Domino Logic Synthesis Using Complex Static Gates

Tyler Thorp, Gin Yee and Carl Sechen
Department of Electrical Engineering
University of Washington, Seattle, WA 98195
{thorp,gsyee,sechen}@twolf.ee.washington.edu

1. Abstract

We address the synthesis of the most general form of a domino gate (dynamic-static domino), which consists of the pairing of a dynamic gate with any inverting static gate. All previous work focused on the synthesis of the most basic domino gate (standard domino), where the inverting static gate is an inverter. We developed a methodology and tools for synthesizing random logic blocks using both dynamic and complex inverting static gates in an alternating fashion. Dynamic-static (DS) domino can be used to reduce both gate levels and clock loading compared to standard domino. Comparisons between DS domino, standard domino, and static CMOS logic families are provided for six MCNC combinational logic benchmark circuits. Spice simulations show DS domino to have an average speed improvement of 53% over static CMOS and an average speed improvement of 17% over standard domino. DS domino also reduced clock loading by an average of 48% over standard domino. This paper introduces DS domino and presents a methodology for synthesizing random logic circuits using DS domino and other monotonic logic families such as Zipper CMOS.

2. Introduction

Domino logic is used in today's high performance microprocessors for implementing circuits that are both high speed and area efficient [1] [4] [5]. Among its many advantages, domino logic provides reduced input capacitance, low switching thresholds, and monotonic glitch-free operation. As with other dynamic logic families [9] [10] [11] [12] [14] [15] [17] [23], noise and charge sharing are key issues that must be dealt with when using domino logic. These challenges have limited domino logic's use to mainly timing critical paths in datapaths which are usually designed by hand. However, as market pressure forces ever faster clock frequencies, it seems that even random logic blocks will have to be implemented using domino logic in order to meet timing requirements. Currently, the performance advantages offered by domino logic are not available to random logic blocks due to the lack of tools for domino logic synthesis.

Synthesis of domino circuits is more complicated than that of static circuits. The added complexity is due to domino logic's monotonic nature which forces it to implement only non-inverting functions. Therefore, domino logic can only be mapped to a network of non-inverting functions, where needed logic inversions must be performed at either primary inputs and/or primary outputs. Any random logic network can be transformed into a network of non-inverting functions by finding a unate network representation.

Generating a unate network from a binate random logic network may require logic duplication since both positive and negative signal phases may be needed. An algorithm for finding the minimum logic duplication necessary when transforming a binate random logic network into an inverter-free unate network has been developed [21]. This algorithm performs technology independent logic optimization before finding the minimum amount of logic duplication necessary for making a unate network. Another approach for producing a unate network from a binate random logic network performs logic duplication before the technology independent logic optimization phase [19]. It is still unclear which method is best for finding a unate network. Binate to unate network conversion will at most double the amount of original logic and will not increase the number of logic levels.

Pioneering work in the area of domino logic synthesis addressed the need for allowing both positive and negative signal phases [2] [3] [6] [13]. A dual-rail logic network can provide both positive and negative signal phases if they are needed. Recently, a synthesis methodology for domino logic in which dual-rail logic cones are independently mapped has been proposed [19]. This approach uses a tree-by-tree mapping algorithm [7] to map a unate network to very large dynamic gates via a parameterized cell library.

By pairing a dynamic gate with a complex inverting static gate (a complex static gate is one which provides greater functionality than an inverter), DS domino can increase the functionality of a domino gate. Consequently, synthesizing domino networks with DS domino will result in fewer gate levels and fewer dynamic gates compared to standard domino. The static inverters in standard domino are typically skewed for a fast pull-up, the same applies to the design of the complex inverting static gates for DS domino. If complex static gates with fast pull-up networks are used to replace standard domino inverters, then the reduction in gate levels will correspond to an increase in performance. Also, reduction in the number of dynamic gates will decrease the clock loading. Therefore, DS domino can produce circuits that are faster and consume less power compared to standard domino.

This paper presents DS domino and a methodology for synthesizing circuits with DS domino and other monotonic logic families. The paper is organized as follows. Section 3 discusses standard domino logic and provides definitions for terms which will be frequently referred to in the following sections. Next, Section 4 provides the critical idea of merging non-inverting functions to form complex inverting gates. DS domino logic is then introduced in Section 5 followed by Section 6 which provides a methodology for synthesizing circuits with DS domino and other monotonic logic families. In Section 7, six MCNC combinational logic benchmarks are used to compare standard domino, DS domino, and static CMOS. Finally, the concluding remarks are given in Section 8.

