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 neighbours; 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 we assign literals to switches in a lattice in order to implement a given target Boolean function? We seek to minimize the lattice size, measured in terms of the number of switches. We present an efficient algorithm for this task – one that does not exhaustively enumerate paths but rather exploits the concept of Boolean function *duality*. Our algorithm produces lattices with a size that grows linearly with the number of products of the target Boolean function. It runs in time that grows polynomially. We evaluate the algorithm on benchmark circuits. We compare the synthesis results 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, Claude 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 Figure 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 Figure 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 $x_1x_2x_3$, $x_1x_2x_5x_6$, $x_4x_5x_2x_3$, and $x_4x_5x_6$.

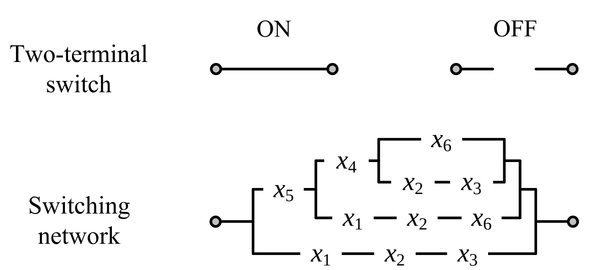

Fig. 1: Two-terminal switching network implementing the Boolean function $x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$.

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 Figure 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 Figure 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 $x_1x_2x_3$, $x_5x_1x_2x_6$, $x_5x_4x_2x_3$, and $x_5x_4x_6$ – the same as those in Figure 1. We conclude that this lattice of four-terminal switches implements the same Boolean function as the network of two-terminal switches in Figure 1.

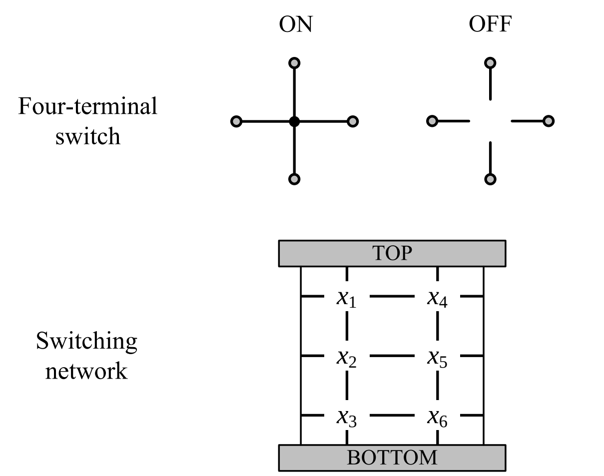

Fig. 2: Four-terminal switching network implementing the Boolean function $x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$.

Although conceptually general, our model of four-terminal switches is applicable for variety of nanoscale technologies, such as nanowire crossbar arrays [3], [4].

- *This work is supported by an NSF CAREER award #0845650.*
- *E-mail: {altu0006, mriedel}@umn.edu*
- *A preliminary version of this paper appeared in [1].*

It may also be applicable for magnetic and molecular switch-based structures [5], [6]. 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 Figure 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".)

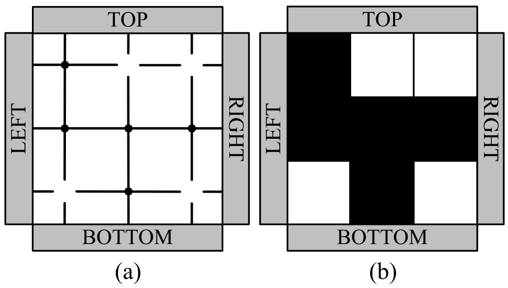

Fig. 3: A 3×3 four-terminal switch network and its lattice form.

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 Figure 4(a). 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 Figure 4(b). The products for its columns are the same as those for (a). 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 (a) 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 (b) 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 $f_b = x_1x_2x_3 + x_1x_4$.

