Lattice-Based Computation of Boolean Functions

Mustafa Altun and Marc D. Riedel
Department of Electrical and Computer Engineering
University of Minnesota, Twin Cities
{altu0006, mriedel}@umn.edu

ABSTRACT

This paper studies the implementation of Boolean functions with lattices of two-dimensional switches. Each switch is controlled by a Boolean literal. If the literal is 1, the switch is connected to its four neighbours; else it is not connected. Boolean functions are implemented in terms of connectivity across the lattice: a Boolean function evaluates to 1 iff there exists a top-to-bottom path. The paper addresses the following synthesis problem: how should we map literals to switches in a lattice in order to implement a given target Boolean function? We seek to minimize the number of switches. Also, we aim for an efficient algorithm – one that does not exhaustively enumerate paths. We exploit the concept of lattice and Boolean function **duality**. We demonstrate a synthesis method that produces lattices with a number of switches that grows linearly with the number of product terms in the function. Our algorithm runs in time that grows polynomially.

Categories and Subject Descriptors

B.7.1 [**Integrating Circuits**]: Types and Design Styles—*Advanced technologies*

General Terms: Theory

Keywords: Boolean Functions, Switching Circuits, Lattices, Lattice Duality

1. INTRODUCTION

One-dimensional (1D) switch-based models of computation were first considered by Shannon [1]. An example of a 1D switch is shown in in the top part of Figure 1. It is either ON (i.e., closed) or OFF (i.e., open). Shannon showed how Boolean functions can be implemented with 1D switches in series/parallel configurations.

In this paper we develop a synthesis method for two-dimensional (2D) switch-based models of computation. An example of a 2D switch is shown in the bottom part of Figure 1. It has four ends that are all either mutually connected (ON) or disconnected (OFF). A network consisting of 2D switches is shown in Figure 2(a). The corresponding lattice form is shown Figure 2(b). Here black and white squares represent ON and OFF switches, respectively. Throughout this paper, we will use the lattice representation and focus on top-to-bottom and left-to-right connectivities.

Suppose that we are asked to implement the Boolean function $f = x_1x_2x_3+x_1x_4$. With 1D switches, this task is easily achieved: we implement the function with a series/parallel network, consisting of two parallel paths, one for each product term.

With 2D switches, we implement the computation in terms of the top-to-bottom connectivity of a lattice. Here the task is not as straightforward. Figure 3 shows two solutions, only one of which is correct. The Boolean functions between the top and bottom plates are different for (a) and (b). In (a) the function is $f = x_1x_2+x_1x_4$.

DAC'2010, June 13–18, 2010, Anaheim, California, USA.

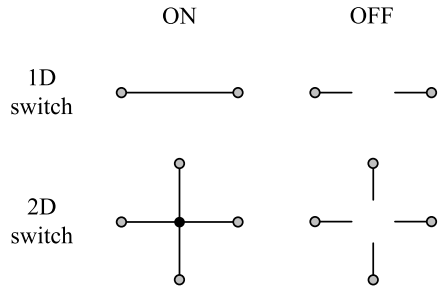

Figure 1: One-dimensional and two-dimensional switches.

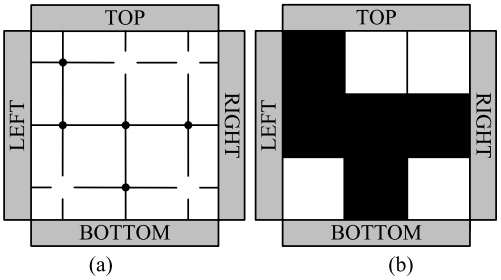

Figure 2: 3×3 2D-switching network and its lattice form.

In (b) the function is $f = x_1x_2x_3 + x_1x_4$. The reason for this difference is that we cannot just consider the column paths; we must consider all possible paths. These include the paths shown in red and blue. Since the path in (a) shown in red covers $x_1x_2x_3$, we conclude that here $f = x_1x_2 + x_1x_4$.

