Logic Synthesis for Switching Lattices

Mustafa Altun and Marc D. Riedel

Abstract—This paper studies the implementation of Boolean functions by lattices of four-terminal switches. Each switch is controlled by a Boolean literal. If the literal takes the value 1, the corresponding switch is connected to its four neighbors; else it is not connected. A Boolean function is implemented in terms of connectivity across the lattice: it evaluates to 1 iff there exists a connected path between two opposing edges of the lattice. The paper addresses the following synthesis problem: how should one assign literals to switches in a lattice in order to implement a given target Boolean function? The goal is to minimize the lattice size, measured in terms of the number of switches. An efficient algorithm for this task is presented—one that does not exhaustively enumerate paths but rather exploits the concept of Boolean function *duality*. The algorithm produces lattices with a size that grows linearly with the number of products of the target Boolean function in ISOP form. It runs in time that grows polynomially. Synthesis trials are performed on standard benchmark circuits. The synthesis results are compared to a lower-bound calculation on the lattice size.

Index Terms—Boolean functions, switching circuits, lattices, nanowire crossbar arrays

◆

1 INTRODUCTION

In his seminal Master's Thesis, Shannon made the connection between Boolean algebra and switching circuits [2]. He considered two-terminal switches corresponding to electromagnetic relays. An example of a two-terminal switch is shown in the top part of Fig. 1. The switch is either ON (closed) or OFF (open). A Boolean function can be implemented in terms of connectivity across a network of switches, often arranged in a series/parallel configuration. An example is shown in the bottom part of Fig. 1. Each switch is controlled by a Boolean literal. If the literal is 1 (0) then the corresponding switch is ON (OFF). The Boolean function for the network evaluates to 1 if there is a closed path between the left and right nodes. It can be computed by taking the sum (OR) of the product (AND) of literals along each path. These products are $x_1x_2x_3$, $x_5x_1x_2x_6$, $x_5x_4x_2x_3$, and $x_5x_4x_6$.

In this paper, we develop a method for synthesizing Boolean functions with networks of four-terminal switches. An example is shown in the top part of Fig. 2. The four terminals of the switch are all either mutually connected (ON) or disconnected (OFF). We consider networks of four-terminal switches arranged in rectangular *lattices*. An example is shown in the bottom part of Fig. 2. Again, each switch is controlled by a Boolean literal. If the literal takes the value 1 (0) then corresponding switch is ON (OFF). The Boolean function for the lattice evaluates to 1 iff there is a closed path between the top and bottom edges of the lattice. Again, the function is computed by taking the sum of the products of the literals along each path. These products are $x_1x_2x_3$, $x_1x_2x_5x_6$, $x_4x_5x_2x_3$, and $x_4x_5x_6$—the same as those in Fig. 1. We conclude that this lattice of four-terminal switches implements the same Boolean function as the network of two-terminal switches in Fig. 1.

Throughout the paper, we will use a "checkerboard" representation for lattices where black and white sites represent ON and OFF switches, respectively, as illustrated in Fig. 3. We will discuss the Boolean functions implemented in terms of connectivity between the top and bottom edges as well as connectivity between the left and right edges. (We will refer to these edges as "plates.")

This paper addresses the following synthesis problem: how should we assign literals to switches in a lattice in order to implement a given target Boolean function? Suppose that we are asked to implement the function $f(x_1, x_2, x_3, x_4) = x_1x_2x_3 + x_1x_4$. We might consider the lattice in Fig. 4a. The product of the literals in the first column is $x_1x_2x_3$; the product of the literals in the second column is x_1x_4. We might also consider the lattice in Fig. 4b. The products for its columns are the same as those for Fig. 4a. In fact, the two lattices implement two different functions, only one of which is the intended target function. To see why this is so, note that we must consider all possible paths, including those shown by the red and blue lines. In Fig. 4a, the product x_1x_2 corresponding to the path shown by the red line covers the product $x_1x_2x_3$ so the function is $f_a = x_1x_2 + x_1x_4$. In Fig. 4b, the products $x_1x_2x_4$ and $x_1x_2x_3x_4$ corresponding to the paths shown by the red and blue lines are redundant, covered by column paths, so the function is $f_b = x_1x_2x_3 + x_1x_4$.

In this example, the target function is implemented by a 3×2 lattice with four paths. If we were given a target function with more products, a larger lattice would likely be needed to implement it; accordingly, we would need to enumerate more paths. Here, the problem is that number of paths grows exponentially with the lattice size. Any synthesis method that enumerates paths quickly becomes intractable. We present an efficient algorithm for this task—one that does not exhaustively enumerate paths but rather exploits the concept of Boolean function *duality* [3], [4]. Our synthesis algorithm produces lattices with a size

---

- *The authors are with the Department of Electrical and Computer Engineering, University of Minnesota, 200 Union St S.E., Minneapolis, MN 55455. E-mail: {altu0006, mriedel}@umn.edu.*

Manuscript received 21 Dec. 2010; revised 3 Aug. 2011; accepted 7 Aug. 2011; published online 31 Aug. 2011.
Recommended for acceptance by I. Markov.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-12-0703.
Digital Object Identifier no. 10.1109/TC.2011.170.

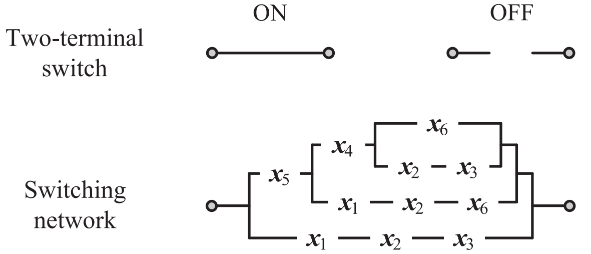


Fig. 1. Two-terminal switching network implementing the Boolean function $x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$.

that grows linearly with the number of products of the target Boolean function. It runs in time that grows polynomially.

The paper is organized as follows. In Section 2, we discuss potential technologies that fit our model of regular lattices of four-terminal switches. In Section 3, we present our general synthesis method that implements any target function with a lattice of four-terminal switches. In Section 4, we discuss the implementation of a specific function, the *parity function*. In Section 5, we derive a lower bound on the size of a lattice required to implement a Boolean function. In Section 6, we evaluate our general synthesis method on standard benchmark circuits. In Section 7, we discuss extensions and future directions for this research.

1.1 Definitions

Definition 1. *Consider k independent* **Boolean variables**, *$x_1, x_2, \ldots, x_k$.* **Boolean literals** *are Boolean variables and their complements, i.e., $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_k, \bar{x}_k$.*

Definition 2. *A* **product (P)** *is an AND of literals, e.g., $P = x_1\bar{x}_3x_4$. A* **set of a product (SP)** *is a set containing all the product's literals, e.g., if $P = x_1\bar{x}_3x_4$ then $SP = \{x_1, \bar{x}_3, x_4\}$. A* **sum-of-products (SOP)** *expression is an OR of products.*

Definition 3. *A* **prime implicant (PI)** *of a Boolean function f is a product that implies f such that removing any literal from the product results in a new product that does not imply f.*

Definition 4. *An* **irredundant sum-of-products (ISOP)** *expression is an SOP expression, where each product is a PI and no PI can be deleted without changing the Boolean function f represented by the expression.*

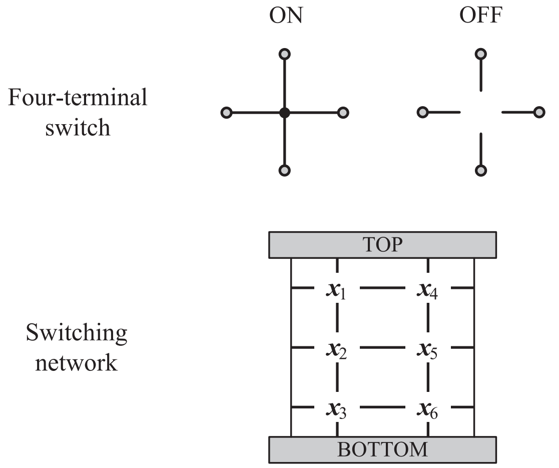
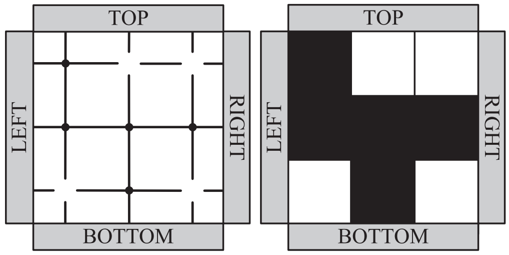


Fig. 2. Four-terminal switching network implementing the Boolean function $x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$.



(a) (b)

Fig. 3. A 3×3 four-terminal switch network and its lattice form.

Definition 5. *f and g are* **dual Boolean functions** *iff*

$$f(x_1, x_2, \ldots, x_k) = \bar{g}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k).$$

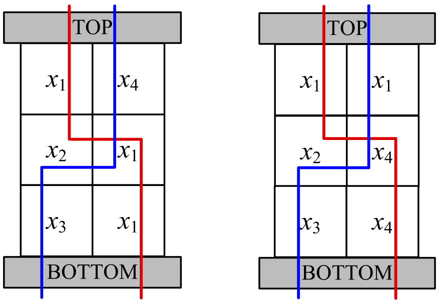
Given an expression for a Boolean function in terms of AND, OR, NOT, 0, and 1, its dual can also be obtained by interchanging the AND and OR operations as well as interchanging the constants 0 and 1. For example, if $f(x_1, x_2, x_3) = x_1x_2 + \bar{x}_1x_3$ then $f^D(x_1, x_2, x_3) = (x_1 + x_2)(\bar{x}_1 + x_3)$. A trivial example is that for $f = 1$, the dual is $f^D = 0$.

