skew direction assignment along such a path results in conflicts, it is easy to find a candidate edge along this path to which we can apply nonalternating skew directions. If such a candidate edge does not exist, it is necessary to identify appropriate fan-in cones (with the least overall duplication cost) to be duplicated.

4.4 Logic Duplication

As described in the previous section, logic duplication is still needed when the ILP-based approach cannot solve all of the logic reconvergence problem or no candidate edges exist along a path connecting two petals. Normally, when a node needs to be duplicated, the nodes within its whole fan-in cone need to be duplicated. Figure 12(a) shows a divergent node D and its fan-in cone to be duplicated. The structure after duplication is shown in Figure 12(b).

However, we can still reduce the duplication cost even when duplication of the fan-in cone is unavoidable. The reason is that the nonalternating skew direction scheme can be applied to the duplicated fan-in cone, too. In Figure 12(c), for example, we do not have to duplicate node W and its fan-in cone when node Z' (duplicate of node Z) is a candidate gate, that is, node Z' allows nonalternating fan-in.

When a fan-in cone of node Y is duplicated, the duplication is performed in the direction opposite to that of the original DAG until the primary inputs are reached. The duplication is performed with a breadth-first traversal that starts from node Y . The duplicate is assigned to a skew direction opposite to the original, both of which are determined according to the skew directions of their own fan-out gates. Once a candidate gate X is met during the duplication process, the nonalternating scheme is applied, and no duplication is required for the candidate gate X and its fan-in cone. The skew directions of the gates in the fan-in cone of X can be determined accordingly.

5. SELECTIVE CLOCKING SCHEME

Now that we have determined the skew direction of each gate, the next step is to determine the skew value, that is, 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. However, the overall gate size and gate capacitance are identical for different implementations of a logic gate. Different skew values result in different precharge and evaluation delays, denoted respectively by t_{pc} and t_{ev} , and thus, different clocking schemes. We apply a dynamic programming-based heuristic algorithm to find the clocking solution. The goal is to find the skew

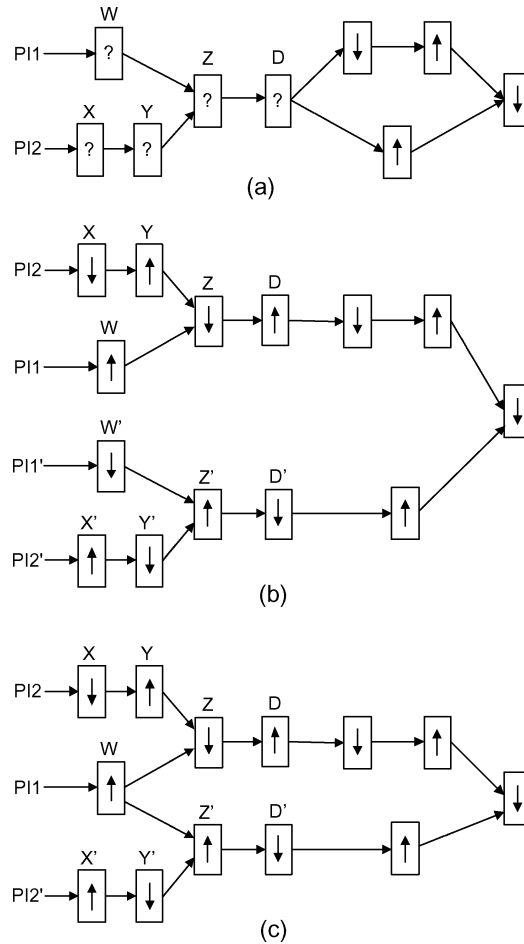


Fig. 12. Applying the nonalternating scheme when duplicating.

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 a primary output as a root. Second, we consider the case when the netlist is a directed acyclic graph. Finally, further reduction of clocked gates of the selective clocking scheme is discussed.

5.1 Netlist With a Tree Structure

With a tree structure, it is natural to apply a dynamic programming-based approach to assign skew values to skewed gates and to determine the location of clocked gates. We apply the dynamic programming approach in a bottom-up fashion, that is, from the first-level logic gates, which should be clocked (as shown in Figure 13).