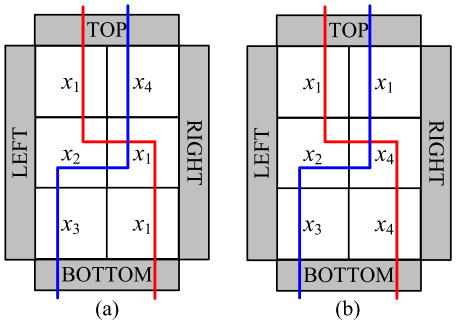

Figure 3: 3×2 lattices with different Boolean functions.

Motivated by this simple example, we consider the general problem of implementing Boolean functions with lattices of 2D switches. We seek to minimize the number of switches. Also, we aim for an efficient algorithm. In our synthesis strategy, we exploit the concept of lattice and Boolean function duality [2, 3]. This forms a novel and rich framework for Boolean computation.

1.1 Definitions

Consider k independent **Boolean variables**, $x_1, x_2, \ldots, x_k$. **Boolean literals** are Boolean variables and their complements, i.e., $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_k, \bar{x}_k$. 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 expression (SOP)** corresponds to an OR of product terms.

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. 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)**.

f and g are **dual Boolean functions** iff

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

A dual of a function can also be obtained by interchanging AND and OR operations as well as the constants 0 and 1. For example, if $f = x_1x_2 + \bar{x}_1x_3$ then $f^D = (x_1 + x_2)(\bar{x}_1 + x_3)$. Another trivial example is that for $f = 1$ the dual is $f^D = 0$.

2. SYNTHESIZING LATTICE-BASED COMPUTATION

In our approach, the input Boolean literals are applied to sites of the lattice. Each site is a 2D switch that is ON (OFF) if the corresponding input literal is 1 (0). Call the Boolean functions that are implemented according to the top-to-bottom and left-to-right plate connectivities f_L and g_L, respectively.

Note that the Boolean functions f_L and g_L are the OR of all top-to-bottom and left-to-right paths, respectively. Since each path corresponds to the AND of inputs, the paths taken together correspond to the OR of these AND terms, so implement a sum-of-products expression.

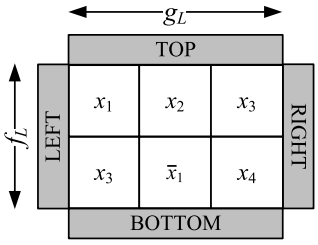

Figure 4: 2×3 lattice with assigned literals.

Consider the example shown in Figure 4: here there are 6 switches, 3 top-to-bottom paths and 4 left-to-right paths. So f_L is the OR of the 3 products $x_1x_3, \bar{x}_1x_2, x_3x_4$ and g_L is the OR of the 4 products $x_1x_2x_3, x_1\bar{x}_1x_2x_4, \bar{x}_1x_2x_3x_3, \bar{x}_1x_3x_4$. As a result, $f_L = x_1x_3 + \bar{x}_1x_2 + x_3x_4$ and $g_L = \bar{x}_1x_3x_4 + x_2x_3$ (both in MSOP form).

This paper address the following logic synthesis problem: given a target Boolean function f_T, how should we map literals to the sites in a lattice such that $f_L = f_T$?

2.1 Mathematical Preliminaries

We propose two useful theorems. The first pertains to lattice functions and their duality relation. Suppose we are given a lattice and asked to compute f_L and g_L. As suggested above, f_L and g_L are obtained by OR'ing all top-to-bottom and left-to-right paths, respectively. So we can compute f_L and g_L by examining all possible paths. Unfortunately, enumerating paths quickly becomes prohibitive as the size of the lattice grows. Indeed, the number of paths grows exponentially. For instance, a 3×3 lattice has 9 top-to-bottom paths, compared to 2 in a 2×2 lattice. The following theorem suggests a different approach.