Definition 6. *A* **parity function** *is a Boolean function that evaluates to 1 iff the number of variables assigned to 1 is an odd number. The parity function f of k variables can be computed by the exclusive-OR (XOR) of the variables: $f = x_1 \oplus x_2 \oplus \cdots \oplus x_k$.*

2 APPLICABLE TECHNOLOGIES

The concept of regular two-dimensional arrays of four-terminal switches is not new; it dates back to a seminal paper by Akers in 1972 [5]. With the advent of a variety of types of emerging nanoscale technologies, the model has found renewed interest [6], [7]. Unlike conventional CMOS that can be patterned in complex ways with lithography, self-assembled nanoscale systems generally consist of regular structures. Logical functions are achieved with crossbar-type switches [8], [9]. Although conceptually general, our model corresponds to exactly this type of switch in a variety of emerging technologies.

A schematic for the realization of our model is shown in Fig. 5. Each site of the lattice is a four-terminal switch, controlled by an input voltage. When a high (logic 1) or low (logic 0) voltage is applied, the switch is ON or OFF, respectively. The output of the circuit depends upon the



(a) (b)

Fig. 4. Two 3×2 lattices implementing different Boolean functions.

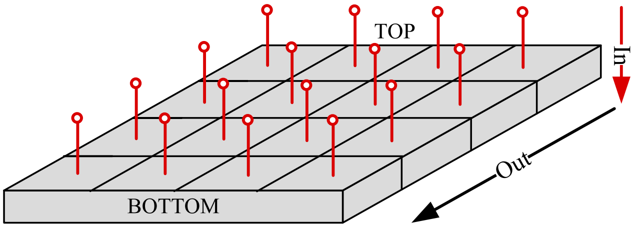


Fig. 5. 3D realization of our circuit model with the inputs and the output.

top-to-bottom connectivity across the lattice. If the top and bottom plates are connected, then the lattice allows signals to flow; accordingly, the output is logic 1. Otherwise the output is logic 0. The output can be sensed with a resistor connected to the bottom plate while a high voltage applied to the top plate. Below, we discuss two potential technologies that fit our model.

In their seminal work, Cui and Lieber investigated crossbar structures for different types of nanowires including n-type and p-type nanowires [10]. They achieved the different types of junctions by crossing different types of nanowires.

By crossing an n-type nanowire and a p-type nanowire, they achieved a diode-like junction. By crossing two n-types or two p-types, they achieved a resistor-like junction (with a very low resistance value). They showed that the connectivity of nanowires can be controlled by an insulated input voltage V-in. A high V-in makes the p-type nanowires conductive and the n-type nanowires resistive; a low V-in makes the p-type nanowires resistive and the n-type nanowires conductive. So they showed that, based on a controlling voltage, nanowires can behave either like short circuits or like open circuits.

A four-terminal switch can be implemented with the techniques of Cui and Lieber, as illustrated in Fig. 6. The switch has crossed p-type nanowires. When a high V-in is applied, the nanowires behave like short circuits. A resistor-like junction is formed, with low resistance. Thus, all four terminals are connected; the switch is ON. When a low V-in is applied, the nanowires behave like open circuits: all four terminals are disconnected; the switch is OFF. The result is a four-terminal switch that corresponds to our model.

Nanowire switches, of course, are assembled in large arrays. Indeed, the impetus for nanowire-based technology is the potential density, scalability and manufacturability [11], [12], [13]. Consider a p-type nanowire array, where each crosspoint is controlled by an input voltage. From the discussion above, we know that each such crosspoint behaves like a four-terminal switch. Accordingly, the nanowire crossbar array can be modeled as a lattice of four-terminal switches as illustrated in Fig. 7. Here, the black and white sites represent crosspoints that are ON and OFF, respectively.

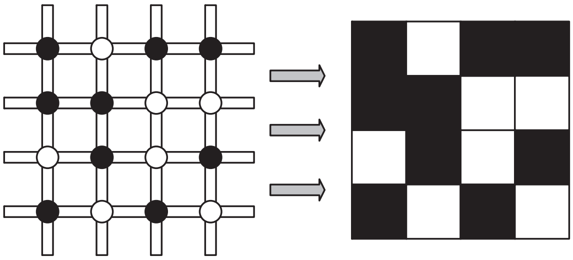


Fig. 7. Nanowire crossbar array with random connections and its lattice representation.

Nanowire crossbar arrays may offer substantial advantages over conventional CMOS when used to implement programmable architectures. Conventional implementations typically employ SRAMs for programming crosspoints. However, for nanoscale technologies, relative to the size of the switches, SRAMs would be prohibitively costly. A variety of techniques have been suggested for fabricating programmable nanowire crosspoints based on bistable switches that form memory cores [13], [14]. Also, molecular switches and solid-electrolyte nanoswitches could be used to form programmable crosspoints [15].

Other novel and emerging technologies fit our model of four-terminal switches. For instance, researchers are investigating *spin waves* [16]. Unlike conventional circuitry such as CMOS that transmits signals electrically, spin-wave technology transmits signals as propagating disturbances in the ordering of magnetic materials. Potentially, spin-wave-based logic circuits could compute with significantly less power than conventional CMOS circuitry.

Spin wave switches are four-terminal devices, as illustrated in Fig. 8. They have two states ON and OFF, controlled by an input voltage V-in. In the ON state, the switch transmits all spin waves; all four terminals are connected. In the OFF state, the switch reflects any incoming spin waves; all four terminals are disconnected. Spin-wave switches, like nanowire switches, are also configured in crossbar networks [17].

3 SYNTHESIS METHOD

In our synthesis method, a Boolean function is implemented by a lattice according to the connectivity between the top and bottom plates. In order to elucidate our method, we

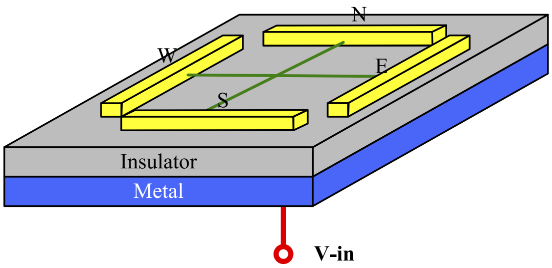


Fig. 6. Nanowire four-terminal switch.

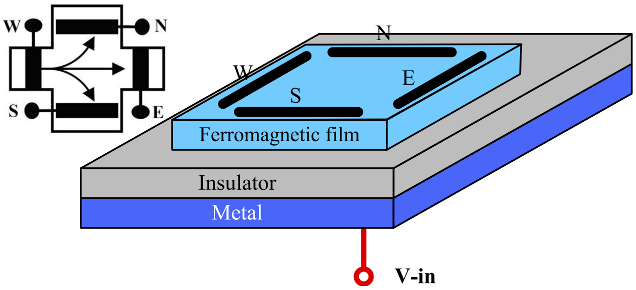
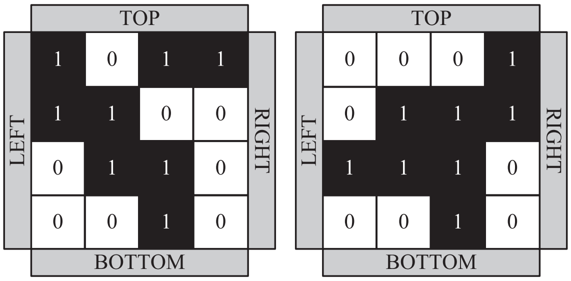


Fig. 8. Spin-wave switch.



(a) (b)

Fig. 9. Relationship between Boolean functionality and paths. (a) $f_L = 1$ and $g_L = 0$. (b) $f_L = 1$ and $g_L = 1$.

will also discuss connectivity between the left and right plates. Call the Boolean functions corresponding to the top-to-bottom and left-to-right plate connectivities f_L and g_L, respectively. As shown in Fig. 9, each Boolean function evaluates to 1 if there exists a path between corresponding plates, and evaluates to 0 otherwise. Thus, f_L can be computed as the OR of all top-to-bottom paths, and g_L as the OR of all left-to-right paths. Since each path corresponds to the AND of inputs, the paths taken together correspond to the OR of these AND terms, so implement sum-of-products expressions.

Example 1. Consider the lattice shown in Fig. 10. It consists of six switches. Consider the three top-to-bottom paths x_1x_4, x_2x_5, and x_3x_6. Consider the four left-to-right paths $x_1x_2x_3$, $x_1x_2x_5x_6$, $x_4x_5x_2x_3$, and $x_4x_5x_6$. While there are other possible paths, such as the one shown by the dashed line, all such paths are covered by the paths listed above. For instance, the path $x_1x_2x_5$ shown by the dashed line is covered by the path x_2x_5 shown by the solid line, and so is redundant. We conclude that the top-to-bottom function is the OR of the three products above, $f_L = x_1x_4 + x_2x_5 + x_3x_6$, and the left-to-right function is the OR of the four products above, $g_L = x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$.

We address the following logic synthesis problem: given a target Boolean function f_T, how should we assign literals to the sites in a lattice such that the top-to-bottom function f_L equals f_T? More specifically, how can we assign literals such that the OR of all the top-to-bottom paths equals f_T? In order to solve this problem, we exploit the concept of lattice duality, and work with both the target Boolean function and its dual.