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

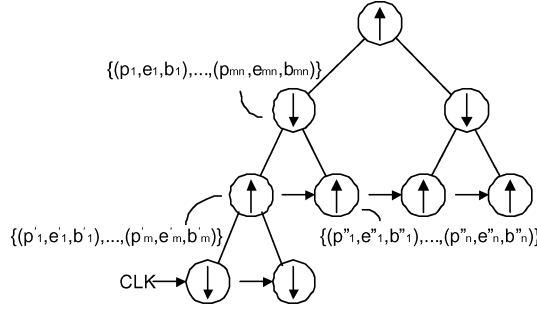


Fig. 13. Tree with leaf nodes connected to clock.

are in the fan-in cone of N (including N) such that there exist no clocked 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 the 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 the 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:

$$(\min(p'_i, p''_j) - t_{pc}, \min(e'_i, e''_j) - t_{ev}, 0), \quad \text{and} \\ (T_{pc} - t'_{pc}, \min(e'_i, e''_j) - t'_{ev}, 1).$$

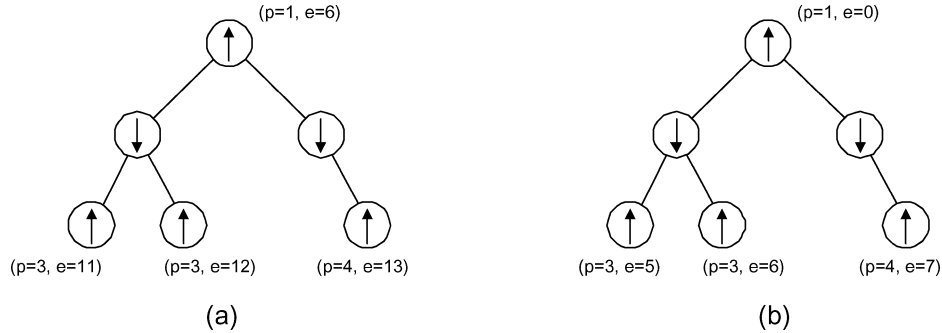


Fig. 14. Deriving precharge and evaluation slacks of shared nodes.

There are in total $O(m * n * s)$ combinations for N , where s is the total number of skew value choices for N , m and n are the number of triples of the child nodes of N . As a result, there are an 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 run-time of the algorithm can also be reduced accordingly [Stockmeyer 1983].

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.

5.2 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 a clocking scheme; for example, the sizing of Domino gates [Zhao and Sapatnekar 1998]. 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 article, 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. The leaf nodes of the tree are all primary inputs. Therefore, the evaluation time of the tree is set to be the given evaluation period T_{ev} , and the dynamic programming-based approach described in the previous section is applied. We remove the nodes in the tree from the netlist and proceed to extract trees from the remaining netlist in a similar fashion. If the root of the extracted tree is a primary output and the leaf nodes are all primary inputs, the evaluation time of the tree is also set to be T_{ev} . If the root and/or leaf nodes of the extracted tree are shared by previously extracted trees, we derive the new precharge and evaluation slacks of those nodes from the results of the previously extracted trees. For example, Figure 14(a) shows the precharge slack p and evaluation slack e of shared nodes

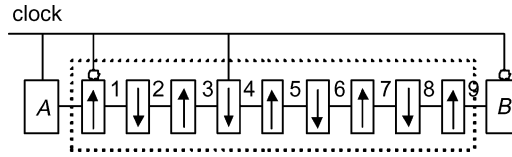


Fig. 15. Eliminating the last stage clock.

according to previously extracted trees, while the new p and e of those nodes are shown in Figure 14(b). From Figure 14, we can see that the precharge slacks are kept the same, whereas we have to deduct the evaluation slack of the root from the evaluation slacks of those nodes. The dynamic programming-based approach is applied to the extracted tree and the procedures repeat until all nodes have been considered.

5.3 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 the clock. After obtaining the clocking scheme as outlined in the previous section, we can always eliminate the clock connected to the gates in the *CFI* of the primary outputs.

As Figure 15 shows, the clock connected to gate 7 in Figure 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.

6. EXPERIMENTAL RESULTS

We have implemented the two-step synthesis scheme presented in Sections 4 and 5 in C++ language. We use a standard ILP package “lp_solve” to solve the ILPs.