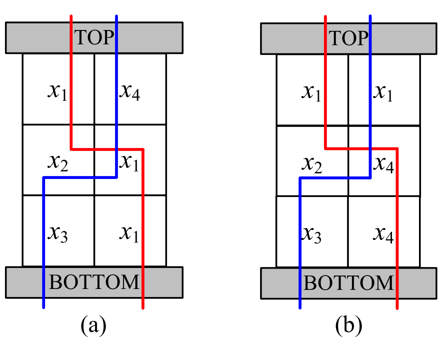

Fig. 4: Two 3×2 lattices implementing different Boolean functions.

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.

In Section 2, we present an efficient algorithm for this task – one that does not exhaustively enumerate paths but rather exploits the concept of Boolean function *duality* [7], [8]. Our algorithm produces lattices with a size that grows linearly with the number of products of the target Boolean function. It runs in time that grows polynomially. In Section 3, we derive a lower bound on the size of a lattice required to implement a Boolean function. In Section 4, we evaluate our synthesis method on standard benchmark circuits.

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 expression (ISOP)** *is an SOP, where each product is a PI and no PI can be deleted without changing the Boolean function f represented by the expression. Among the SOPs for f, one with the minimum number of products is a* **minimum sum-of-products expression (MSOP)**.

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 $f^D = 0$.

2 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 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 Figure 5, 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.

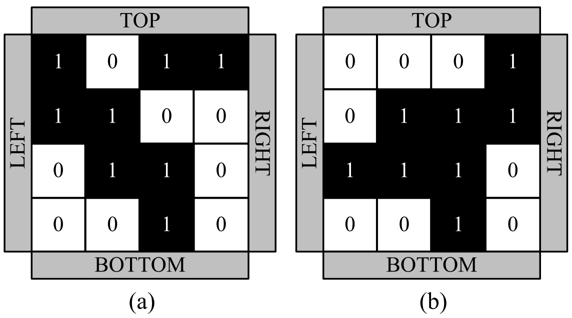

Fig. 5: Relationship between Boolean functionality and paths. (a): $f_L = 1$ and $g_L = 0$. (b): $f_L = 1$ and $g_L = 1$.

Example 1 *Consider the lattice shown in Figure 6. 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, $g_L = x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$.*

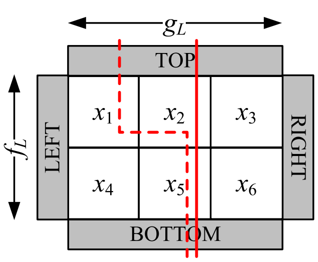

Fig. 6: A 2×3 lattice with assigned literals.

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

$$f_T = P_1 + P_2 + \cdots + P_n \quad \text{and}$$
$$f_T^D = P_1' + P_2' + \cdots + P_m'$$

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, $j = 1, \ldots m$.† 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'.

2.1 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 Figure 7.
4) Arbitrarily select a literal from an intersection set and assign it to the corresponding site.

The proposed implementation technique is illustrated in Figure 7. 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:*

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

†. Here $'$ is used to distinguish symbols. It does *not* indicate negation.

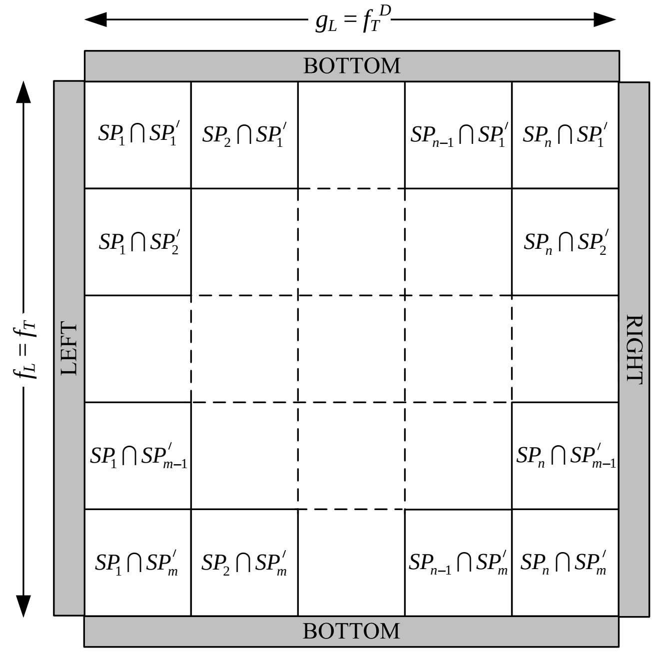

Fig. 7: Proposed implementation technique.