Suppose that we are given a target Boolean function f_T and its dual f_T^D, both in ISOP form such that

$$\begin{aligned} f_T &= P_1 + P_2 + \cdots + P_n \quad \text{and} \\ f_T^D &= P_1' + P_2' + \cdots + P_m', \end{aligned}$$

where each P_i is a prime implicant of f_T, $i = 1, \ldots n$, and each P_j' is a prime implicant of f_T^D, $j = 1, \ldots m$.[1] We use a set representation for the prime implicants

$$\begin{aligned} P_i &\rightarrow SP_i, \quad i = 1, 2, \ldots, n \\ P_j' &\rightarrow SP_j', \quad j = 1, 2, \ldots, m, \end{aligned}$$

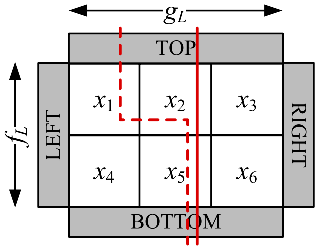


Fig. 10. A 2×3 lattice with assigned literals.

where each SP_i is the set of literals in the corresponding product P_i and each SP_j' is the set of literals in the corresponding product P_j'.

3.1 Algorithm

We first present the synthesis algorithm; then we illustrate it with examples; then we explain why it works.

Above we argued that, in establishing the Boolean function that a lattice implements, we must consider all possible paths. Paradoxically, our method allows us to consider only the *column paths* and the *row paths*, that is to say, the paths formed by straight-line connections between the top and bottom plates and between the left and right plates, respectively. Our algorithm is formulated in terms of the set representation of products and their intersections

1. Begin with f_T and its dual f_T^D, both in ISOP form. Suppose that f_T and f_T^D have n and m products, respectively.
2. Start with an $m \times n$ lattice. Assign each product of f_T to a column and each product of f_T^D to a row.
3. Compute intersection sets for every site, as shown in Fig. 11.
4. Arbitrarily select a literal from an intersection set and assign it to the corresponding site.

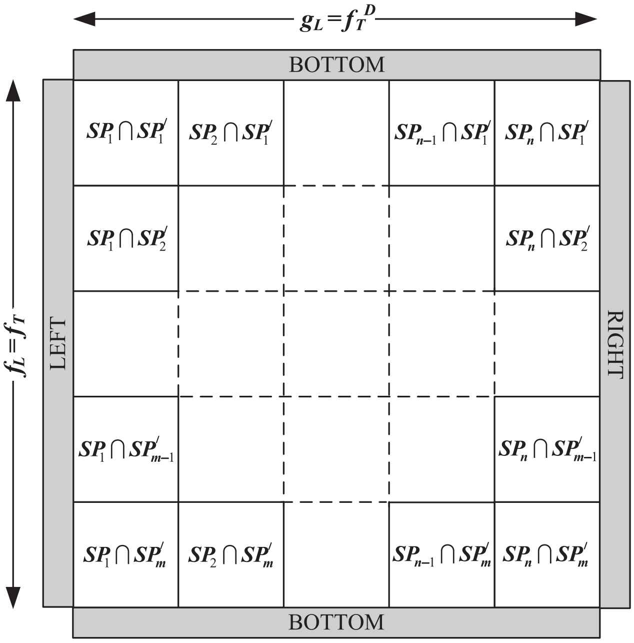
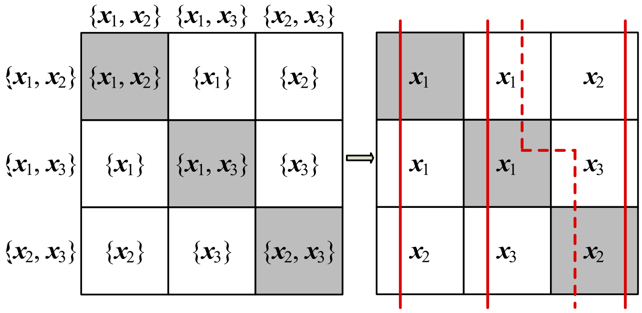


Fig. 11. Proposed implementation technique.

---

[1] Here $'$ is used to distinguish symbols. It does *not* indicate negation.



(a) (b)

Fig. 12. Implementing $f_T = x_1x_2 + x_1x_3 + x_2x_3$. (a) Lattice sites with corresponding sets. (b) Lattice sites with corresponding literals.

The proposed implementation technique is illustrated in Fig. 11. The technique implements f_T with an $m \times n$ lattice where n and m are the number of products of f_T and f_T^D, respectively. Each of the n column paths implements a product of f_T and each of the m row paths implements a product of f_T^D. As we explain in the next section, the resulting lattice implements f_T and f_T^D as the top-to-bottom and left-to-right functions, respectively. None of the paths other than the column and row paths need be considered.

We present a few examples to elucidate our algorithm.

Example 2. Suppose that we are given the following target function f_T in ISOP form:

$$f_T = x_1x_2 + x_1x_3 + x_2x_3.$$

We compute its dual f_T^D in ISOP form

$$f_T^D = (x_1 + x_2)(x_1 + x_3)(x_2 + x_3),$$
$$f_T^D = x_1x_2 + x_1x_3 + x_2x_3.$$

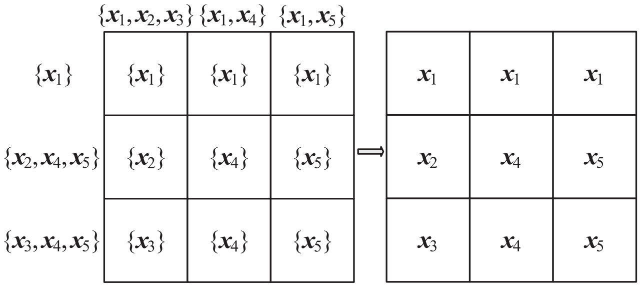
We have

$$SP_1 = \{x_1, x_2\}, \quad SP_2 = \{x_1, x_3\}, \quad SP_3 = \{x_2, x_3\},$$
$$SP'_1 = \{x_1, x_2\}, \quad SP'_2 = \{x_1, x_3\}, \quad SP'_3 = \{x_2, x_3\}.$$

Fig. 12 shows the implementation of the target function. Gray sites represent sets having more than one literal, which literal is selected for these sites is arbitrary. For example, selecting x_2, x_3, x_3 instead of x_1, x_1, x_2 does not change f_L and g_L. In order to implement the target function, we only use column paths; these are shown by the solid lines. All other paths are, in fact, redundant. Indeed there are a total of nine top-to-bottom paths: three column paths and six other paths; however, all other paths are covered by the column paths. For example, the path $x_1x_2x_3$ shown by the dashed line is a redundant path covered by the column paths. The lattice implements the top-to-bottom and left-to-right functions $f_L = f_T = x_1x_2 + x_1x_3 + x_2x_3$ and $g_L = f_T^D = x_1x_2 + x_1x_3 + x_2x_3$, respectively.

Example 3. Suppose that we are given the following target function f_T in ISOP form:

$$f_T = x_1x_2x_3 + x_1x_4 + x_1x_5.$$



(a) (b)

Fig. 13. Implementing $f_T = x_1x_2x_3 + x_1x_4 + x_1x_5$. (a) Lattice sites with corresponding sets. (b) Lattice sites with corresponding literals.

We compute its dual f_T^D in ISOP form

$$f_T^D = (x_1)(x_2 + x_4 + x_5)(x_3 + x_4 + x_5).$$
$$f_T^D = x_1 + x_2x_4x_5 + x_3x_4x_5.$$

We have

$$SP_1 = \{x_1, x_2, x_3\}, \quad SP_2 = \{x_1, x_4\}, \quad SP_3 = \{x_1, x_5\},$$
$$SP'_1 = \{x_1\}, \quad SP'_2 = \{x_2, x_4, x_5\}, \quad SP'_3 = \{x_3, x_4, x_5\}.$$

Fig. 13 shows the implementation of the target function. In this example, all the intersection sets are singletons, so the choice of which literal to assign is clear. The lattice implements $f_L = f_T = x_1x_2x_3 + x_1x_4 + x_1x_5$ and $g_L = f_T^D = x_1 + x_2x_4x_5 + x_3x_4x_5$.

We give another example, this one somewhat more complicated.

Example 4. Suppose that f_T and f_T^D are both given in ISOP form as follows:

$$f_T = x_1\bar{x}_2x_3 + x_1\bar{x}_4 + x_2x_3\bar{x}_4 + x_2x_4x_5 + x_3x_5 \quad \text{and}$$
$$f_T^D = x_1x_2x_5 + x_1x_3x_4 + x_2x_3\bar{x}_4 + \bar{x}_2\bar{x}_4x_5.$$

Fig. 14 shows the implementation of the target function. Gray sites represent intersection sets having more than one literal. For these sites, selection of the final literal is arbitrary. The result is $f_L = f_T = x_1\bar{x}_2x_3 + x_1\bar{x}_4 + x_2x_2\bar{x}_4 + x_2x_4x_5 + x_3x_5$ and $g_L = f_T^D = x_1x_2x_5 + x_1x_3x_4 + x_2x_3\bar{x}_4 + \bar{x}_2\bar{x}_4x_5$.

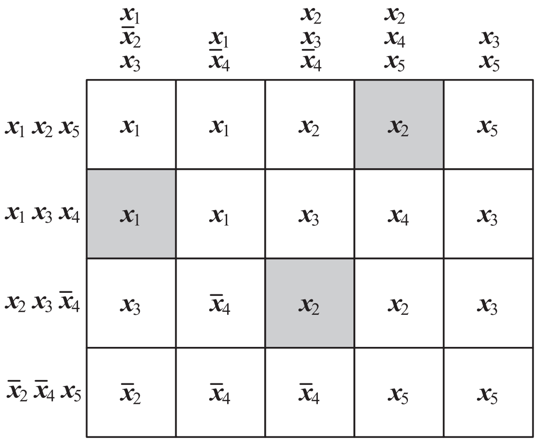
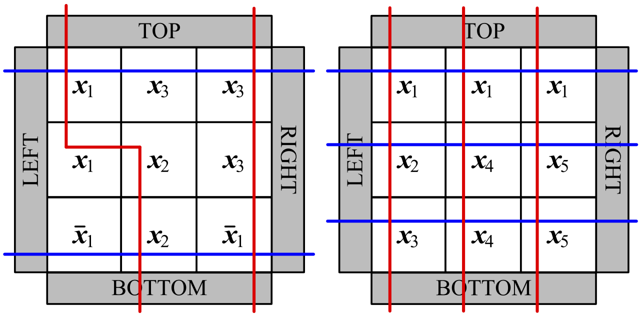


Fig. 14. Implementing $f_T = x_1\bar{x}_2x_3 + x_1\bar{x}_4 + x_2x_2\bar{x}_4 + x_2x_4x_5 + x_3x_5$.