Ten ISCAS benchmark circuits are used for experiments running on Sun Ultra10, 440 MHz, 512M RAM. The solving of ILPs is dominating in run-time. For each benchmark circuit, the total number of petals, the total number of cycles, the average number of cycles in a petal, the maximum number of cycles in a petal, the total number of integer variables, the average number of integer variables in a cycle, and the maximum number of integer variables in a cycle are listed in Table I. Using the simplification technique presented in Section 4.2, average reduction in the number of inequality constraints and the run-time are 2.8x and 5x, respectively (See Table I).

For comparison, the ISCAS benchmark circuits are implemented as skewed logic circuits and Domino circuits. For skewed logic circuits, the library contains inverters, up to 4-input NAND and NOR gates, with six different skew values each (values 3, 4, and 5 for both skewing up and skewing down). By using SIS [Sentovich et al. 1992], the benchmark circuits are mapped to the generic library gates (NAND, NOR, and inverters) without determining skew directions or values first. After that, the two-step approach is applied. For Domino circuits, we restrict the library of gates to contain only inverters, 2-input NAND, and

Table I. Comparison of ILP Solving Run-Time Before and After the Simplification of Constraints

Circuit	C432	C499	C880	C1355	C1908	C2670	C3540	C5315	C6288	C7552	Avg
# petals	4	20	18	33	21	25	19	33	16	27	–
# cycles	46	134	151	315	445	489	977	692	1272	1163	–
avg # cycles per petal	12	7	8	10	21	20	51	21	80	43	–
max # cycles in a petal	27	69	62	98	134	110	292	248	411	355	–
# integer variables	192	358	311	492	620	667	1315	949	1780	1911	–
avg # integer variables per cycle	6	7	6	7	7	6	9	7	8	8	–
max # integer variables in a cycle	11	21	16	20	27	29	59	34	55	68	–
# constraints (before simplification)	1084	3462	2007	4146	7539	7927	30402	14276	29751	38874	–
ILP time(min) (before simplification)	16	52	48	92	180	198	1154	496	1255	1304	–
# constraints (after simplification)	533	1625	1023	1792	2676	2354	8222	4678	14292	9425	–
ILP time(min) (after simplification)	4	15	18	31	49	40	145	88	201	168	–
Reduction of # constraints	2x	2.1x	2x	2.3x	2.8x	3.4x	3.7x	3.1x	2.1x	4.1x	2.8x
Reduction of time	4x	3.5x	2.7x	3x	3.7x	5x	8x	5.6x	6.2x	7.8x	5x

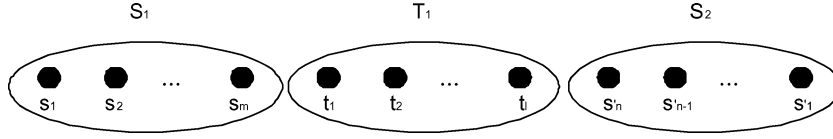
up to 6-input NOR gates, all with minimum sizes. Note that logic duplication is used to resolve the issue of logic reconvergence. We synthesize the Domino logic circuits first and obtain the critical delay of each circuit. We use that as the delay constraint when the dynamic programming-based approach is used to determine the skew values of the corresponding skewed logic circuit. Therefore, the delays of the Domino logic circuit and the skewed logic circuit of each benchmark are almost identical.

All experiments use 0.35 μm CMOS technology at a supply voltage of 3.3 V. The effective channel widths for PMOS and NMOS transistors when unskewed (or when skew value = 1) are 4.5 μm and 1.8 μm , respectively. Power and delay for each circuit are obtained by using PowerMill and PathMill, respectively. The results are summarized in Table II. In Table II, “SL” stands for skewed logic circuit synthesized using our method and “DL” for Domino logic circuit.

From Table II, we observe that our skewed logic implementation significantly reduces the amount of logic duplication required when compared with that required by the Domino logic implementation (18.6% versus 68.0%). This contributes to substantial power savings. The average power saving of skewed logic over Domino is 41.1%. (We could have used the methods in Puri et al. [1996] and Zhao and Sapatnekar [2000] to reduce the duplication cost for Domino logic. Puri et al. [1996] and Zhao and Sapatnekar [2000] reported about 15% reduction in the duplication cost. (If the reduction rate were applicable to the benchmark circuits used in this experiment, the average duplication cost of Domino logic would reduce to about 55%).