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 1 *We analyze the two lattices shown in Figure 5.*

Lattice (a)*: The top-to-bottom paths shown by red lines implement a Boolean function $f = x_1x_2 + \bar{x}_1x_3$. The left-to-right paths represented by blue lines implement a Boolean function $g = x_1x_3 + \bar{x}_1x_2$. 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 obtained 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, $f_L = x_1\bar{x}_1 + x_1x_2 + x_1\bar{x}_1x_2x_3 + x_1\bar{x}_1x_2x_3 + x_2x_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_1x_2x_3 + x_2x_3 + \bar{x}_1x_3$ which is equal to $x_1x_2 + \bar{x}_1x_3$. Since there are 9 total left-to-right paths, $g_L = x_1x_3 + x_1x_2x_3 + x_1\bar{x}_1x_2x_3 + x_1x_2x_3 + x_1x_2x_3 + x_1\bar{x}_1x_2 + \bar{x}_1x_2x_3 + \bar{x}_1x_2x_3 + \bar{x}_1x_2$ which is equal to $x_1x_3 + \bar{x}_1x_2$. So Theorem 1 holds true for this example.*

Lattice (b)*: The top-to-bottom paths shown by red lines implement a Boolean function $f = x_1x_2 + x_2x_3$. The left-to-right paths represented by blue lines implement a Boolean function $g = x_1x_3 + x_2$. Since $g = f^D$, we can apply Theorem 1: $f_L = f = x_1x_2 + x_2x_3$ and $g_L = f^D = x_1x_3 + x_2$. Again, we can confirm that the theorem holds.*

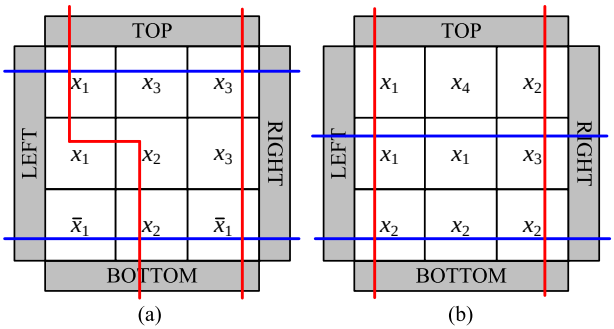

Figure 5: Examples to illustrate Theorem 1.

PROOF. 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, i.e., $f_L = 0$. Thus, we conclude that $f_L = f$. Following the same argument for g, we conclude that $g_L = f^D$. □

Note that Theorem 1 serves not only for analyzing lattice functions; it also provides a constructive method for synthesizing lattices with the requisite property, namely that the top-to-bottom and left-to-right functions are duals. In particular, the theorem suggests that we need only consider paths that are straight lines, i.e., columns and rows, in order to implement a target Boolean function. Before describing our approach, we present some useful properties of dual functions.

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

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

where the P's and P''s are prime implicants.[1] We will use set representations for the prime implicants:

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

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

where each SP_i is the set of literals in the corresponding P_i and each SP'_j is the set of literals in the corresponding 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 [2]:

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

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

PROOF. The proof is by contradiction. Suppose that we focus on one product term 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 term 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 were previously assigned 0. If follows that $f^D = 1$, a contradiction. □

Theorem 2 *Assume f and f^D are in ISOP form. For any product term 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 z_i single-element disjoint sets that each represents one of the 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 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 2 *Suppose we are given a target function f_T and its dual f_T^D in ISOP form such that*

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

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 3 *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's 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. The proof is by contradiction. Consider a product term 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 one. This is a contradiction. □

2.2 Mapping Boolean Functions onto Lattices

Based on these mathematical preliminaries, we turn to the task of mapping input literals onto a lattice in order to implement a target function. Suppose that we are given a target Boolean function f_T and its dual f_T^D, both in ISOP form,

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

where each P and P' is a prime implicant.

Mapping problem: How should we map the literals of a given Boolean function onto sites of the lattice such that $f_L = f_T$ and $g_L = f_T^D$?