(a) (b)

Fig. 15. Examples to illustrate Theorem 1.

3.2 Proof of Correctness

We present a proof of correctness of the synthesis method. Since our method does not enumerate paths, we must answer the question: for the top-to-bottom lattice function, how do we know that all paths other than the column paths are redundant? The following theorem answers this question. It pertains to the lattice functions and their duals.

Theorem 1. *If we can find two dual functions f and f^D that are implemented as subsets of all top-to-bottom and left-to-right paths, respectively, then $f_L = f$ and $g_L = f^D$.*

Before presenting the proof, we provide some examples to elucidate the theorem.

Example 5. We analyze the two lattices shown in Fig. 15.

Lattice (a). The top-to-bottom paths shown by the red lines implement $f = x_1x_2 + \bar{x}_1x_3$. The left-to-right paths shown by the blue lines implement $g = x_1x_3 + \bar{x}_1x_2$. Since $g = f^D$, we can apply Theorem 1: $f_L = f = x_1x_2 + \bar{x}_1x_3$ and $g_L = f^D = x_1x_3 + \bar{x}_1x_2$. Relying on the theorem, we obtain the functions *without* examining all possible paths. Let us check the result by using the formal definition of f_L and g_L, namely the OR of all corresponding paths. Since there are nine total top-to-bottom paths, $f_L = x_1x_1\bar{x}_1 + x_1x_1x_2x_2 + x_1x_1x_2x_3\bar{x}_1 + x_3x_2x_1\bar{x}_1 + x_3x_2x_2 + x_3x_2x_3\bar{x}_1 + x_3x_3\bar{x}_1 + x_3x_3x_2x_2 + x_3x_3x_2x_1\bar{x}_1$, which is equal to $x_1x_2 + \bar{x}_1x_3$. Thus, all the top-to-bottom paths but the paths shown by the red lines are redundant. Since there are nine total left-to-right paths, $g_L = x_1x_3x_3 + x_1x_3x_2x_3 + x_1x_3x_2x_2\bar{x}_1 + x_1x_2x_3x_3 + x_1x_2x_3 + x_1x_2x_2\bar{x}_1 + \bar{x}_1x_2x_2x_3x_3 + \bar{x}_1x_2x_2x_3 + \bar{x}_1x_2\bar{x}_1$, which is equal to $x_1x_3 + \bar{x}_1x_2$. Thus, all the left-to-right paths but the paths shown by the blue lines are redundant. So, Theorem 1 holds for this example.

Lattice (b). The top-to-bottom paths shown by the red lines implement $f = x_1x_2x_3 + x_1x_4 + x_1x_5$. The left-to-right paths shown by the blue lines implement $g = x_1 + x_2x_4x_5 + x_3x_4x_5$. Since $g = f^D$, we can apply Theorem 1: $f_L = f = x_1x_2x_3 + x_1x_4 + x_1x_5$ and $g_L = f^D = x_1 + x_2x_4x_5 + x_3x_4x_5$. Again, we see that Theorem 1 holds for this example.

Proof of Theorem 1. If $f(x_1, x_2, \ldots, x_k) = 1$ then $f_L = 1$. From the definition of duality, if $f(x_1, x_2, \ldots, x_k) = 0$ then $g(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k) = \bar{f}(x_1, x_2, \ldots, x_k) = 1$. This means that there is a left-to-right path consisting of all 0s; accordingly, $f_L = 0$. Thus, we conclude that $f_L = f$. Following the same argument for g, we conclude that $g_L = f^D$. □

Theorem 1 provides a constructive method for synthesizing lattices with the requisite property, namely that the top-to-bottom and left-to-right functions f_T and f_T^D are duals, and each column path of the lattice implements a product of f_T and each row path implements a product of f_T^D.

We begin by lining up the products of f_T as the column headings and the products of f_T^D as the row headings. We compute intersection sets for every lattice site. We arbitrarily select a literal from each intersection set and assign it to the corresponding site. The following lemma and theorem explain why we can make such an arbitrary selection.

Suppose that functions $f(x_1, x_2, \ldots, x_k)$ and $f^D(x_1, x_2, \ldots, x_k)$ are both given in ISOP form such that

$$f = P_1 + P_2 + \cdots + P_n \quad \text{and}$$
$$f^D = P'_1 + P'_2 + \cdots + P'_m,$$

where each P_i is a prime implicant of f, $i = 1, \ldots n$, and each P'_j is a prime implicant of f^D, $j = 1, \ldots m$. Again, we use a set representation for the prime implicants

$$P_i \rightarrow SP_i, \quad i = 1, 2, \ldots, n$$
$$P'_j \rightarrow SP'_j, \quad j = 1, 2, \ldots, m,$$

where each SP_i is the set of literals in the corresponding product P_i and each SP'_j is the set of literals in the corresponding product P'_j. Suppose that SP_i and SP'_j have z_i and z'_j elements, respectively. We first present a property of dual Boolean functions from [3].

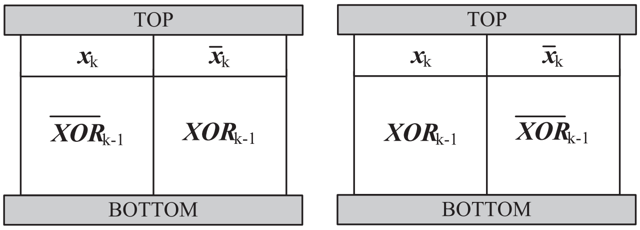
Lemma 1. *Dual pairs f and f^D must satisfy the condition*

$$SP_i \cap SP'_j \neq \emptyset \quad \textit{for every} \quad i = 1, 2, \ldots, n \quad \textit{and} \quad j = 1, 2, \ldots, m.$$

Proof of Lemma 1. The proof is by contradiction. Suppose that we focus on one product P_i from f and assign all its literals, namely those in the set SP_i, to 0. In this case, $f^D = 0$. However, if there is a product P'_j of f^D such that $SP'_j \cap SP_i = \emptyset$, then we can always make P'_j equal 1 because SP'_j does not contain any literals that have been previously assigned to 0. If follows that $f^D = 1$, a contradiction. □

Lemma 2. *Consider a product P with a corresponding set representation SP. Consider a Boolean function $f = P_1 + P_2 + \cdots + P_n$ with a corresponding set representation SP_i for each of its products P_i, $i = 1, 2, \ldots, n$. If SP has nonempty intersections with every SP_i, $i = 1, 2, \ldots, n$, then P is a product of f^D.*

Proof of Lemma 2. To prove that P is a product of f^D, we assign 1s to all the variables of P and see if this always results in $f^D = 1$. Since SP has nonempty intersections with every SP_i, $i = 1, 2, \ldots, n$, each product of f should have at least one assigned 1. From the definition of duality, these assigned 1s always result in $f^D = (1 + \cdots)(1 + \cdots) \cdots (1 + \cdots) = 1$. □



(a) (b)

Fig. 16. (a) Implementation of XOR_k. (b) Implementation of $\overline{XOR_k}$.

Theorem 2. *Assume that f and f^D are both in ISOP form. For any product P_i of f, there exist m nonempty intersection sets, $(SP_i \cap SP'_1), (SP_i \cap SP'_2), \ldots, (SP_i \cap SP'_m)$. Among these m sets, there must be at least z_i single-element disjoint sets. These single-element sets include all z_i literals of P_i.*

We can make the same claim for products of f^D: for any product P'_j of f^D, there exist n nonempty intersection sets, $(SP'_j \cap SP_1), (SP'_j \cap SP_2), \ldots, (SP'_j \cap SP_n)$. Among these n sets, there must be at least z'_j single-element disjoint sets that each represents one of the z'_j literals of P'_j.

Before proving the theorem, we elucidate it with examples.

Example 6. Suppose that we are given a target function f_T and its dual f_T^D, both in ISOP form such that

$$f_T = x_1\bar{x}_2 + \bar{x}_1x_2x_3 \quad \text{and} \quad f_T^D = x_1x_2 + x_1x_3 + \bar{x}_1\bar{x}_2.$$

Thus,

$$SP_1 = \{x_1, \bar{x}_2\}, \quad SP_2 = \{\bar{x}_1, x_2, x_3\},$$
$$SP'_1 = \{x_1, x_2\}, \quad SP'_2 = \{x_1, x_3\}, \quad SP'_3 = \{\bar{x}_1, \bar{x}_2\}.$$

Let us apply Theorem 2 for SP_2 $(z_2 = 3)$

$$SP_2 \cap SP'_1 = \{x_2\}, \; SP_2 \cap SP'_2 = \{x_3\}, \; SP_2 \cap SP'_3 = \{\bar{x}_1\}.$$

Since these three sets are all the single-element disjoint sets of the literals of SP_2, Theorem 2 is satisfied.