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:

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

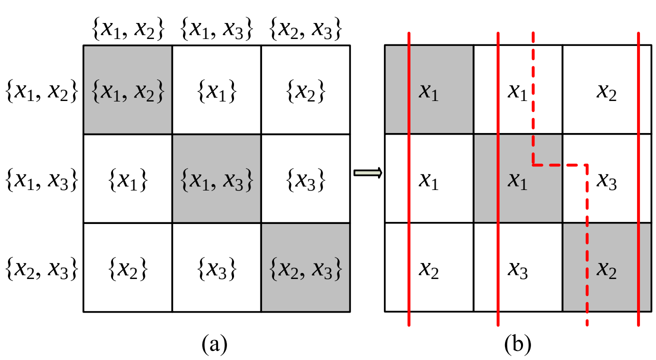

Fig. 8: Implementing $f_T = x_1x_2 + x_1x_3 + x_2x_3$. (a): Lattice sites with corresponding sets. (b): Lattice sites with corresponding literals.

Figure 8 shows the implementation of the target function. Grey 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 9 top-to-bottom paths: the 3 column paths and 6 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:*

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

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:

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

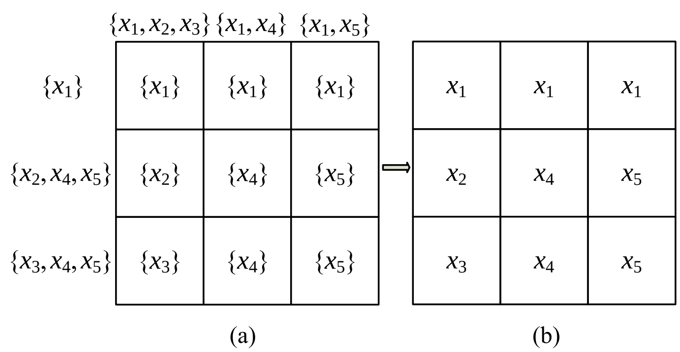

Fig. 9: Implementing $f_T = x_1x_2x_3 + x_1x_4 + x_1x_5$. (a): Lattice sites with corresponding sets. (b): Lattice sites with corresponding literals.

Figure 9 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 given in ISOP form as follows:*

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

Figure 10 shows the implementation of the target function. Grey 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 $g_L = f_T^D = x_1x_2x_5 + x_1x_3x_4 + x_2x_3\bar{x}_4 + \bar{x}_2\bar{x}_4x_5$.

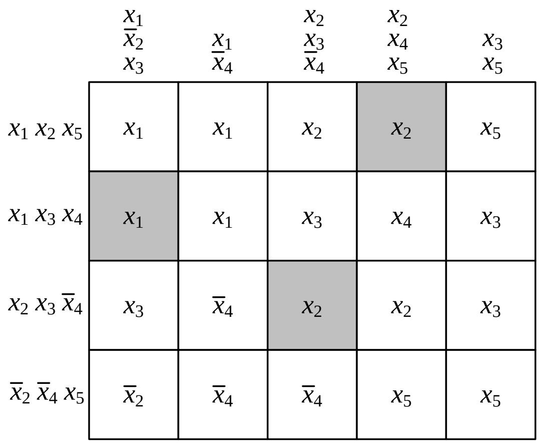

Fig. 10: Implementing $f_T = x_1\bar{x}_2x_3 + x_1\bar{x}_4 + x_2x_2\bar{x}_4 + x_2x_4x_5 + x_3x_5$.