Table II. Comparison Among Skewed Logic Synthesized Using Our Method (SL), Domino Logic (DL), Static CMOS (SC), and Skewed Logic Synthesized Using Method in Thorp et al. [1999a]

Circuit	Type	# Gates	% Logic Duplication	Circuit Power (mW)	Clock Power (mW)	Total Power (mW)	Delay (ns)	Area ($\times 1000 \mu m^2$)
C432	SL	336	20.0	18.11	8.04	26.15	2.1	37.248
	DL	510	82.1	38.27	11.97	50.24	2.1	51.014
	SC	280	–	13.82	–	13.82	3.6	29.773
	[Thorp et al. 1999a]	498	78	40.44	18.62	59.06	2.0	57.365
C499	SL	713	14.8	49.91	18.31	68.22	3.5	69.506
	DL	1097	76.7	102.28	33.96	136.24	3.6	90.384
	SC	621	–	44.08	–	44.08	5.3	56.014
	[Thorp et al. 1999a]	1066	71.6	98.06	40.42	138.48	3.3	93.116
C880	SL	488	6.3	50.28	20.27	70.55	3.3	51.519
	DL	695	46.3	89.40	28.28	117.68	3.3	65.839
	SC	459	–	46.21	–	46.21	5.9	46.355
	[Thorp et al. 1999a]	686	49.4	90.25	42.02	132.27	3.3	71.195
C1355	SL	799	21.1	63.54	22.65	86.19	3.5	80.606
	DL	1126	70.6	101.11	33.89	135.00	3.7	98.228
	SC	660	–	52.99	–	52.99	6.5	63.480
	[Thorp et al. 1999a]	1101	66.9	97.87	49.33	147.20	3.4	110.801
C1908	SL	898	10.5	64.61	19.11	83.72	3.8	78.014
	DL	1416	69.0	109.33	32.05	141.38	3.7	105.860
	SC	812	–	57.64	–	57.64	6.2	67.128
	[Thorp et al. 1999a]	1428	75.7	120.08	46.66	166.74	3.8	135.689
C2670	SL	1385	16.3	89.05	36.37	125.42	4.5	89.533
	DL	1990	46.6	184.73	56.18	240.91	4.7	102.265
	SC	1191	–	75.34	–	75.34	7.8	74.708
	[Thorp et al. 1999a]	1949	63.7	160.04	80.31	240.35	4.2	118.330
C3540	SL	1794	14.2	119.90	41.24	161.14	4.8	129.124
	DL	2787	77.4	217.49	66.65	284.14	4.8	179.346
	SC	1571	–	102.92	–	102.92	8.7	107.785
	[Thorp et al. 1999a]	2840	80.8	202.15	93.38	295.53	4.7	177.468
C5315	SL	3069	10.9	149.01	56.26	205.27	5.1	187.073
	DL	4114	48.6	241.73	78.32	320.05	5.2	224.165
	SC	2767	–	131.75	–	131.75	9.0	161.277
	[Thorp et al. 1999a]	4088	47.7	242.30	108.53	350.83	4.9	234.220
C6288	SL	4257	58.4	118.76	41.84	160.60	8.3	370.057
	DL	5159	91.9	155.16	58.71	213.87	8.4	413.458
	SC	2688	–	88.66	–	88.66	13.9	228.429
	[Thorp et al. 1999a]	5025	86.9	166.87	99.55	266.42	8.0	470.513
C7552	SL	4288	13.7	243.84	110.47	354.31	5.8	231.741
	DL	6425	70.4	460.65	168.95	629.60	5.6	313.534
	SC	3772	–	189.09	–	189.09	8.8	194.742
	[Thorp et al. 1999a]	6388	69.4	438.35	237.71	676.06	5.8	360.088
Average	SL	1803	18.6	96.70	37.46	134.16	4.47	132.442
	DL	2532	68.0	170.02	56.90	226.91	4.51	164.409
	SC	1482	–	80.25	–	80.25	7.57	102.969
	[Thorp et al. 1999a]	2507	69.0	165.64	81.65	247.29	4.34	182.879

Fig. 16. Subgroups S_1 , S_2 and T_1 .