Example 7. Suppose that we are given a target function f_T and its dual f_T^D, both in ISOP form such that

$$f_T = x_1x_2 + x_1x_3 + x_2x_3 \quad \textit{and} \quad f_T^D = x_1x_2 + x_1x_3 + x_2x_3.$$

Thus,

$$SP_1 = \{x_1, x_2\}, \quad SP_2 = \{x_1, x_3\}, \quad SP_3 = \{x_2, x_3\},$$
$$SP'_1 = \{x_1, x_2\}, \quad SP'_2 = \{x_1, x_3\}, \quad SP'_3 = \{x_2, x_3\}.$$

Let us apply Theorem 2 for SP'_1 $(z'_1 = 2)$

$$SP'_1 \cap SP_1 = \{x_1, x_2\}, \; SP'_1 \cap SP_2 = \{x_1\}, \; SP'_1 \cap SP_3 = \{x_2\}.$$

Since $\{x_1\}$ and $\{x_2\}$, the single-element disjoint sets of the literals of SP'_1, are among these sets, Theorem 2 is satisfied.

Proof of Theorem 2. The proof is by contradiction. Consider a product P_i of f such that $SP_i = \{x_1, x_2, \ldots, x_{z_i}\}$. From Lemma 1, we know that SP_i has nonempty intersections with every SP'_j, $j = 1, 2, \ldots, m$. For one of the elements of SP_i, say x_1, assume that none of the intersection sets $(SP_i \cap SP'_1), (SP_i \cap SP'_2), \ldots, (SP_i \cap SP'_m)$ is $\{x_1\}$. This means that if we extract x_1 from SP_i then the new set $\{x_2, \ldots, x_{z_i}\}$ also has nonempty intersections with every SP'_j, $j = 1, 2, \ldots, m$. From Lemma 2, we know that the product $x_2x_3 \ldots x_{z_i}$ must be a product of f. This product covers P_i. However, in an ISOP expression, all products including P_i are irredundant, not covered by a product of f. So, we have a contradiction. □

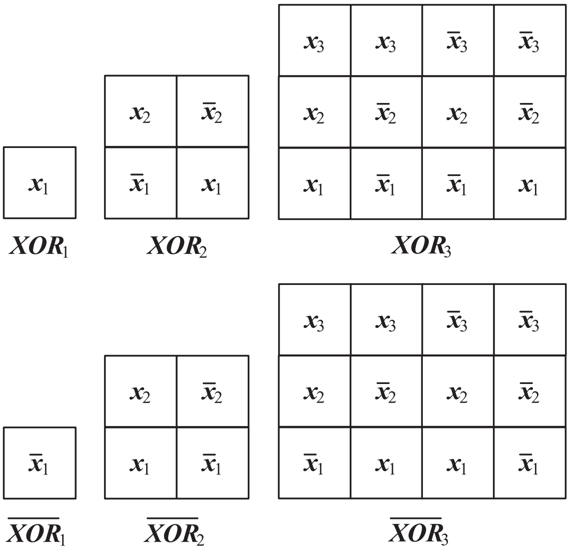

x_1
XOR_1
x_2 $\bar{x}_2$
$\bar{x}_1$ x_1
XOR_2
x_3 x_3 $\bar{x}_3$ $\bar{x}_3$
x_2 $\bar{x}_2$ x_2 $\bar{x}_2$
x_1 $\bar{x}_1$ $\bar{x}_1$ x_1
XOR_3
$\bar{x}_1$
$\overline{XOR}_1$
x_2 $\bar{x}_2$
x_1 $\bar{x}_1$
$\overline{XOR}_2$
x_3 x_3 $\bar{x}_3$ $\bar{x}_3$
x_2 $\bar{x}_2$ x_2 $\bar{x}_2$
$\bar{x}_1$ x_1 x_1 $\bar{x}_1$
$\overline{XOR}_3$

Fig. 17. Implementation of XOR_1, $\overline{XOR_1}$, XOR_2, $\overline{XOR_2}$, XOR_3, and $\overline{XOR_3}$.

From Lemma 1, we know that none of the lattice sites will have an empty intersection set. Theorem 2 states that the intersection sets of a product include single-element sets for *all* of its literals. So, the corresponding column or row has always all literals of the product regardless of the final literal selections from multiple-element sets. Thus, we obtain a lattice whose column paths and row paths implement f_T and f_T^D, respectively.

4 PARITY FUNCTIONS

The algorithm proposed in Section 3 provides a general method for implementing any type of Boolean function with an $m \times n$ lattice, where n and m are the number of products of the function and its dual, respectively. In this section, we discuss a method for implementing a specific function, the *parity function*, with a $(log(m) + 1) \times n$ lattice. Compared to the general method, we improve the lattice size by a factor of $m/(log(m) + 1)$ for this function.

As defined in Section 1.1, a k-variable parity function can be represented as a k-variable XOR operation. We exploit the following properties of XOR functions:

$$XOR_k = x_k\overline{XOR}_{k-1} + \bar{x}_kXOR_{k-1}$$
$$\overline{XOR}_k = x_kXOR_{k-1} + \bar{x}_k\overline{XOR}_{k-1}.$$

These properties allow us to implement both XOR_k and its complement $\overline{XOR_k}$ recursively. The approach for the k-variable parity function is illustrated in Fig. 16. The approach for 1, 2, and 3 variable parity functions is shown in Fig. 17. As in our general method, we implement each product of the target function with a separate column path;

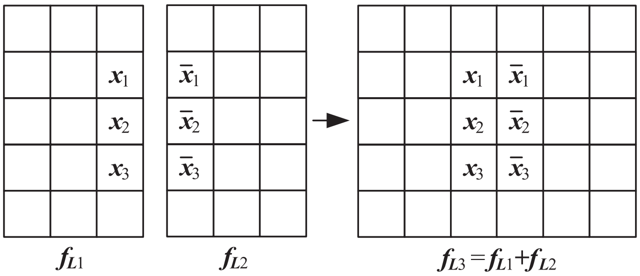


Fig. 18. An example illustrating Lemma 3.

in this construction, all paths other than column paths are redundant. The following lemma explains why this configuration works. Fig. 18 illustrates the lemma.

Lemma 3. *Consider two lattices with the same number of rows. Suppose that the lattices implement the Boolean functions f_{L1} and f_{L2}. Construct a new lattice with the two lattices side by side. If the attached columns of the lattices have negated variables facing each other for all rows except the first and the last, then the new lattice implements the Boolean function $f_{L3} = f_{L1} + f_{L2}$.*

Proof of Lemma 3. The new lattice has three types of paths: paths having all sites from the first lattice that implement f_{L1}, paths having all sites from the second lattice that implement f_{L2}, and paths having sites from both the first and the second lattices that implement f_{L1-2}. The Boolean function f_{L3} implemented by the third lattice is OR of the all paths; $f_{L3} = f_{L1} + f_{L2} + f_{L1-2}$. Since negated variables in attached columns result in $f_{L1-2} = 0$, we conclude that $f_{L3} = f_{L1} + f_{L2}$. □

We exploit Lemma 3 to compute the parity function as follows (please refer back to Fig. 16). We attach the lattices implementing $f_{L1} = x_k\overline{XOR}_{k-1}$ and $f_{L2} = \bar{x}_k XOR_{k-1}$ to implement $f_{L3} = XOR_k$. We attach the lattices implementing $f_{L1} = x_k XOR_{k-1}$ and $f_{L2} = \bar{x}_k\overline{XOR}_{k-1}$ to implement $f_{L3} = \overline{XOR}_k$. One can easily see that attached columns always have the proper configuration of negated variables to ensure that $f_{L1-2} = 0$.

5 A LOWER BOUND ON THE LATTICE SIZE

In this section, we propose a lower bound on the size of any lattice implementing a Boolean function. Although it is a weak lower bound, it allows us to gauge the effectiveness of our synthesis method. The bound is predicated on the maximum length of any path across the lattice. The length of such a path is bounded from below by the maximum number of literals in terms of an ISOP expression for the function.

5.1 Preliminaries

Definition 7. *Let the* **degree** *of an SOP expression be the maximum number of literals in terms of the expression.*

A Boolean function might have several different ISOP expressions and these might have different degrees. Among all the different expressions, we need the one with the smallest degree for our lower bound. (We need only consider ISOP expressions; every SOP expression is covered by an ISOP expression of equal or lesser degree.)

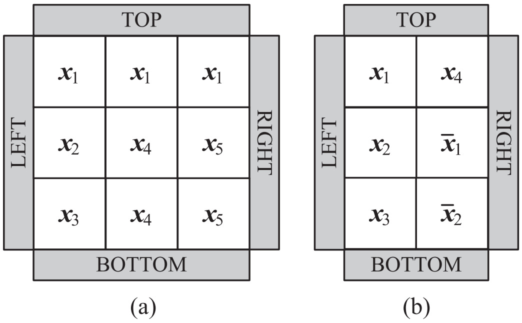


Fig. 19. Minimum-sized lattices (a) $f_L = f_{T1} = x_1x_2x_3 + x_1x_4 + x_1x_5$. (b) $f_L = f_{T2} = x_1x_2x_3 + \bar{x}_1\bar{x}_2x_4 + x_2x_3x_4$.

Consider a target Boolean function f_T and its dual f_T^D, both in ISOP form. We will use v and y to denote the minimum degrees of f_T and f_T^D, respectively. For example, if $v = 3$ and $y = 5$, this means that every ISOP expression for f_T includes terms with three literals or more, and every ISOP expression for f_T^D includes terms with five literals or more. Our lower bound, described in the next section by Theorem 4, consists of inequalities on v and y. We first illustrate how it works with an example.

Example 8. Consider two target Boolean functions $f_{T1} = x_1x_2x_3 + x_1x_4 + x_1x_5$ and $f_{T2} = x_1x_2x_3 + \bar{x}_1\bar{x}_2x_4 + x_2x_3x_4$, and their duals $f_{T1}^D = x_1 + x_2x_4x_5 + x_3x_4x_5$ and $f_{T2}^D = x_1x_4 + \bar{x}_1x_2 + \bar{x}_2x_3$. These expressions are all in ISOP form with minimum degrees. Since each expressions consists of three products, the synthesis method described in Section 3 implements each target function with a 3 × 3 lattice.