3. Standard Domino Logic

A standard domino gate, as shown in Fig. 1, is composed of a dynamic gate and an inverter. It can only implement a non-inverting function. During the precharge phase of the clock, the output of the dynamic gate is set to 1. This sets the output from the inverter to 0. During the evaluation phase, the output of the dynamic gate will either switch from 1 to 0 or stay the same depending on its inputs. Any change in the dynamic gate's output will also cause the inverter's output to change. This propagation of logic through a domino network is similar to a chain of falling dominos, since each logic gate can evaluate true only if a previous logic gate has also evaluated true.

The dynamic gate is considered a pull-down gate since its output will either switch from 1 to 0 during evaluation or stay the same. The inverter is considered to be a pull-up gate since its output will either switch from 0 to 1 during evaluation or stay the same. Domino logic must guarantee that inputs to dynamic pull-down gates never make 1 to 0 transitions while the circuit is in evaluation, since any lost charge cannot be recovered. This constraint forces all inputs to dynamic gates to be monotonically increasing (only switch from 0 to 1 or stay the same).

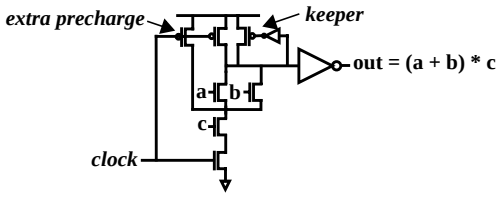

Fig. 1 Example of domino gate.

Charge sharing is a common problem among dynamic logic families that affects both circuit speed and power [16] [18] [20]. Under certain input conditions, charge sharing can occur between parasitic capacitances within a dynamic gate's internal nodes possibly causing glitches at the output node and falsely tripping subsequent gates. For internal nodes in the pull-down network that have considerable capacitance (for example, when three or more devices are tied to a node), the use of *extra precharge* devices is necessary as indicated in Fig. 1. However, the use of these extra devices decreases circuit performance. To help minimize the need for the extra precharge devices, dynamic gates with short evaluation paths should be used [8].

4. Implementing Non-inverting Functions

There are two basic forms for implementing a non-inverting function *f*. The first form shown in Fig. 2 illustrates function *f* as an AOI pull-down gate followed by an inverter. The second form shown in Fig. 2 illustrates function *f* as an OAI pull-up gate with its inputs being driven by inverters. For both of these two non-inverting forms, monotonically increasing inputs will exercise the paths through *f*. The paths through $\bar{f}$ do not affect the output switching time of the circuit during evaluation since they are only exercised during precharge.

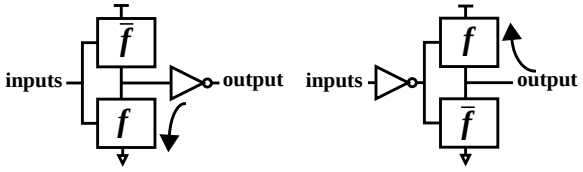

Fig. 2 Forms for implementing a non-inverting function *f*.

Using the two basic forms shown above, two non-inverting functions *f1* and *f2* can be merged to form a larger non-inverting function composed of a pull-down gate for implementing *f1* and a pull-up gate for implementing *f2*. Merging two non-inverting functions *f1* and *f2* can be achieved by either performing inverter cancellation, as shown in Fig. 3(a), or by pushing inverters toward the primary inputs and primary outputs, as shown in Fig. 3(b). The base case of two complex non-inverting functions being merged together to form two complex inverting functions can be extended to a

network of non-inverting functions. Any network of non-inverting functions can be mapped to an alternating pattern of complex pull-up and pull-down gates by applying the merging techniques, where the only remaining inverters would be at primary inputs and/or primary outputs.