We propose a solution that 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. The time complexity of computing this solution is polynomial in n and m: $O(n^2m^2)$.

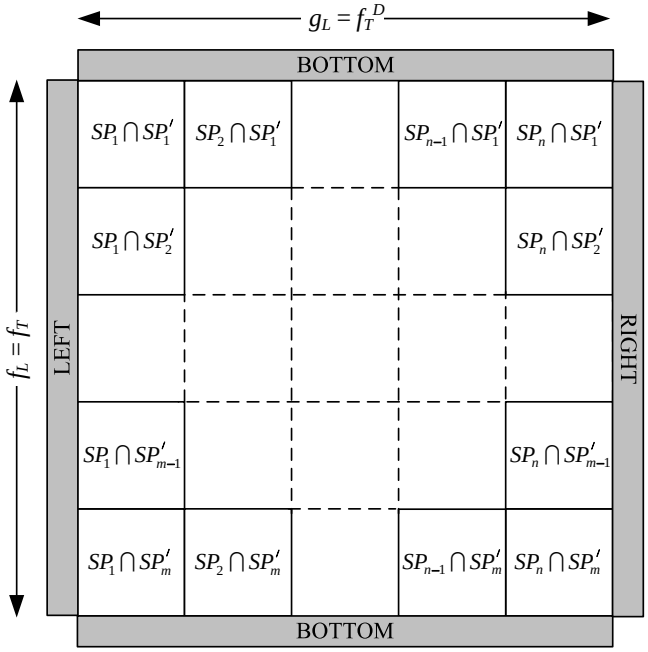

Figure 6: Proposed mapping technique.

The proposed technique is illustrated in Figure 6. As shown in the figure, each site has its own intersection set. After computing these intersection sets, we select an arbitrary literal from each set and assign this literal to the corresponding lattice site. This implements the requisite target function f_T.

Example 4 *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:

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

We have:

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

Figure 7 shows the implementation of the target function. Grey sites represent sets having more than one literal. Selection of the final literal for grey sites is arbitrary, e.g., 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 and its dual, we only use paths of columns and rows. Thus all other paths must be redundant. For example, there are a total of 9 top-to-bottom paths: the 3 column-paths and

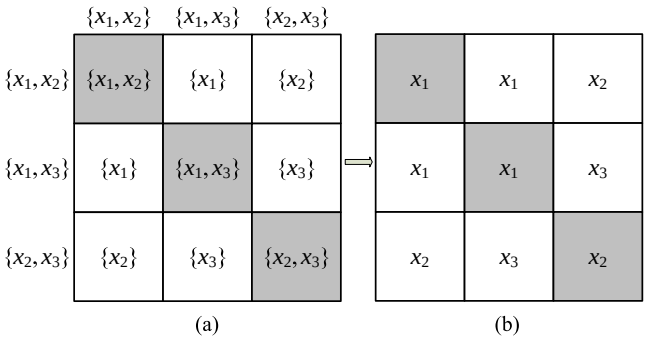

Figure 7: Mapping $f_T = x_1x_2 + x_1x_3 + x_2x_3$. (a): Lattice sites with corresponding sets. (b): Lattice sites with corresponding literals.

6 more paths; however the Boolean function implemented by the 6 non-column-paths is covered by the column-paths. The lattice implements $f_L = f_T = x_1x_2 + x_1x_3 + x_2x_3$ and $g_L = f_T^D = x_1x_2 + x_1x_3 + x_2x_3$.

Below is the detailed explanation and proof of correctness for the proposed mapping technique.

Mapping technique:

1. Begin with f_T and its dual f_T^D in ISOP form. Suppose that f_T and f_T^D have n and m product terms, respectively.
2. Start with an $m \times n$ lattice. Implement every product term of f_T by a column-path and every product term of f_T^D by a row-path. Theorem 1 allows us to do this.
3. Compute intersection sets for every site, as shown in Figure 6.
4. From Lemma 1 we know that all the intersection sets are non-empty. If the intersection set consists of a single literal, then we map it to the corresponding site. If the intersection set has more than one literal, then the question arises: which one should we use? Here we exploit Theorem 2. It says that the intersection sets of a product term include single-element sets for all of its literals. So the corresponding site can be any of them. We pick arbitrarily. We obtain a lattice in which columns and rows implement product terms of f_T and f_T^D, respectively.