2.2 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 Figure 11.*

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 9 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 9 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.*

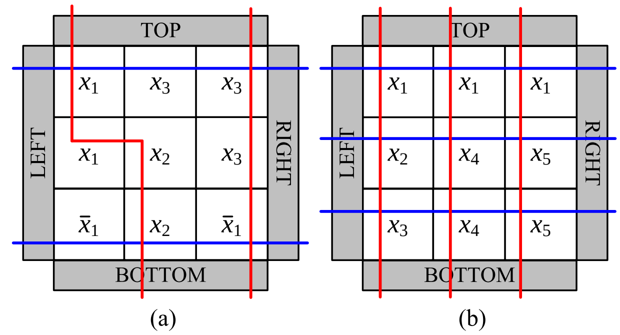

Fig. 11: Examples to illustrate Theorem 1.

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 0's; 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 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 [7]:

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

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

Proof: 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. □

Theorem 2 *Assume f and f^D are in ISOP form. For any product P_i of f, there exist m non-empty 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 non-empty 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 we are given a target function f_T and its dual f_T^D in ISOP form such that*

$$f_T = x_1\bar{x}_2 + \bar{x}_1 x_2 x_3 \text{ and } f_T^D = x_1x_2 + x_1x_3 + \bar{x}_1\bar{x}_2.$$

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 we are given a target function f_T and its dual f_T^D in ISOP form such that*

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

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 $SP_i = \{x_1, x_2, \ldots, x_{z_i}\}$. For one of the elements of SP_i, say x_1, assume that none of the intersection sets $(SP_i \cap SP'_1), (SP_i \cap SP'_2), \ldots, (SP_i \cap SP'_m)$ are $\{x_1\}$. This means that if we extract x_1 from SP_i then the new set $\{x_2, \ldots, x_{z_i}\}$ also has non-empty intersections with every SP'_j. Note that that the product $x_2x_3 \ldots x_{z_i}$ is one of the products of f. This product covers P_i. However in an ISOP there is no product that covers another. So we have a contradiction. □

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.

3 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.

3.1 Preliminaries

Definition 6 *Let the* **weight** *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 weights. Among all the different expressions, we need the one with the smallest weight for our lower bound. (We need only consider ISOP expressions; every SOP expression is covered by an ISOP expression of equal or lesser weight.)

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 weights 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 3 literals or more, and every ISOP expression for f_T^D includes terms with 5 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 in all ISOP form with minimum weights. Since each expressions consists of three products, the synthesis method described in Section 2 implements each target function with a 3×3 lattice.*

Examining the expressions, we see that the weights of f_{T1} and f_{T2} are $v_1 = 3$ and $v_2 = 3$, respectively, and the weights *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, Figure 12 shows minimum-sized lattices for for f_{T1} and f_{T2}. Here the 3×2 lattice for f_{T2} was obtained through exhaustive search.*

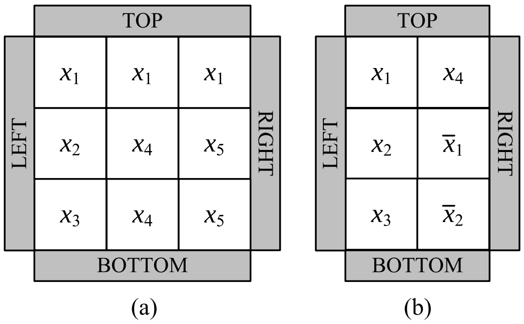

Fig. 12: Minimum-sized lattices (a): $f_L = f_{T1} = x_1x_2x_3 + x_1x_4 + x_1x_5$. (b): $f_L = f_{T2} = x_1x_2x_3 + \bar{x}_1\bar{x}_2x_4 + x_2x_3x_4$.

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 weight of an ISOP expression for f_T. The following theorem explains the role of y, the minimum weight of f_T^D. It is based on *eight-connected* paths.

Definition 7 *An* **eight-connected path** *consists of both directly and diagonally adjacent sites.*

An example is shown in Figure 13. 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.

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.

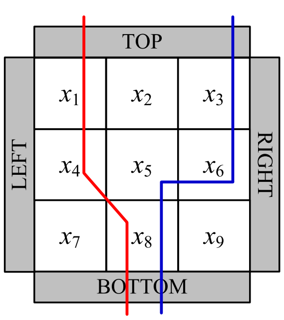

Fig. 13: A lattice with eight-connected paths.