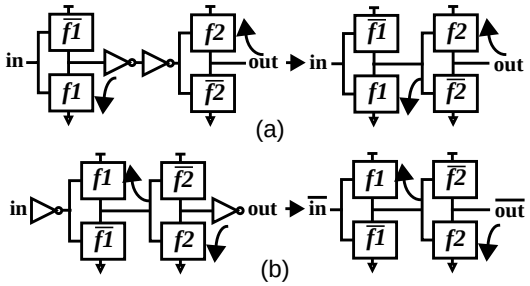

Fig. 3 Merging of two non-inverting functions *f1* and *f2*.

5. DS Domino Logic

As seen with standard domino, pairing a dynamic gate with an inverter will create a non-inverting gate. A DS domino gate, as shown in Fig. 4, creates a non-inverting gate by pairing a dynamic gate with an inverting complex static gate.

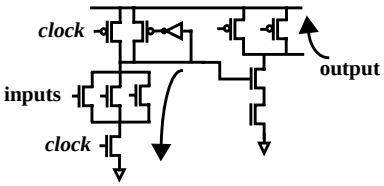

Fig. 4 Example of a DS domino gate.

If the inputs to the dynamic gate are guaranteed to be monotonically increasing, then the output of the static gate will also be monotonically increasing. Extending this idea to a network of gates, an alternating pattern of dynamic pull-down and static pull-up gates will also satisfy the requirement that all inputs to dynamic gates be monotonically increasing.

The merging techniques discussed in the previous section allow a network of non-inverting functions to be mapped to an alternating pattern of complex pull-up and pull-down gates. As shown in Fig. 5, a DS domino network can then be generated by implementing the pull-down gates as dynamic and the pull-up gates as static. Essentially, DS domino replaces the simple inverter in standard domino with a complex inverting static gate, thus allowing DS domino to provide greater functionality per gate compared to standard domino.

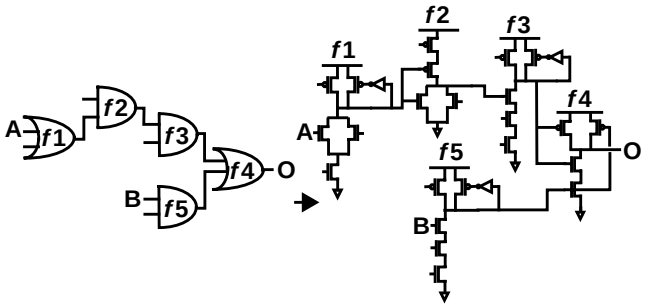

Fig. 5 Mapping a non-inverting network to DS domino.

6. Synthesis of DS Domino and Other Monotonic Logic

Since monotonic logic such as domino is inherently non-inverting, it must be mapped to a network that does not contain intermediate inversions. The removal of intermediate inverters within a logic network can be accomplished by finding a unate representation for the network. However, generating a unate network from a random logic network may require logic duplication since both positive and negative signal phases may be needed.

The first step in the process of generating a unate network from a binate random logic network is to create a base network consisting of AND2 gates, OR2 gates, and inverters. Using SIS [22], the base network is created by performing technology independent optimization using the *script.rugged* script and then technology mapping to the base network gates. After the base network is created, a bubble pushing algorithm is used to generate the unate network. (We could have used the method in [21], which may result in slightly smaller networks, i.e. less duplication. However, this would not impact the number of logic levels and hence the speed of the network, and the implementation of [21] is considerably more complex.) The bubble pushing algorithm removes inverters from a binate network by using DeMorgan's laws to push them towards the primary inputs. If an inverter cannot be pushed through a gate by simply applying DeMorgan's laws, then that gate must be duplicated in order to provide both positive and negative signal phases.

After a unate network representation has been created, the network can be technology mapped to domino gates. There are two methods for obtaining a DS domino circuit. The first method performs a modified two-coloring algorithm on an already mapped standard domino circuit. The two-coloring algorithm used for finding a pattern of complex pull-down gates and complex pull-up gates starts from the primary outputs and assigns colors to each gate such that every dynamic gate (D) is driving a static gate (S) and

every static gate is driving a dynamic gate. Fig. 6 shows a network of non-inverting functions and the resulting colored network of inverting functions. Since the internal inverters cancel out, only the gates labeled with a D or S will remain. Notice that the original non-inverting functions 1-6 are preserved throughout the network (i.e. we maintain logical correctness, since inverter cancellation takes place between every D-S connection).