We give another example, this one somewhat more complicated.

Example 5 *Suppose that f_T and f_T^D are given in ISOP form:*

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

Figure 8 shows the implementation of the target function. Grey

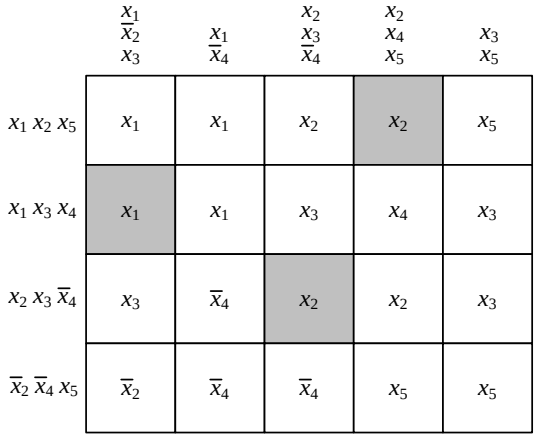

Figure 8: Mapping $f_T = x_1\bar{x}_2x_3 + x_1\bar{x}_4 + x_2x_2\bar{x}_4 + x_2x_4x_5 + x_3x_5$.

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 $g_L = f_T^D = x_1x_2x_5 + x_1x_3x_4 + x_2x_3\bar{x}_4 + \bar{x}_2\bar{x}_4x_5$.

2.3 Experimental results

We report synthesis results for some benchmark circuits in Figure 9. We used the usual suspects, namely the Espresso and LGSynth93 collections. We selected circuits based on size and suitability. We considered each output as a separate Boolean function. We obtained MSOPs for the Boolean functions and their duals using the Berkeley SIS environment (using Espresso minimization) [4]. Here n and m are the number of products of the corresponding Boolean function and its dual, respectively, in MSOP form.

Circuit	n	m	Lattice size	Circuit	n	m	Lattice size
C17	3	3	9	rd53	16	16	256
C17	4	2	8	rd53	10	10	100
9symm	84	72	6048	rd53	5	10	50
alu2	4	4	16	rd73	64	64	4096
alu2	33	33	1089	rd73	35	35	1225
alu2	67	68	4556	rd73	42	42	1764
alu2	2	2	4	rd84	128	128	16384
alu2	1	2	2	rd84	1	8	8
alu2	36	37	1332	rd84	84	79	6636
clip	42	42	1764	rd84	70	57	3990
clip	21	21	441	cm85a	16	24	38
clip	24	24	576	cm85a	16	9	144
clip	20	20	400	cordic	143	774	110682
clip	31	31	961	cordic	771	149	114879

Figure 9: Lattice sizes for the output functions of benchmark circuits.

3. DISCUSSION

Our method produces lattice sizes that are linear in the number of product terms of the target Boolean function. The time complexity of our algorithm is polynomial in the number of product terms. We do not claim that our method always produces the optimal lattice size for every target function, but it performs well in this regard.

This work is related to our prior work on synthesizing robust digital computation in lattices with random connectivity based on the phenomenon of percolation [5].

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 mutually dual. This is one of the few problems in circuit complexity whose precise tractability status is unknown [6].

Acknowledgments: We would like to thank Ivo Rosenberg for his contributions.

4. REFERENCES

[1] C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Transactions of the AIEE*, vol. 57, pp. 713–723, 1938.

[2] 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.

[3] 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.

[4] E. M. Sentovich *et al.*, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep., 1992.

[5] M. Altun, M. Riedel, and C. Neuhauser, "Nanoscale Digital Computation through Percolation," in *DAC'09*, pp. 615–616.

[6] 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.