Integer Linear Programming-Based Synthesis of Skewed Logic Circuits*

Aiqun Cao, Naran Sirisantana, Cheng-Kok Koh and Kaushik Roy
School of Electrical and Computer Engineering, Purdue University
Email: {caoa, sirisant, chengkok, kaushik}@ecn.purdue.edu

Abstract— We present an integer linear programming-based approach for solving the logic reconvergence problem in skewed logic circuits with minimal logic duplication cost. A simplification technique is applied to reduce the complexity of the ILP problem greatly so that the run time is more affordable. Experimental results show that an average of 18% of original gates are duplicated in skewed logic circuits, whereas 65% in Domino logic circuits are duplicated. The average power saving over Domino logic circuits is 40.9%.

1 Introduction

Although Domino logic can achieve very high performance, it has two inherent drawbacks: it dissipates more power and has lower noise margin than static logic. A new noise-immune high-performance logic family, called monotonic static CMOS logic [7] or skewed logic [6], was proposed to solve the problems in Domino logic. Skewed logic circuit is 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 high-to-low transition, respectively.

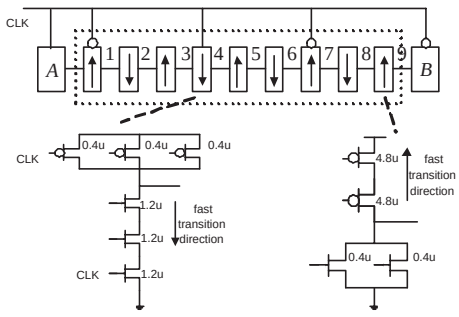

Figure 1: Topology and cascading of skewed logic.

When operating in precharge-evaluate fashion, skewed logic is comparable in speed to Domino logic. Fast transition is used for evaluation while precharging can be accomplished either by clocked skewed logic gates, or by the propagation of the precharged values using slow transition along the logic chain (see Figure 1). The structures of a clocked skewed-down NAND gate and an unclocked skewed-up NOR gate without clock are shown in Figure 1 also.

To ensure fast evaluation, alternating skew directions should be assigned to successive logic gates, i.e., skewed-down gates are followed by skewed-up gates and vice versa. Consequently, skewed logic encounters the same problem as Domino logic: logic reconvergent paths. For the synthesis of Domino logic, the fan-in cone of the reconvergent paths needs to be duplicated when there is an inverter trapped in it [3]. The duplication cost can be reduced by proper output phase assignment [4, 8]. The logic duplication technique and the output phase assignment technique for Domino logic synthesis can both be applied to the synthesis of skewed logic [7].

Ref. [5] made use of pass transistors to alleviate the reconvergent paths problem. However, it did not consider the glitch problem caused by introducing pass transistors, which would affect the performance of circuits. Ref. [1] corrected that and exploited the static nature of skewed logic to reduce the logic duplication penalty. The key lies in the observation that under certain conditions, assigning non-alternating skew directions to skewed logic gates does not impede the performance of the circuit. The reconvergent paths problem can be overcome by assigning non-alternating skew directions. In Ref. [1], each pair of reconvergent paths was solved independently, which made the solution space more restricted since there may be many reconvergent paths overlapping with each other.

In this paper, we take the overlapping of reconvergent paths into account by using an edge-based interger linear program (ILP) formulation. The reconvergent paths problem is solved with further reduced duplication cost. Moreover, we apply a simplification technique on the ILP formulation to reduce the complexity and run time of the approach. Experimental results show remarkable improvements.

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

2 Non-alternating skew directions

A careful analysis of the static skewed logic circuit reveals that we can make use of the NAND gate *B* in Figure 2(a) to avoid logic duplication as follows [1]: 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 2(b). The skew directions between gates *D* and *B* are non-alternating.

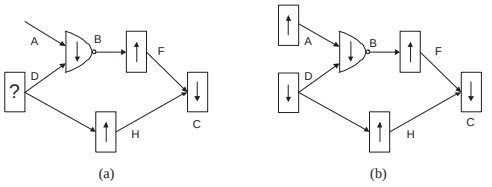

Figure 2: (a) Reconvergent paths with a NAND gate. (b) Non-alternating skew directions.

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 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, the problem that was overlooked in Ref. [1]. A similar analysis can be extended to NOR 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 NAND (NOR) gate is skewed up (down). We refer to the gates that allow non-alternating skew directions as *candidate gates*.