Unfortunately, not every network can be two-colored. A modification to the two-coloring algorithm is needed to resolve color conflicts caused by an uneven difference in the number of gates between two reconvergent fanout paths. The conflict resolution scheme for DS domino is shown in Fig. 7. The non-inverting functions in Fig. 7 are represented by squares (the gates embedded within each square form a non-inverting function). Color conflicts are resolved by placing either a dual output standard domino gate or a dual output complex static gate at the root of the reconvergent fanout paths. This allows a valid coloring of the network since both (D) and (S) gates may be driven. The logic network maintains its correctness since the original non-inverting functions are still preserved.

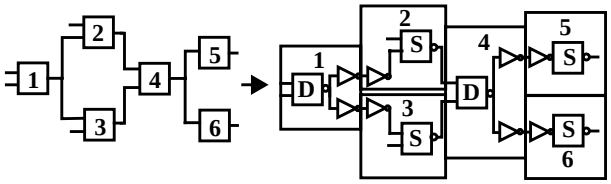

Fig. 6 Coloring a non-inverting network.

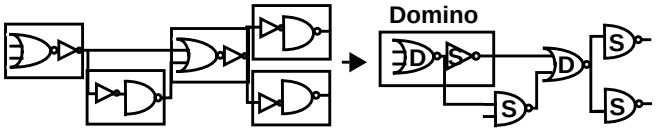

Fig. 7 Resolving color conflicts.

Performing the modified two-coloring algorithm after technology mapping does not allow different limits to be placed on dynamic and complex inverting static gates. In order to ensure high speed transitions during the evaluation phase, it may be desirable to place different limits on the number of devices in series and the number of inputs to dynamic and static gates. For a gate implementing a non-inverting function with monotonically increasing inputs (for either form in Fig. 2), the maximum number of transistors in series along the evaluation path is equal to the number of literals in the largest product term. Therefore, non-inverting functions can be merged together as long as the number of literals in the largest product term is less than or equal to the maximum allowable number of transistors in series.

The second method for obtaining a DS domino circuit alleviates the problem of not being able to assign different limits to dynamic and static gates by concurrently performing a valid two-coloring of the network and technology mapping. Our algorithm uses a library-free technology mapping approach to merge the unate network's non-inverting functions. It also ensures that each dynamic gate (or pull-down gate) will have no pull-down path (transistor chain) longer than a user specified limit and that each static gate (or pull-up gate) will have no pull-up path longer than a user-specified limit. This algorithm is outlined in Fig. 8.

```
procedure Merge_Nodes_and_Color(Network N)
  while performing a postorder traversal of network N from its outputs
    for all predecessors of current_node that are not multi-output nodes
        predecessor_node = current_node's predecessor with
         a) the greatest distance from a primary input and, to break ties,
         b) the fewest number of transistors in series
        if current_node has not been colored
          if node limits for predecessor's color are satisfied by the merge
            // current_node gets predecessor's color
            merge_predecessor_with_current_node();
          else
            // current_node gets opposite of predecessor's color
        elseif current_node & potential_merge_node have same color
          if node limits for predecessor's color are satisfied by the merge
            // current_node is already colored properly
            merge_predecessor_with_current_node();
          else
            resolve_color_conflict();
        else current_node & potential_merge_node have different colors
          if node limits for current_node's color are satisfied by the merge
            merge_predecessor_with_current_node();
            // since predecessor_node is changing its color,
            // conflicts need to be resolved with its predecessors
            resolve_color_conflicts();
          else
            // no color changes are needed
end Merge_Nodes;
```

Fig. 8 Algorithm for the coloring and mapping of a network.

After a network has been two-colored using either of the two methods discussed in this section, it can then be directly mapped to *any* alternating pattern of complex pull-down and complex pull-up inverting gates. Our synthesis procedure can produce the following monotonic networks: a) a monotonic static network consisting of static pull-down gates and static pull-up gates, b) a dual-rail domino network consisting of dynamic pull-down gates and pull-up inverters, c) a DS domino network consisting of dynamic pull-down gates and static pull-up gates, and d) a Zipper CMOS network consisting of dynamic nMOS pull-down gates and dynamic pMOS pull-up gates.