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 Figure 14. Here f_L is the OR of 3 top-to-bottom four-connected paths x_1x_4, x_2x_5, and x_3x_6; g_L is the OR of 4 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 7 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 8 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 $f_L = g_{L-8}^D$ and $f_{L-8} = g_L^D$. Accordingly, Theorem 3 holds true for this example.*

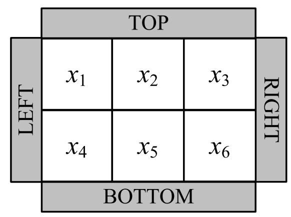

Fig. 14: A 2×3 lattice with assigned literals.

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

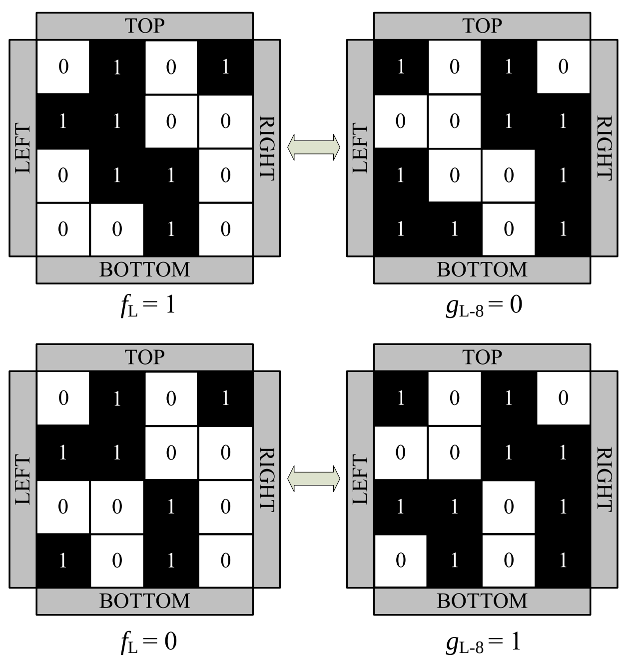

Fig. 15: Conceptual proof of Theorem 3.

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

Figure 15 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_T^D are implemented with eight-connected left-to-right paths. Now consider y, the weight of f_T^D. We know that we cannot implement f_T^D with eight-connected right-to-left paths having fewer than y literals. Consider v, the weight 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 9 (3×3) for f_{T1}, and 6 (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 3 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 3 and 2 literals, respectively.

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

3.2 Lower Bound

Consider a target Boolean function f_T and its dual f_T^D, both in ISOP form. Recall that v and y are defined as the minimum weights of f_T and f_T^D, 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_T^D are irredundant, i.e., not covered by other products. Therefore, we need only consider irredundant paths:

Definition 8 *A path between plates is* **irredundant** *if removing any site from the path does not result in another path between 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 3. Accordingly, no Boolean function with y greater than 3 can be implemented by a 3×3 lattice. Figure 16 shows eight-connected left-to-right paths in a 3×3 lattice. The path in (a) consists of 3 sites. The path in (b) consists of 4 sites; however it is a redundant path – removing the site indicated by $\times$ results in the path in (a).

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

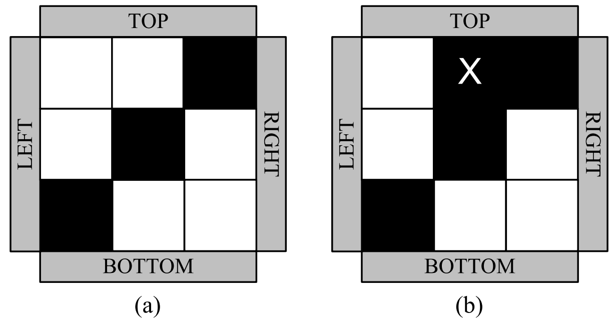

Fig. 16: Lattices with (a) irredundant paths and (b) redundant paths.