3 Integer linear program (ILP) formulation

3.1 Definitions

Given any pair of reconvergent paths, we refer to the node from which the reconvergent paths depart as *divergent node*, and the node at which the paths converge *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 paper, although there are no cyclic paths in the directed acyclic graph defined by the logic network.

Different cycles may overlap on some edges, thus a *petal* [2] is introduced. A petal consists of all the cycles that share edges among them. Moreover, some cycles are the union of two or more other cycles (after removing the shared edges in these cycles). These are called composite cycles, contrary to simple 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*.

3.2 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 non-alternating skew directions can be applied, while in Ref. [1] only one candidate gate in each cycle is considered, which made the solution space even more restricted. In our formulation, only candidate edges are the (binary) variables in the ILP. Edge e is assigned 0 if H_e is in the same skew direction as T_e, and 1 otherwise. All non-candidate edges implicitly have an assignment of 1.

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 a candidate gate: The non-alternating fan-ins of a candidate gate must be faster than the alternating fan-ins. Therefore, when a candidate edge is determined to be of value 1, all the slower sibling edges must also have value 1.
(ii) *Path constraints* stem from the fact that there is only one non-alternating skew direction for each candidate gate.

Consider the simple example in Figure 3(a). Candidate gate *A* (*D*) can only be skewed-up (skewed-down) for non-alternating scheme. But they cannot be assigned non-alternating directions simultaneously, as shown in Figure 3(a). In other words, candidate edges e_1 and e_2 cannot be both assigned 0. That constraint is captured by the following inequality: $e_1 + e_2 \geq 1$.

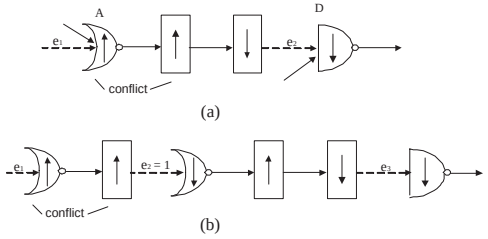

Figure 3: Path constraints.

Similarly, for three successive candidate edges, e_1, e_2 and e_3 in Figure 3(b), the two constraints for e_1 and e_2, and e_2 and e_3 are: $e_1 + e_2 \geq 1, e_2 + e_3 \geq 1$, 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 non-candidate edge, and the circuit degenerates to the case in Figure 3(a). Therefore, we would require a new constraint for e_1 and e_3: $e_1 + e_3 \geq 1$ if $e_2 = 1$, i.e. , $e_1 + e_3 - e_2 \geq 0$.

Therefore, 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 can be adequately captured by a single inequality constraint as follows: $e_1 + e_2 + e_3 \geq 2$, 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 by one single inequality: $e_n + e_{n-1} + ... + e_1 \geq n - 1$, which implies that only one candidate edge among those n edges can have value 0.

Above is a special case of the path constraint and the simplification technique. 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 $\binom{k}{2}$. For brevity, we omit the proof here. Without the simplification, the number of inequality constraints among these j candidate edges would be $\binom{j}{2}$.

(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 and similar simplification technique applies.

3.3 Skew direction assignment

First, we consider the skew direction assignment within a petal. Clearly, we can formulate the problem as an ILP in which the variables are the candidate edges, and the constraints are those *gate, path, and cycle constraints* formed by simple cycles within the petal. It can be shown by simple induction that if all those constraints are satisfied, the skew assignment of the whole petal is correct, i.e., the reconvergence problems are solved.

However, there is one compelling reason to abandon the approach suggested above. The suggested approach will produce a YES/NO answer, but 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 as possible cycles whose reconvergence problems can be resolved by non-alternating skew direction assignment. The other cycles would have their fan-in cones duplicated to overcome the logic reconvergence problem. To achieve that, 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. 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, 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.

4 Experimental results

We have implemented the ILP-based synthesis algorithm presented in Sections 3 in C++ language. We use a standard ILP package "lp_solve" to solve the ILPs. The solving of ILPs dominates the run time. Using the simplification technique presented in section 3.2 on the ISCAS benchmark circuits, the average reductions in the number of inequality constraints and the run time are 2.8x and 4.8x, respectively. After determining the skew direction of each gate, a dynamic-programming based heuristic is used to determine the skew value of each gate and to achieve an optimal clocking scheme [1].