In order to compare our work with previous work on skewed logic synthesis, we also implement the method in Thorp et al. [1999a]. In Thorp et al. [1999a], logic duplication is used to resolve the issue of logic reconvergence. Moreover, every gate is clocked. When compared with the results of the method in Thorp et al. [1999a], the logic duplication of our method is 18.6% versus 69.0%, and the average power saving over Thorp et al. [1999a] is 45.7%. This is because the method in Thorp et al. [1999a] has similar duplication cost to Domino logic and even more clock power than Domino logic. However, its delay is slightly better since the gates on the critical path can have the largest available skew values due to the fact that each gate is clocked. We also synthesize the Static CMOS circuit for each benchmark with $0.35 \mu\text{m}$ CMOS technology by using Silicon Ensemble. The results for Static CMOS circuits are also included in Table II. In comparison, Static CMOS circuits have the smallest area and power dissipation, but the largest delay.

7. CONCLUSION

In this article, we propose a two-step synthesis scheme for skewed logic circuits. An ILP-based approach overcomes the logic reconvergence problem in skewed logic circuits with minimal logic duplication cost. A dynamic programming-based selective clocking scheme reduces the clock load. These two factors contribute to produce the reduction in total power consumption.

APPENDIX

THEOREM. *Suppose a group of pair-wise conflicting candidate edges S can be divided into two subgroups, S_1 , and S_2 , with each subgroup consisting of successive candidate edges only:*

$$S_1 = \{s_i \mid 1 \leq i \leq m\},$$

$$S_2 = \{s'_j \mid 1 \leq j \leq n\}.$$

The candidate edges between subgroup S_1 and S_2 form subgroup T_1 (Figure 16) of another group of pair-wise conflicting candidate edges T :

$$T_1 = \{t_k \mid 1 \leq k \leq l\}.$$

There is no conflict between S and T . The conflicts within subgroup S_1 and S_2 can be captured by one inequality constraint, respectively:

$$s_1 + s_2 + \dots + s_m \geq m - 1, \quad (\text{A-1})$$

$$s'_1 + s'_2 + \dots + s'_n \geq n - 1, \quad (\text{A-2})$$

whereas the number of inequalities between two candidate edges within different subgroups is $(m \times n)$. However, these $(m \times n)$ inequalities can be replaced by one

single inequality which captures the same constraints. The single inequality has the form:

$$s_1 + \cdots + s_m + s'_1 + \cdots + s'_n - t_1 - \cdots - t_l \geq m + n - l - 1. \quad (\text{A-3})$$

PROOF We start from the simple case with n equal to 1. The set of inequalities that capture the constraints, denoted by Z , consists of the single inequality constraint within S_1 and the m inequality constraints between S_1 and S_2 :

$$s_1 + \cdots + s_m \geq m - 1, \quad (\text{A-4})$$

$$s_m + s'_1 - t_1 - \cdots - t_l \geq -(l - 1), \quad (\text{A-5})$$

$$s_{m-1} + s'_1 - t_1 - \cdots - t_l - s_m \geq -l, \quad (\text{A-6})$$

$$\vdots \quad (\text{A-7})$$

$$s_1 + s'_1 - t_1 - \cdots - t_l - s_m - \cdots - s_2 \geq -(l + m - 2). \quad (\text{A-8})$$

In the left-hand side of Equation (A-3), s_i and s'_j have the same polarity, whereas t_k has the opposite polarity. Multiplying each of the preceding $(m + 1)$ inequalities ((A-4)-(A-8)) with a proper positive coefficient and summing all of them, we can derive an inequality of the same form as Equation (A-3) as follows:

$$\begin{aligned} & a_{m+1} \times (\text{A-4}) + a_m \times (\text{A-5}) + a_{m-1} \times (\text{A-6}) + \cdots + a_1 \times (\text{A-8}) \\ & \geq a_{m+1} \times (m - 1) - a_m \times (l - 1) - a_{m-1} \times l - a_1 \times (l + m - 2), \end{aligned} \quad (\text{A-9})$$

where

$$\begin{aligned} & a_1 + a_{m+1} \\ & = a_2 - a_1 + a_{m+1} \\ & \quad \vdots \\ & = a_m - a_{m-1} - \cdots - a_1 + a_{m+1} \\ & = a_1 + \cdots + a_m \\ & = -(-a_1 - \cdots - a_m). \end{aligned} \quad (\text{A-10})$$