7. Experimental Results

Using the domino logic synthesis methodology described in the previous section, six MCNC combinational logic benchmark circuits were implemented using standard

domino (DOM), DS domino (DSD), and static CMOS (STA) logic families. Logic minimization for all circuits was performed using SIS and *script.rugged*. For the static CMOS circuits, a library similar to mcnc.genlib was used for technology mapping. Fanout optimization was used to limit the number of fanouts to eight. For the standard domino circuits, our library-free technology mapping algorithm without coloring constraints was used for mapping to domino gates with limits of two to four devices in series and six devices in parallel. For the DS domino circuits, our library-free technology mapping algorithm was used for mapping to dynamic gates with limits of two to four devices in series and six in parallel and to static gates with two devices in series and six in parallel. For the circuits generated using our domino logic synthesis tool, buffers were appended to gates with fanout greater than or equal to eight.

All logic devices in the nMOS (pMOS) networks for static CMOS, standard domino, and DS domino were 12 (30) µm. For dynamic gates, evaluate devices were 20µm and precharge devices were 8µm. Worst-case paths were found using Pathmill and Spice simulations were performed using MOSIS's 0.8µm scalable CMOS process parameters.

The worst-case delays for each benchmark are summarized in Table 1. The results show DS domino to have an average speed improvement of 53% over static CMOS and an average speed improvement of 17% over standard domino.

Table 1: Worst-case delay comparisons

Ckt.	STA [ns]	DOM [ns]	DSD [ns]	DOM/ STA	DSD/ STA	DSD/ DOM
b9	2.4	1.4	1.0	.58	.42	.71
k2	7.3	3.9	3.6	.53	.49	.92
term1	3.6	2.0	1.7	.55	.47	.85
rot	7.2	4.3	3.2	.59	.44	.74
x3	4.5	2.8	2.3	.62	.51	.82
dalu	8.4	4.7	4.3	.55	.51	.91
avg.				.57	.47	.83

A measure of clock loading can be seen by the number of dynamic gates used to implement a circuit. Table 2 displays the number of dynamic gates used in the implementation of each benchmark for dynamic and static gates with limits of two devices in series and six in parallel. The results show DS domino reduces the number of dynamic gates by an average of 48% over standard domino.

Gate counts for the static CMOS implementations are shown in Table 3. (S) refers to the number of complex static gates and (I) refers to the number of inverters. Gate counts for the standard domino and DS domino implementations are shown in Table 4. The number of inverters in the network immediately after unate network generation are given in the second column. (D) refers to the number of dynamic gates and (I) refers to the number of inverters, which is equal to the number of dynamic gates for standard domino. (S) refers to the number of complex static gates. The number of inverters needed to resolve coloring conflicts is shown in the final column. The standard domino and DS domino circuits below have been restricted to two devices in series and six devices in parallel.

Table 2: Dynamic gate count comparisons

Ckt.	DOM	DSD	DSD/DOM
b9	67	35	.52
k2	438	230	.53
term1	97	52	.54
rot	332	154	.46
x3	349	183	.52
dalu	279	152	.54
avg.			.52

Table 3: Gate counts for static CMOS

Ckt.	STA(S)	STA(I)
b9	62	47
k2	600	375
term1	95	54
rot	358	214
x3	386	227
dalu	505	217

Table 4: Gate counts for standard domino and DS domino

Ckt.	Initial inv's	DOM (D)	DOM (I)	DSD (D)	DSD (S)	Conflict inv's
b9	49	67	67	35	32	18
k2	62	438	438	230	208	119
term1	69	97	97	52	45	31
rot	196	332	332	154	178	81
x3	124	349	349	183	166	59
dalu	74	279	279	152	127	106

8. Conclusion

This paper introduces DS domino and describes a methodology for synthesizing circuits with DS domino and other monotonic logic families. We also believe this is the first reported synthesis procedure for generating Zipper CMOS networks for random logic blocks. By pairing dynamic gates with complex inverting static gates, DS domino can reduce both gate levels and clock loading compared to standard domino. The comparisons in Section 7 show DS domino circuits to have an average speed improvement of 53% over static CMOS circuits and an average speed improvement of 17% over standard domino circuits. DS domino is also shown to reduce clock loading by an average of 48% over standard domino. These results indicate DS domino to be an attractive option for implementing high performance and lower power VLSI circuits.

9. References

[1] B. Benschneider, A. Black, et al., "A 300-MHz 64-b quad-issue CMOS RISC microprocessor," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1203-1214, Nov. 1995.