For comparison, the benchmark circuits are implemented as Domino and skewed logic circuits. For Domino, we restrict the library of gates to contain only inverters, 2-input NAND and NOR gates of up to 6-input. Note that logic duplication is used to resolve the logic reconvergence in the Domino circuits. For skewed logic, the library con-

Circuit	Type	gates number	Circuit power (mW)	% reduction	% reduction from [1]	Total power (mW)	% reduction	% reduction from [1]
C432	Skew	349	28.05	41.9	30.7	32.32	45.7	35.8
	Domino	510	48.28			59.53		
C499	Skew	730	60.65	46.9	30.3	78.68	47.7	33.9
	Domino	1097	114.21			150.44		
C880	Skew	488	67.36	38.9	21.7	82.79	39.7	25.1
	Domino	695	110.24			137.30		
C1355	Skew	834	87.38	30.1	28.5	108.02	32.7	31.3
	Domino	1126	125.01			160.55		
C1908	Skew	898	81.50	36.9	24.5	98.69	38.9	27.1
	Domino	1416	129.16			161.53		
C2670	Skew	1385	137.94	42.9	31.4	167.15	45.2	34.3
	Domino	1990	241.72			305.18		
C3540	Skew	1879	149.07	40.7	34.7	179.45	42.3	37.2
	Domino	2787	251.36			311.22		
C5315	Skew	3094	223.16	31.8	29.9	266.27	34.3	33.1
	Domino	4114	327.21			405.09		
C7552	Skew	4309	348.99	40.9	32.3	422.67	42.0	35.2
	Domino	6425	590.50			728.74		
Average				39.0	29.3		40.9	32.6

Table 1: Experiment results.

tains inverters, and NAND and NOR gates of up to 4-input.

For every benchmark circuit, the two implementations operate at the same clock frequency (as dictated by how fast the Domino circuit is). Circuit simulation using PowerMill is performed using the 0.35μm CMOS technology at a supply voltage of 3.3V. 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 (18% versus 65%). That contributes to substantial power savings. The average power saving of skewed logic over Domino is 40.9%.

We also report in Table 1 the results obtained in Ref. [1]. Compared with Ref. [1], which needed 32% duplication and saved 32.6% power over Domino, we have remarkable improvements, while the run time is comparable. Note that although the results in Ref. [5] are better than those reported in Ref. [1] and this paper, the skewed logic circuits in Ref. [5] are more susceptible to glitches; therefore it is unfair to compare with its results.

References

[1] A. Cao, N. Sirisantana, C.-K. Koh, and K. Roy. Synthesis of selectively clocked skewed logic circuits. In *Proc. Int. Symp. on Quality Electronic Design*, pages 229–234, March 2002.

[2] S. Dey, F. Brglez, and G. Kedem. Corolla based circuit partitioning and resynthesis. In *Proc. Design Automation Conf*, pages 607–612, June 1990.

[3] M. R. Prasad, D. Kirkpatrick, R. K. Brayton, and A. Sangiovanni-Vincentelli. Domino logic synthesis and technology mapping. In *Proc. Int. Workshop on Logic Synthesis*, May 1997.

[4] R. Puri, A. Bjorksten, and T. E. Rosser. Logic optimization by output phase assignment in dynamic logic synthesis. In *Proc. Int. Conf. on Computer Aided Design*, pages 2–8, November 1996.

[5] N. Sirisantana, A. Cao, S. Davidson, C.-K. Koh, and K. Roy. Selectively clocked skewed logic (SCSL): a robust low-power logic style for high-performance applications. In *Proc. Int. Symp. on Low Power Electronics and Design*, pages 267–270, August 2001.

[6] A. Solomatnikov, D. Somasekhar, K. Roy, and C.-K. Koh. Skewed CMOS: noise-immune high-performance low-power static circuit family. In *Proc. IEEE Int. Conf. on Computer Design*, pages 241–246, September 2000.

[7] T. Thorp, G. Yee, and C. Sechen. Design and synthesis of monotonic circuits. In *Proc. IEEE Int. Conf. on Computer Design*, pages 569–572, October 1999.

[8] M. Zhao and S. S. Sapatnekar. Dual-monotonic domino gate mapping and optimal output phase assignment of domino logic. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pages 309–312, May 2000.