Note that Equation (A-10) has essentially m equations with $(m + 1)$ variables (a_1, \dots, a_{m+1}) . One solution to the set of equations is:

$$a_1 = 1, a_2 = 2, a_3 = 4, \dots, a_m = 2^{m-1}, a_{m+1} = 2^m - 2,$$

with the right-hand side of Equation (A-9) being:

$$(2^m - 2)(m - 1) - \sum_{i=0}^{m-1} 2^i (l + m - 2 - i) = (2^m - 1)(m - l - 1) + 1.$$

Dividing both sides of Equation (A-9) by $a_1 + a_{m+1} = 2^m - 1$, we obtain:

$$s_1 + \cdots + s_m + s'_1 - t_1 - \cdots - t_l \geq m - l - 1 + \frac{1}{2^m - 1}.$$

As the left-hand side of the preceding inequality are all binary variables, we can rewrite it as:

$$s_1 + \cdots + s_m + s'_1 - t_1 - \cdots - t_l \geq m - l. \quad (\text{A-11})$$

As Equation (A-11) is derived from the inequality set Z , a solution to the set of inequalities Z is therefore a solution to Equation (A-11). Moreover, Equations (A-5) to (A-8) of inequality set Z can be derived from Equation (A-11) also, as shown in the following. As they are all binary variables, the following holds:

$$-2(s_m + \cdots + s_{i+1}) - (s_{i-1} + \cdots + s_1) \geq -(2m - i - 1). \quad (\text{A-12})$$

Adding Equations (A-11) and (A-12), we have:

$$s_i + s'_1 - t_1 - \cdots - t_l - s_m - \cdots - s_{i+1} \geq -(l + m - i - 1), \quad (\text{A-13})$$

which is the generalization of Equations (A-5) to (A-8). It means a solution to Equation (A-11) is also a solution to Equations (A-5) to (A-8). Therefore, Equations (A-5) to (A-8) can be replaced by Equation (A-11) with the same solutions. Note that Equation (A-4) cannot be derived from (A-11), so it remains without being replaced. Thus, the set of inequalities Z with $(m + 1)$ inequalities can be replaced by a new set of inequalities Z' with two inequalities, (A-4) and (A-11).

Now, we consider the general case where $n > 1$. The set of inequalities that capture all the constraints, denoted by X , consists of Equations (A-1), (A-2), and $(m \times n)$ inequalities between S_1 and S_2 . These $(m \times n)$ inequalities can be grouped into n subsets, Z_1, Z_2, \dots, Z_n , each of which consists of m inequalities. Applying the preceding simplification technique applied to Z to subset $Z_i, i = 1, \dots, n$, the set of inequalities X with $(m \times n + 2)$ inequalities can be replaced by a new set of inequalities X' with $(n + 2)$ inequalities:

$$s_1 + s_2 + \cdots + s_m \geq m - 1, \quad (\text{A-14})$$

$$s'_1 + s'_2 + \cdots + s'_n \geq n - 1, \quad (\text{A-15})$$

$$s'_n + s_1 + \cdots + s_m - t_1 - \cdots - t_l \geq m - l, \quad (\text{A-16})$$

$$s'_{n-1} + s_1 + \cdots + s_m - t_1 - \cdots - t_l - s'_n \geq m - (l + 1), \quad (\text{A-17})$$

$$\vdots \quad (\text{A-18})$$

$$s'_1 + s_1 + \cdots + s_m - t_1 - \cdots - t_l - s'_n - \cdots - s'_2 \geq m - (l + n - 1). \quad (\text{A-19})$$

Treating the m elements in subgroup S_1 as one single element in the preceding inequalities ((A-14)-(A-19)), we can further replace the n inequalities Equations (A-16) to (A-19) with one single inequality:

$$\begin{aligned} & s_1 + \cdots + s_m + s'_1 + \cdots + s'_n - t_1 - \cdots - t_l \\ & \geq \frac{(2^n - 2)(n - 1) + \sum_{i=0}^{n-1} 2^i (m - n - l + 1 + i)}{2^n - 1} \\ & = \frac{(m + n - l - 2)(2^n - 1) + 1}{2^n - 1} = m + n - l - 2 + \frac{1}{2^n - 1}. \end{aligned}$$

It is the same as:

$$s_1 + \cdots + s_m + s'_1 + \cdots + s'_n - t_1 - \cdots - t_l \geq m + n - l - 1, \quad (\text{A-20})$$

since the variables are binary valued. Therefore, the set of inequalities X' is further simplified to a set X'' with only 3 inequalities, Equations (A-1), (A-2) and (A-20).