Examining the expressions, we see that the degrees of f_{T1} and f_{T2} are $v_1 = 3$ and $v_2 = 3$, respectively, and the degrees of f_{T1}^D and f_{T2}^D are $y_1 = 3$ and $y_2 = 2$, respectively. Our lower bounds based on these values are 3 × 3 for f_{T1} and 3 × 2 for f_{T2}. Thus, the lower bound for f_{T2} suggests that our synthesis method might not be producing optimal results. Indeed, Fig. 19 shows minimum-sized lattices for f_{T1} and f_{T2}. Here, the 3 × 2 lattice for f_{T2} was obtained through exhaustive search.

Since we implement Boolean functions in terms of top-to-bottom connectivity across the lattice, it is apparent that we cannot implement a target function f_T with top-to-bottom paths consisting of fewer than v literals, where v is the minimum degree of an ISOP expression for f_T. The following theorem explains the role of y, the minimum degree of f_T^D. It is based on *eight-connected* paths.[2]

Definition 8. *An* **eight-connected path** *consists of both directly and diagonally adjacent sites.*

An example is shown in Fig. 20. Here, the paths $x_1x_4x_8$ and $x_3x_6x_5x_8$ shown by red and blue lines are both eight-connected paths; however, only the blue one is four connected.

---

[2] Note that because our synthesis methodology is based on lattices of four-terminal switches, the target function f_T is always implemented by four-connected paths. We discuss eight-connected paths only because it is helpful to do so in order to prove our lower bound.

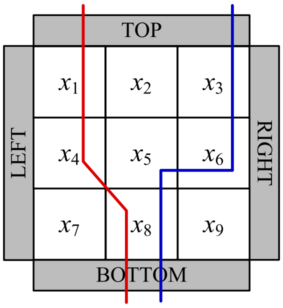


Fig. 20. A lattice with eight-connected paths.

Recall that f_L and g_L are defined as the OR of all four-connected top-to-bottom and left-to-right paths, respectively. (A lattice implements a given target function f_T if $f_L = f_T$.) We define f_{L-8} and g_{L-8} to be the OR of all eight-connected top-to-bottom and left-to-right paths, respectively.

Theorem 3. *The functions f_L and g_{L-8} are duals. The functions f_{L-8} and g_L duals.*

Before proving the theorem, we elucidate it with an example.

Example 9. Consider the lattice shown in Fig. 21. Here, f_L is the OR of three top-to-bottom four-connected paths x_1x_4, x_2x_5, and x_3x_6; g_L is the OR of four left-to-right four-connected paths $x_1x_2x_3$, $x_1x_2x_5x_6$, $x_4x_5x_2x_3$, and $x_4x_5x_6$; f_{L-8} is the OR of seven eight-connected top-to-bottom paths x_1x_4, x_1x_5, x_2x_4, x_2x_5, x_2x_6, x_3x_5, and x_3x_6; and g_{L-8} is the OR of eight eight-connected left-to-right paths $x_1x_2x_3$, $x_1x_2x_6$, $x_1x_5x_3$, $x_1x_5x_6$, $x_4x_2x_3$, $x_4x_2x_6$, $x_4x_5x_3$, and $x_4x_5x_6$. We can easily verify that $f_L = g^D_{L-8}$ and $f_{L-8} = g^D_L$. Accordingly, Theorem 3 holds true for this example.

Proof of Theorem 3. We consider two cases, namely $f_L = 1$ and $f_L = 0$.

Case 1: If $f_L(x_1, x_2, \ldots, x_k) = 1$, there must be a four-connected path of 1s between the top and bottom plates. If we complement all the inputs ($1 \rightarrow 0, 0 \rightarrow 1$), these four-connected 1s become 0s and vertically separate the lattice into two parts. Therefore, no eight-connected path of 1s exists between the left and right plates; accordingly, $g_{L-8}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k) = 0$. As a result, $\bar{g}_{L-8}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k) = f_L(x_1, x_2, \ldots, x_k) = 1$

Case 2: If $f_L(x_1, x_2, \ldots, x_k) = 0$, there must be an eight-connected path of 0s between the left and right plates. If we complement all the inputs, these eight-connected 0s become 1s; accordingly, $g_{L-8}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k) = 1$. As a result $\bar{g}_{L-8}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k) = f_L(x_1, x_2, \ldots, x_k) = 0$ □

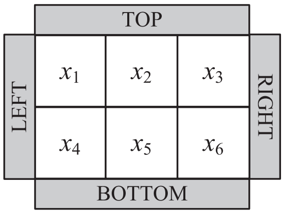


Fig. 21. A 2×3 lattice with assigned literals.

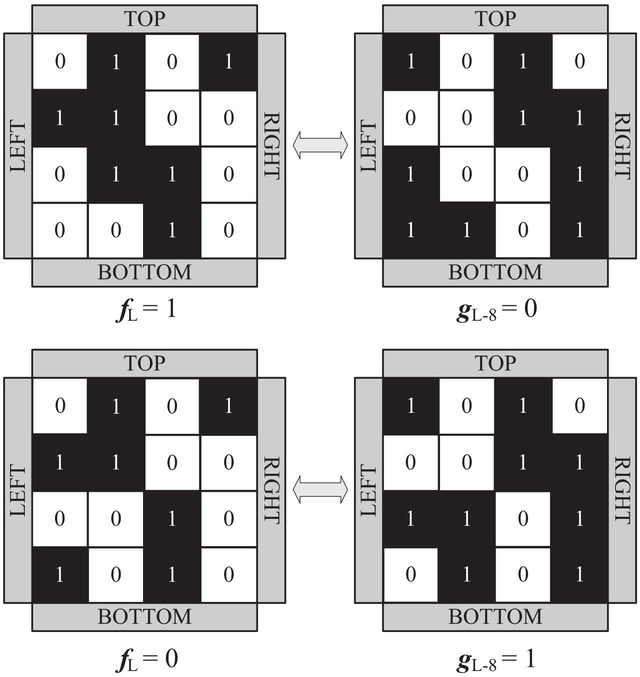


Fig. 22. Conceptual proof of Theorem 3.

Fig. 22 illustrates the two cases. Taken together, the two cases prove that f_L and g_{L-8} are duals. With inverse reasoning, we can prove that f_{L-8} and g_L are duals.

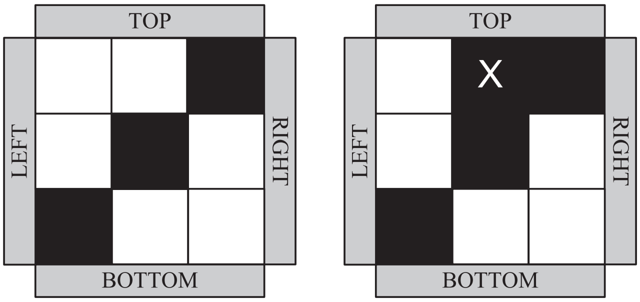
Theorem 3 tells us that the products of f^D_T are implemented with eight-connected left-to-right paths. Now consider y, the degree of f^D_T. We know that we cannot implement f^D_T with eight-connected right-to-left paths having fewer than y literals. Consider v, the degree of f_T. We know that we cannot implement f_T with four-connected top-to-bottom paths having fewer than v literals.

Returning to the functions in Example 8, we can now prove that lower bounds on the lattice sizes are nine (3×3) for f_{T1}, and six (3×2) for f_{T2}. Since $v_1 = 3$ and $y_1 = 3$ for f_{T1}, a 3×3 lattice is a minimum-size lattice that has four-connected top-to-bottom and eight-connected left-to-right paths of at least three literals, respectively. Since $v_2 = 3$ and $y_2 = 2$ for f_{T2}, a 3×2 lattice is a minimum-size lattice that has four-connected top-to-bottom and eight-connected left-to-right paths of at least three and two literals, respectively.

Based on these preliminaries, we now formulate the lower bound.

5.2 Lower Bound

Consider a target Boolean function f_T and its dual f^D_T, both in ISOP form. Recall that v and y are defined as the minimum degrees of f_T and f^D_T, respectively. Our lower bound is based on the observation that a minimum-size lattice must have a four-connected top-to-bottom path with at least v literals and an eight-connected left-to-right path with at least y literals. Since the functions are in ISOP form, all products of f_T and f^D_T are irredundant, i.e., not covered by other products. Therefore, we need only to consider irredundant paths:



(a) (b)

Fig. 23. Lattices with (a) an irredundant path and (b) a redundant path.

Definition 9. *A four-connected (eight-connected) path between plates is* **irredundant** *if it is not covered by another four-connected (eight-connected) path between the corresponding plates.*

We bound the length of irredundant paths. For example, the length of an eight-connected left-to-right path in a 3×3 lattice is at most three. Accordingly, no Boolean function with y greater than three can be implemented by a 3×3 lattice. Fig. 23 shows eight-connected left-to-right paths in a 3×3 lattice. The path in Fig. 23a consists of three sites. The path in Fig. 23b consists of four sites; however it is a redundant path—it is covered by the path in Fig. 23a.

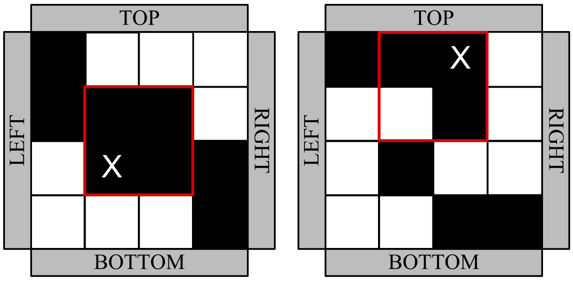
The following simple lemmas pertain to irredundant paths of a lattice.