[2] R. Brayton, C.L. Chen, Y. J. Yamour, "Method for optimizing logic for single-ended domino logic circuits," *IBM Technical Disclosure Bulletin*, vol. 27, no. 7B, pp. 4398-4401, Dec. 1984.

[3] R. Brayton, C.L. Chen, C. McMullen, R. Otten, and Y. J. Yamour, "Automated implementation of switching functions as dynamic CMOS circuits," *Proc. IEEE Custom Integrated Circuits Conference*, May 1984, pp. 346-350.

[4] A. Charnas, et. al., "A 64b microprocessor with multimedia support," *ISSCC Dig. of Tech. Papers*, Feb 1995, pp. 177-179.

[5] R. Colwell and R. Steck, "A 0.6mm BiCMOS processor with dynamic execution," *ISSCC Dig. of Tech. Papers*, Feb. 1995, pp. 176-177.

[6] G. De Micheli, "Performance-oriented synthesis of large-scale domino CMOS circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 5, pp. 751-765, Sep. 1987.

[7] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," *IEEE/ACM Int. Conf. on Computer-Aided Design*, pp 116-119, 1987.

[8] Tom Fletcher, private communication, Intel Corporation, May 1997.

[9] V. Friedman and S. Liu, "Dynamic logic CMOS circuits," *IEEE J. Solid-State Circuits*, vol. SC-19, no. 2, pp. 263-266, Apr. 1984.

[10] N. Goncalves and H. De Man, "NORA: a racefree dynamic CMOS technique for pipelined logic structures," *IEEE J. Solid-State Circuits*, vol. SC-18, no. 3, pp. 614-619, June 1983.

[11] R. Gu and M. Elmasry, "All-N-logic high-speed true-single-phase dynamic CMOS logic," *IEEE J. Solid-State Circuits*, vol. 31, no. 2, pp. 221-9, Feb. 1996.

[12] L. Heller and W. Griffin, "Cascade voltage switch logic: a differential CMOS logic family," *ISSCC Dig. of Tech. Papers*, pp. 16-17, 1984.

[13] M. Hofmann and A. Newton, "A domino CMOS logic synthesis system," *Proc. IEEE Int. Symposium on Curcuits and Systems*, pp. 411-414, 1985.

[14] Y. Ji-ren and C. Svensson, "High-speed CMOS circuit technique," *IEEE J. Solid-State Circuits*, vol. 24, no. 1, pp. 62-70, Feb 1989.

[15] R. H. Krambeck, C. M. Lee, H. S. Law, "High-speed compact circuits with CMOS," *IEEE J. Solid-State Circuits*, vol. SC-17, no. 3, pp. 614-619, June 1982.

[16] Patrik Larsson and Christer Svensson, "Noise in digital dynamic CMOS circuits," *IEEE J. Solid-State Circuits*, vol. SC-29, no. 3, pp. 655-662, June 1994.

[17] C. Lee and E. Szeto, "Zipper CMOS," *IEEE Circuits and Devices Magazine*, pp. 10-16, May 1986.

[18] V. G. Oklobdzija and R. K. Montoye, "Design-performance tradeoffs in CMOS domino logic," *Proc. IEEE 1985 Custom Integrated Circuits Conf.*, pp. 334-337, May 1985.

[19] M. R. Prasad, D. Kirkpatrick, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Domino logic synthesis and technology mapping," *Proc. Int. Workshop on Logic Synthesis*, May 1997, vol. 1.

[20] J. A. Pretorius, A. S. Shubat, and C. A. T Salama, "Charge redistribution and noise margins in Domino CMOS logic," *IEEE Transactions on Circuits and Systems*, vol CAS-33, no. 8, pp. 786-793, August 1986.

[21] R. Puri, A. Bjorksten, and T. E. Rosser, "Logic optimization by output phase assignment in dynamic logic synthesis," *IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 2-8, 1996.

[22] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: a system for sequential circuit synthesis," Technical Report UCB/ERL M92/41, University of California, Berkeley, CA, May 1992.

[23] G. Yee and C. Sechen, "Dynamic Logic Synthesis," *Proc. IEEE Custom Integrated Circuits Conference*, May 1997, pp. 345-348.