For a group S of j pair-wise conflicting candidate edges that can be divided into k subgroups, S_1, \dots, S_k , of successive candidate edges, the number of inequality constraints can be reduced from $\binom{j}{2}$ to $\binom{k}{2}$. \square

REFERENCES

- CAO, A., SIRISANTANA, N., KOH, C.-K., AND ROY, K. 2002. Synthesis of selectively clocked skewed logic circuits. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*. San Jose, CA. 229–234.
- CAO, A., SIRISANTANA, N., KOH, C.-K., AND ROY, K. 2003. Integer linear programming-based synthesis of skewed logic circuits. In *Proceedings of the IEEE Asia South Pacific Design Automation Conference*. Kitakyushu, Japan. 820–823.
- DEY, S., BRGLEZ, F., AND KEDEM, G. 1990. Corolla based circuit partitioning and resynthesis. In *Proceedings of the ACM/IEEE Conference on Design Automation*. Orlando, FL. 607–612.
- GONCALVES, N. F. AND MAN, H. J. D. 1983. NORA: a race free dynamic CMOS technique for pipelined logic structures. *IEEE J. Solid State Circuits* 18, 3 (June), 261–266.
- KIM, C., LEE, J., BAEK, K. H., MARTINA, E., AND KANG, S. M. 2000. High-performance low-power skewed static logic in very deep-submicron (vds) technology. In *Proceedings of the IEEE International Conference on Computer Design*. Austin, TX. 59–64.
- KRAMBECK, R. H., LEE, C. M., AND LAW, H.-F. S. 1982. High-speed compact circuits with CMOS. *IEEE J. Solid State Circuits* 17, 3 (June), 614–619.
- PRASAD, M. R., KIRKPATRICK, D., BRAYTON, R. K., AND SANGIOVANNI-VINCENTELLI, A. 1997. Domino logic synthesis and technology mapping. In *Proceedings of the ACM/IEEE International Workshop on Logic Synthesis*. Tahoe City, CA.
- PURI, R., BJORKSTEN, A., AND ROSSER, T. E. 1996. Logic optimization by output phase assignment in dynamic logic synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. San Jose, CA. 2–8.
- REDDY, S. M. 1973. Complete test sets for logic functions. *IEEE Trans. Comput. C-22*, 11 (Nov.), 1016–1020.
- SENTOVICH, E. M., SINGH, K. J., LAVAGNO, L., AND ET AL. 1992. SIS: A system for sequential circuit synthesis. Tech. Rep. UCB/ERL M92/41 (May) *University of California at Berkeley*.
- SIRISANTANA, N., CAO, A., DAVIDSON, S., KOH, C.-K., AND ROY, K. 2001. Selectively clocked skewed logic (SCSL): A robust low-power logic style for high-performance applications. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design*. Huntington Beach, CA. 267–270.
- SOLOMATNIKOV, A., SOMASEKHAR, D., ROY, K., AND KOH, C.-K. 2000. Skewed CMOS: Noise-immune high-performance low-power static circuit family. In *Proceedings of the IEEE International Conference on Computer Design*. Austin, TX. 241–246.
- SOMASEKHAR, D. 1999. Power and dynamic noise considerations in high performance CMOS VLSI design. Ph.D. thesis, Purdue University.
- STOCKMEYER, L. 1983. Optimal orientations of cells in slicing floorplan designs. *Inform. Control* 57 (2/3), 91–101.
- THORP, T., YEE, G., AND SECHEN, C. 1999a. Design and synthesis of monotonic circuits. In *Proceedings of the IEEE International Conference on Computer Design*. Austin, TX. 569–572.
- THORP, T., YEE, G., AND SECHEN, C. 1999b. Monotonic static CMOS and dual Vt technology. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design*. San Diego, CA. 151–155.

ZHAO, M. AND SAPATNEKAR, S. S. 1998. Technology mapping for domino logic. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. San Jose, CA. 248–251.

ZHAO, M. AND SAPATNEKAR, S. S. 2000. Dual-monotonic domino gate mapping and optimal output phase assignment of domino logic. In *Proceedings of the IEEE International Symposium on Circuits and Systems*. Geneva, Switzerland. 309–312.

Received February 2003; revised September 2003; accepted March 2004