Lemma 4. *An irredundant top-to-bottom path of a lattice contains exactly one site from the topmost row and exactly one site from the bottommost row. An irredundant left-to-right path of a lattice contains exactly one site from the leftmost column and exactly one site from the rightmost column.*

Proof of Lemma 4. All sites in the first row of a lattice are connected through the top plate. Therefore, we do not need a path to connect any two sites in this row; such a path is redundant. Similarly for the last row. Similarly for the first and last columns. □

Lemma 5. *An irredundant four-connected path of a lattice contains at most three of four sites in any 2×2 sublattice. An irredundant eight-connected path of a lattice contains at most two of four sites in any 2×2 sublattice.*

Proof of Lemma 5. In order to connect any two sites of a 2×2 sublattice with a four-connected path, we need at most three sites of the sublattice. Similarly, in order to connect any two sites of a 2×2 sublattice with an eight-connected path, we need at most two sites of the sublattice. □



(a) (b)

Fig. 24. Examples to illustrate Lemma 5.

Fig. 24 shows examples illustrating Lemma 5. The lattice in Fig. 24a has a four-connected top-to-bottom path. This path contains four of the four sites in the 2×2 sublattice encircled in red. Lemma 5 tells us that the path in Fig. 24a is redundant. Indeed, it is covered by the path achieved by removing the site marked by ×. The lattice in Fig. 24b has an eight-connected left-to-right path. This path contains three of four sites in the 2×2 sublattice encircled in red. Lemma 5 tells us that the path in Fig. 24b is redundant. Indeed, it is covered by the path achieved by removing the site marked by ×.

From Lemmas 4 and 5, we have the following theorem consisting of two inequalities. The first inequality states that the degree of f_T is equal to or less than the maximum number of sites in any four-connected top-to-bottom path. The second inequality states that the degree of f_T^D is less than or equal to the maximum number of sites in any eight-connected left-to-right path.

Theorem 4. *If a target Boolean function f_T is implemented by an $R \times C$ lattice then the following inequalities mu st be satisfied:*

$$v \le \begin{cases} R, & if\ R \le 2\ or\ C \le 1 \\ 3\left\lceil \frac{R-2}{2} \right\rceil \left\lceil \frac{C}{2} \right\rceil + \frac{2+(-1)^R+(-1)^C}{2}, & if R > 2\ and\ C > 1, \end{cases}$$

$$y \le \begin{cases} C, & if\ R \le 3\ or\ C \le 2 \\ 2\left\lceil \frac{R}{2} \right\rceil \left\lceil \frac{C-2}{2} \right\rceil + \frac{2+(-1)^R+(-1)^C}{2}, & if\ R > 3\ and\ C > 2, \end{cases}$$

where v and y are the minimum degrees of f_T and its dual f_T^D, respectively, both in ISOP form.

Proof of Theorem 4. If R and C are both even then all irredundant top-to-bottom and left-to-right paths contain at most $\frac{3}{4}(R-2)C+2$ and $\frac{2}{4}R(C-2)+2$ sites, respectively; this follows directly from Lemmas 4 and 5. If R or C are odd then we first round these up to the nearest even number. The resulting lattice contains at least one extra site (if either R or C but not both are odd) or two extra sites (if both R and C are odd). Accordingly, we compute the maximum number of sites in top-to-bottom and left-to-right paths and subtract 1 or 2. This calculation is reflected in the inequalities. □

The theorem proves our lower bound. Table 1 shows the calculation of the bound for different values of v and y up to 10.

6 EXPERIMENTAL RESULTS

In Table 2, we report synthesis results for a few standard benchmark circuits [18]. We treat each output of a benchmark circuit as a separate target function.

The values for n and m represent the number of products for each target function f_T and its dual f_T^D, respectively. We obtained these through sum-of-products minimization using the program Espresso [19]. The lattice size, representing the number of switches, is computed as a product of n and m.

TABLE 1
Lower Bounds on the Lattice Size for Different Values of v and y

v \ y	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	12	15	20	20	20	24
4	4	6	9	12	12	15	20	20	20	24
5	5	8	9	12	12	15	20	20	20	24
6	6	9	9	12	12	15	20	20	20	24
7	7	10	12	12	12	15	20	20	20	24
8	8	12	15	15	15	15	20	20	20	24
9	9	14	15	15	15	15	20	20	20	24
10	10	14	15	15	15	15	20	20	20	24

For the lower bound calculation, we obtained values of v and y, the minimum degrees of f_T and f_T^D, as follows: first we generated prime implicant tables for the target functions and their duals using Espresso with the "`-Dprimes`" option; then we deleted prime implicants one by one, beginning with those that had the most literals, until we obtained an expression of minimum degree. Given values of v and y, we computed the lower bound from the inequalities in Theorem 4.

Table 2 lists the runtimes for the lattice size and the lower bound calculations. The runtimes for the lattice size consist of the time for obtaining the functions' duals and for SOP minimization of both the functions and their duals. The runtimes for the lower bound consist of the time for generating the prime implicant tables and for obtaining the minimum degrees from the tables. We performed trials on an AMD Athlon 64 X2 6000+ Processor (at 3 GHz) with 3.6 GB of RAM running Linux.

Examining the numbers in Table 2, we see that, often, the synthesized lattice size matches the lower bound. In these cases, our results are optimal. However, for most of the Boolean functions, especially those with larger values of n and m, the lower bound is much smaller than the synthesized lattice size. This is not surprising since the lower bound is weak, formulated based on path lengths.

In the final column of Table 2, we list the number of transistors required in a CMOS implementation of the functions. We obtained the transistor counts through synthesis trials with the Berkeley tool ABC [20]. We applied the standard synthesis script "resyn2" in ABC and then mapped to a generic library consisting of NAND2 gates and inverters. We assume that each NAND2 gate requires four transistors and each inverter requires two transistors.

The number of switches needed by our method compares very favorably to the number of transistors required in a CMOS implementation. Of course, a rigorous comparison would depend on the specifics of the types of technology used. Each four-terminal switch might equate to more than one transistor. Then again, in some nanoscale technologies, it might equate to much less: the density of crosspoints in nanowire arrays is generally much higher than the density achievable with CMOS transistors.

7 DISCUSSION

The two-terminal switch model is fundamental and ubiquitous in electrical engineering [21]. Either implicitly or explicitly, nearly all logic synthesis methods target circuits built from independently controllable two-terminal switches (i.e., transistors). And yet, with the advent of novel nanoscale technologies, synthesis methods targeting lattices of multiterminal switches are *apropos*. Our treatment is at a technology-independent level; nevertheless, we comment that our synthesis results are applicable to technologies such as nanowire crossbar arrays with independently controllable crosspoints [13].

In this paper, we presented a synthesis method targeting regular lattices of four-terminal switches. Significantly, our method assigns literals to lattice sites without enumerating paths. It produces lattice sizes that are linear in the number of products of the target Boolean function. The time complexity of our synthesis algorithm is polynomial in the number of products. Comparing our results to a lower bound, we conclude that the synthesis results are not optimal. However, this is hardly surprising: at their core, most logic synthesis problems are computationally intractable; the solutions that are available are based on heuristics. Furthermore, good lower bounds on circuit size are notoriously difficult to establish. In fact, such proofs are related to fundamental questions in computer science, such as the separation of the P and NP complexity classes. (To prove that $P \neq NP$, it would suffice to find a class of problems in NP that cannot be computed by a polynomially sized circuit [22].)

The results on benchmarks illustrate that our method is effective for Boolean functions of practical interest. We should note, however, we would not expect it to be effective on some specific types of Boolean functions. In particular, our method will not be effective for Boolean functions that have duals with large number of products. The lattices for such functions will be inordinately large. For example, consider the function $f = x_1x_2x_3 + x_4x_5x_6 + x_7x_8x_9 + x_{10}x_{11}x_{12} + x_{13}x_{14}x_{15} + x_{16}x_{17}x_{18}$. It has only six products, but its dual has $3^6 = 729$ products. With our method, a lattice with 729 rows and six columns would be required.

The cost of implementing such functions could be mitigated by decomposing and implementing Boolean functions with separate lattices (or physically separated regions in a single lattice). This paper did not consider the topic of sharing logic among multiple output functions. Techniques for functional decomposition are well established [19], [23]. In future work, we will explore techniques for exploiting such decompositions and logic sharing in lattice-based implementations. Implementing multiple output functions will require some kind of physical partitioning of the lattice.