Lemma 2 *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: 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 3 *An irredundant four-connected path of a lattice contains at most 3 of 4 sites in any 2×2 sub-lattice. An irredundant eight-connected path of a lattice contains at most 2 of 4 sites in any 2×2 sub-lattice.*

Proof: In order to connect any 2 sites of a 2×2 sub-lattice with a four-connected path, we need at most 3 sites of the sub-lattice. Similarly, in order to connect any 2 sites of a 2×2 sub-lattice with an eight-connected path, we need at most 2 sites of the sub-lattice. □

Figure 17 shows examples illustrating Lemma 3. The lattice in (a) has a four-connected top-to-bottom path. This path contains 4 of the 4 sites in the 2×2 sub-lattice encircled in red. Lemma 3 tells us that the path in (a) is redundant. Indeed, the site marked by $\times$ can be removed. The lattice in (b) has an eight-connected left-to-right path. This path contains 3 of 4 sites in the 2×2 sub-lattice encircled in red. Lemma 3 tells us that the path in in (b) is redundant. Indeed the site marked by $\times$ can be removed.

From Lemmas 2 and 3, we have the following theorem consisting of two inequalities. The first inequality states that the weight 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 weight

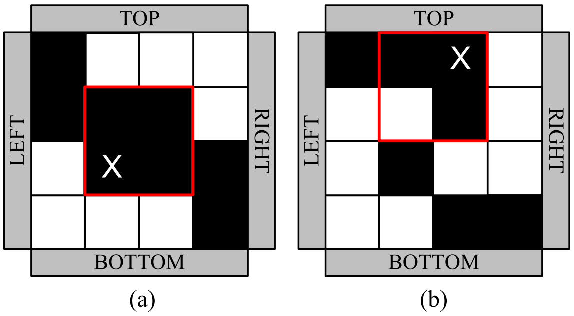

Fig. 17: Examples to illustrate Lemma 3.

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 must be satisfied:*

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

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

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

Proof: 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 2 and 3. 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 sties (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.

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

TABLE 1: Lower bounds on the lattice size for different values of v and y.

4 Experimental Results

We report synthesis results for a few standard benchmark circuits [9]. We treat each output of a benchmark circuit as a target function. Table 2 lists values for n and m, the number of products for each target function f_T and its dual f_T^D, respectively. These were obtained through sum-of-products minimization using the program Espresso [10]. (The runtimes are simply those for SOP minimization.)

For the lower bound calculation, we obtained values of v and y, the minimum weights 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 weight. (Again, the runtimes in the table are simply those for SOP minimization.) Given values of v and y, the lower bound is computed from the inequalities in Theorem 4.

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.

5 Discussion

The two-terminal switch model is fundamental and ubiquitous in electrical engineering [11]. 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 multi-terminal 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 [3].