Another future direction is to extend the results in this paper to lattices of eight-terminal switches, and then to 2^k-terminal switches, for arbitrary k. Another direction is to study methods for synthesizing robust computation in

TABLE 2
Proposed Lattice Sizes, Lower Bounds on the Lattice Sizes, and CMOS Transistor Counts for Standard Benchmark Circuits

Circuit	n	m	**Lattice size** (number of switches)	**Runtime (s)**	v	y	**Lower bound**	**Runtime (s)**	**CMOS circuit size** (number of transistors)
alu1	3	2	6	< 0.01	2	3	6	0.02	18
alu1	2	3	6		3	2	6		26
alu1	1	3	3		3	1	3		16
clpl	4	4	16	< 0.01	4	4	12	0.01	42
clpl	3	3	9		3	3	9		26
clpl	2	2	4		2	2	4		10
clpl	6	6	36		6	6	15		74
clpl	5	5	25		5	5	12		64
newtag	8	4	32	< 0.01	3	6	15	< 0.01	60
dc1	4	4	16	< 0.01	3	3	9	< 0.01	38
dc1	2	3	6		3	2	6		24
dc1	4	4	16		3	4	12		40
dc1	4	5	20		4	3	9		42
dc1	3	3	9		2	3	6		26
misex1	2	5	10	< 0.01	4	2	6	0.01	64
misex1	5	7	35		4	4	12		84
misex1	5	8	40		5	4	12		64
misex1	4	7	28		5	3	9		58
misex1	5	5	25		4	4	12		76
misex1	6	7	42		4	4	12		64
misex1	5	7	35		4	3	9		36
ex5	1	3	3	0.26	3	1	3	3.17	16
ex5	1	5	5		5	1	5		24
ex5	1	4	4		4	1	4		18
ex5	1	7	7		7	1	7		36
ex5	1	8	8		8	1	8		40
ex5	1	6	6		6	1	6		34
ex5	8	4	32		3	6	15		46
ex5	10	4	40		3	8	20		52
ex5	7	3	21		3	7	20		44
ex5	7	3	21		3	6	15		48
ex5	8	2	16		2	8	16		42
ex5	9	4	36		3	8	20		56
ex5	8	2	16		2	7	14		42
ex5	12	6	72		4	7	20		70
ex5	14	8	112		4	7	20		388
ex5	7	2	14		2	7	14		38
ex5	6	3	18		3	6	15		40
ex5	6	2	12		2	6	12		36
ex5	10	7	70		3	7	20		76
ex5	6	6	36		3	6	15		64
ex5	12	10	120		4	8	20		318
ex5	14	8	112		5	7	20		350
ex5	8	5	40		3	7	20		86
ex5	10	8	80		3	7	20		116
ex5	12	7	84		4	7	20		356
ex5	9	3	27		3	8	20		60
ex5	5	2	10		2	5	10		44
b12	4	6	24	0.01	4	3	9	0.41	50
b12	7	5	35		4	4	12		54
b12	7	6	42		5	4	12		70
b12	4	2	8		2	2	4		16
b12	4	2	8		2	4	8		28
b12	5	1	5		1	5	5		30
b12	9	6	54		6	4	12		332
b12	6	4	24		4	6	15		60
b12	7	2	14		2	7	14		62
newbyte	1	5	5	< 0.01	5	1	5	< 0.01	26
newapla2	1	6	6	< 0.01	6	1	6	< 0.01	32
c17	3	3	9	< 0.01	2	3	6	< 0.01	16
c17	4	2	8		2	2	4		18
mp2d	11	1	11	0.01	1	11	11	0.61	46
mp2d	8	6	48		5	8	20		82
mp2d	10	5	50		4	10	24		102
mp2d	6	10	60		9	3	15		318
mp2d	1	5	5		5	1	5		26
mp2d	3	6	18		5	2	8		46
mp2d	1	8	8		8	1	8		36
mp2d	5	1	5		1	5	5		28

Each row lists the numbers for a separate output function of the benchmark circuit.

lattices with *random connectivity*. We have been exploring methods based on the principle of *percolation* [24].

A significant tangent for this work is its mathematical contribution: lattice-based implementations present a novel view of the properties of Boolean functions. We are curious to study the applicability of these properties to the famous problem of testing whether two monotone Boolean functions in ISOP form are dual. This is one of the few problems in circuit complexity whose precise tractability status is unknown [25].

ACKNOWLEDGMENTS

The authors would like to thank Ivo Rosenberg for his contributions. A preliminary version of this paper appeared in [1]. This work is supported by US National Science Foundation (NSF) CAREER award #0845650.

REFERENCES

[1] M. Altun and M.D. Riedel, "Lattice-Based Computation of Boolean Functions," *Proc. IEEE Design Automation Conf.*, pp. 609-612, 2010.

[2] C.E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans. Am. Inst. of Electrical Eng.*, vol. 57, no. 12, pp. 713-723, 1938.

[3] M.L. Fredman and L. Khachiyan, "On the Complexity of Dualization of Monotone Disjunctive Normal Forms," *J. Algorithms,* vol. 21, no. 3, pp. 618-628, 1996.

[4] T. Ibaraki and T. Kameda, "A Theory of Coteries: Mutual Exclusion in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 7, pp. 779-794, July 1993.

[5] S.B. Akers, "A Rectangular Logic Array," *IEEE Trans. Computers,* vol. C-21, no. 8, pp. 848-857, Aug. 1972.

[6] M. Chrzanowska-Jeske, Y. Xu, and M. Perkowski, "Logic Synthesis for a Regular Layout," *VLSI Design,* vol. 10, no. 1, pp. 35-55, 1999.

[7] M. Chrzanowska-Jeske and A. Mishchenko, "Synthesis for Regularity Using Decision Diagrams," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS)*, pp. 4721-4724, 2005.

[8] M.M. Ziegler and M.R. Stan, "CMOS/Nano Co-Design for Crossbar-Based Molecular Electronic Systems," *IEEE Trans. Nanotechnology,* vol. 2, no. 4, pp. 217-230, Dec. 2003.

[9] Y. Zomaya, "Molecular and Nanoscale Computing and Technology," *Handbook of Nature-Inspired and Innovative Computing,* ch. 14, pp. 478-520, Springer, 2006.

[10] Y. Cui and C.M. Lieber, "Functional Nanoscale Electronic Devices Assembled Using Silicon Nanowire Building Blocks," *Science,* vol. 291, no. 5505, pp. 851-853, 2001.

[11] Y. Huang, X. Duan, Y. Cui, L.J. Lauhon, K. Kim, and C.M. Lieber, "Logic Gates and Computation from Assembled Nanowire Building Blocks," *Science,* vol. 294, no. 5545, pp. 1313-1317, 2001.

[12] Y. Luo, C.P. Collier, J.O. Jeppesen, K.A. Nielsen, E. Delonna, G. Ho, J. Perkins, H. Tseng, T. Yamamoto, J.F. Stoddart, and J.R. Heath, "Two-Dimensional Molecular Electronics Circuits," *ChemPhysChem,* vol. 3, no. 6, pp. 519-525, 2002.

[13] A. DeHon, "Nanowire-Based Programmable Architectures," *ACM J. Emerging Technologies in Computing Systems,* vol. 1, no. 2, pp. 109-162, 2005.

[14] X. Duan, Y. Huang, and C.M. Lieber, "Nonvolatile Memory and Programmable Logic from Molecule-Gated Nanowires," *Nano Letters,* vol. 2, no. 5, pp. 87-90, 2002.

[15] S. Kaeriyama, T. Sakamoto, H. Sunamura, M. Mizuno, H. Kawaura, T. Hasegawa, K. Terabe, T. Nakayama, and M. Aono, "A Nonvolatile Programmable Solid-Electrolyte Nanometer Switch," *IEEE J. Solid-State Circuits,* vol. 40, no. 1, pp. 168-176, Jan. 2005.

[16] M.M. Eshaghian-Wilner, A. Khitun, S. Navab, and K. Wang, "A Nano-Scale Reconfigurable Mesh with Spin Waves," *Proc. Int'l Conf. Computing Frontiers,* pp. 5-9, 2006.

[17] A. Khitun, M. Bao, and K.L. Wang, "Spin Wave Magnetic Nanofabric: A New Approach to Spin-Based Logic Circuitry," *IEEE Trans. Magnetics,* vol. 44, no. 9, pp. 2141-2152, Sept. 2008.

[18] K. McElvain "IWLS93 Benchmark Set: Version 4.0, Distributed as Part of the IWLS93 Benchmark Distribution," http://www.cbl.ncsu.edu:16080/benchmarks/lgsynth93/, 1993.

[19] R.K. Brayton, C. McMullen, G.D. Hachtel, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1984.

[20] A. Mishchenko, "ABC: A System for Sequential Synthesis and Verification," http://www.eecs.berkeley.edu/alanmi/abc/, 2007.

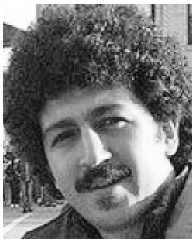
[21] R.E. Bryant, "Boolean Analysis of MOS Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 6, no. 4, pp. 634-649, July 1987.

[22] I. Wegener, "Lower Bounds on Circuit Complexity," *The Complexity of Boolean Functions,* ch. 5, pp. 119-144, John Wiley & Sons, 1987.

[23] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proc. IEEE,* vol. 78, no. 2, pp. 264-300, Feb. 1990.

[24] M. Altun, M.D. Riedel, and C. Neuhauser, "Nanoscale Digital Computation through Percolation," *Proc. Design Automation Conf.,* pp. 615-616, 2009.

[25] T. Eiter, K. Makino, and G. Gottlob, "Computational Aspects of Monotone Dualization: A Brief Survey," *Discrete Applied Math.,* vol. 156, no. 11, pp. 1952-2005, 2008.



Mustafa Altun received the BS and MSc degrees in electronics engineering at Istanbul Technical University, Turkey. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering at the University of Minnesota.

Marc D. Riedel received the BEng degree in electrical engineering with a Minor in mathematics at McGill University, and the MSc and PhD degrees in electrical engineering at Caltech. He has been an assistant professor of electrical and computer engineering at the University of Minnesota since 2006. He is also a member of the Graduate Faculty in Biomedical Informatics and Computational Biology. From 2004 to 2005, he was a lecturer in Computation and Neural Systems at Caltech. He has held positions at Marconi Canada, CAE Electronics, Toshiba, and Fujitsu Research Labs. His PhD dissertation titled "Cyclic Combinational Circuits" received the Charles H. Wilts Prize for the best doctoral research in Electrical Engineering at Caltech. His paper "The Synthesis of Cyclic Combinational Circuits" received the Best Paper Award at the Design Automation Conference. He is a recipient of the US National Science Foundation (NSF) CAREER Award.