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

Circuit	n	m	Lattice size	v	y	Lower bound
alu1	3	2	6	2	3	6
alu1	2	3	6	3	2	6
alu1	1	3	3	3	1	3
clpl	4	4	16	4	4	12
clpl	3	3	9	3	3	9
clpl	2	2	4	2	2	4
clpl	6	6	36	6	6	15
clpl	5	5	25	5	5	12
newtag	8	4	32	3	6	15
dc1	4	4	16	3	3	9
dc1	2	3	6	3	2	6
dc1	4	4	16	3	4	12
dc1	4	5	20	4	3	9
dc1	3	3	9	2	3	6
misex1	2	5	10	4	2	6
misex1	5	7	35	4	4	12
misex1	5	8	40	5	4	12
misex1	4	7	28	5	3	9
misex1	5	5	25	4	4	12
misex1	6	7	42	4	4	12
misex1	5	7	35	4	3	9
ex5	1	3	3	3	1	3
ex5	1	5	5	5	1	5
ex5	1	4	4	4	1	4
ex5	1	7	7	7	1	7
ex5	1	8	8	8	1	8
ex5	1	6	6	6	1	6
ex5	8	4	33	3	6	15
ex5	10	4	40	3	8	20
ex5	7	3	21	3	7	20
ex5	7	3	21	3	6	15
ex5	8	2	16	2	8	16
ex5	9	4	36	3	8	20
ex5	8	2	16	2	7	14
ex5	12	6	72	4	7	20
ex5	14	8	112	4	7	20
ex5	7	2	14	2	7	14
ex5	6	3	18	3	6	15
ex5	6	2	12	2	6	12
ex5	10	7	70	3	7	20
ex5	6	6	36	3	6	15
ex5	12	10	120	4	8	20
ex5	14	8	112	5	7	20
ex5	8	5	40	3	7	20
ex5	10	8	80	3	7	20
ex5	12	7	84	4	7	20
ex5	9	3	27	3	8	20
ex5	5	2	10	2	5	10
b12	4	6	24	4	3	9
b12	7	5	35	4	4	12
b12	7	6	42	5	4	12
b12	4	2	8	2	2	4
b12	4	2	8	2	4	8
b12	5	1	5	1	5	5
b12	9	6	54	6	4	12
b12	6	4	24	4	6	15
b12	7	2	14	2	7	14
newbyte	1	5	5	5	1	5
newapla2	1	6	6	6	1	6
c17	3	3	9	2	3	6
c17	4	2	8	2	2	4
mp2d	11	1	11	1	11	11
mp2d	8	6	48	5	8	20
mp2d	10	5	50	4	10	24
mp2d	6	10	60	9	3	15
mp2d	1	5	5	5	1	5
mp2d	3	6	18	5	2	8
mp2d	1	8	8	8	1	8
mp2d	5	1	5	1	5	5

TABLE 2: Proposed lattice sizes and lower bounds on the lattice sizes for the output functions of benchmark circuits.

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 [12].)

A 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 lattices with *random connectivity*. We have been exploring methods based on the principle of *percolation* [13].

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 [14].

Acknowledgments

We would like to thank Ivo Rosenberg for his contributions.

References

[1] M. Altun and M. D. Riedel, "Lattice-based computation of Boolean functions," in *Design Automation Conference*, 2010, pp. 609–612.

[2] C. E. Shannon, "A symbolic analysis of relay and switching circuits," *Transactions of the AIEE*, vol. 57, pp. 713–723, 1938.

[3] A. DeHon, "Nanowire-based programmable architectures," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 1, no. 2, pp. 109–162, 2005.

[4] Y. Luo, C. P. Collier, J. O. Jeppesen, K. A. Nielsen *et al.*, "Two-dimensional molecular electronics circuits," *ChemPhysChem*, vol. 3, no. 6, pp. 519–525, 2002.

[5] A. Khitun, M. Bao, and K. L. Wang, "Spin wave magnetic nanofabric: A new approach to spin-based logic circuitry," *IEEE Transactions on Magnetics*, vol. 44, no. 9, pp. 2141–2152, 2008.

[6] Y. Zomaya, "Molecular and nanoscale computing and technology," in *Handbook of Nature-Inspired and Innovative Computing*. Springer, 2006, ch. 14, pp. 478–520.

[7] M. L. Fredman and L. Khachiyan, "On the complexity of dualization of monotone disjunctive normal forms," *Journal of Algorithms*, vol. 21, no. 3, pp. 618–628, 1996.

[8] T. Ibaraki and T. Kameda, "A theory of coteries: Mutual exclusion in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 7, pp. 779–794, 1993.

[9] B. from the 1991 International Workshop on Logic Synthesis available at http://www.cbl.ncsu.edu:16080/benchmarks/LGSynth93/.

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

[11] R. E. Bryant, "A switch-level model and simulator for MOS digital systems," *IEEE Transactions on Computers*, vol. 33, no. 2, pp. 160–177, 1984.

[12] I. Wegener, "Lower bounds on circuit complexity," in *The Complexity of Boolean Functions*. John Wiley & Sons, 1987, ch. 5, pp. 119–144.

[13] M. Altun, M. D. Riedel, and C. Neuhauser, "Nanoscale digital computation through percolation," in *Design Automation Conference*, 2009, pp. 615–616.

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