Synthesizing Logic with Percolation in Nanoscale Lattices

MUSTAFA ALTUN, University of Minnesota
MARC D. RIEDEL, University of Minnesota

This paper proposes a novel probabilistic framework for digital computation with lattices of nanoscale switches based on the mathematical phenomenon of percolation. With random connectivity, percolation gives rise to a sharp non-linearity in the probability of global connectivity as a function of the probability of local connectivity. This phenomenon is exploited to compute Boolean functions robustly, in the presence of defects. It is shown that the *margins*, defined in terms of the steepness of the non-linearity, translate into the degree of defect tolerance. Achieving good margins entails a mapping problem. Given a target Boolean function, the problem is how to assign literals to regions of the lattice such that there are no diagonal paths of 1's in any assignment that evaluates to 0. Assignments with such paths result in poor error margins due to stray, random connections that can form across the diagonal. A necessary and sufficient condition is formulated for a mapping strategy that preserves good margins: the top-to-bottom and left-to-right connectivity functions across the lattice must be *dual* functions. Based on lattice duality, an efficient algorithm to perform the mapping is proposed. The algorithm optimizes the lattice area while meeting prescribed worst-case margins. Its effectiveness is demonstrated on benchmark circuits.

Categories and Subject Descriptors: J.6 [**Computer-Aided Engineering**]: Computer-Aided Design (CAD)

General Terms: Design, Reliability

Additional Key Words and Phrases: Percolation, defect tolerance, duality, Boolean functions

1. INTRODUCTION

As current CMOS-based technology is approaching its anticipated limits, research is shifting to novel forms of nanoscale technologies including molecular-scale self-assembled systems [Whitesides and Grzybowski 2002; Yan et al. 2003]. Unlike conventional CMOS that can be patterned in complex ways with lithography, self-assembled systems generally consist of regular structures such as crossbar arrays [Ziegler and Stan 2003; Zomaya 2006]. In particular, with self-assembly, nanoscale technologies are often characterized by high defect rates. A variety of techniques have been proposed for mitigating against defects [Huang et al. 2004; Kuekes et al. 2005; Sun and Zhang 2006; Hogg and Snider 2007; Snider and Williams 2007].

In prior work, we discussed strategies for implementing Boolean functions with lattices of four-terminal switches [Altun and Riedel 2010; 2011]. We addressed the synthesis problem of how best to assign literals to switches in a lattice in order to implement a given target Boolean function, with the goal of minimizing the lattice size, measured in terms of the number of switches. We presented an efficient synthesis algorithm for this task. The algorithm has polynomial time complexity (significantly, it does not exhaustively enumerate paths). It produces lattices with a size that grows linearly with the number of products of the target Boolean function.

In this paper, we address the problem of implementing Boolean functions with lattices of four-terminal switches in the presence of defects. We assume that such defects occur probabilistically. Although not tied to any particular technology, our model could be applicable for emerging technologies such as nanowire crossbar arrays [Cui and Lieber 2001] and magnetic switch-based structures [Khitun et al. 2008].

Our approach is predicated on the mathematical phenomenon of percolation. With random connectivity, percolation gives rise to a sharp non-linearity in the probability of global connectivity as a function of the probability of local connectivity. We exploit this phenomenon to compute Boolean functions robustly, within prescribed error margins.

• This work is supported by an NSF CAREER award 0845650.

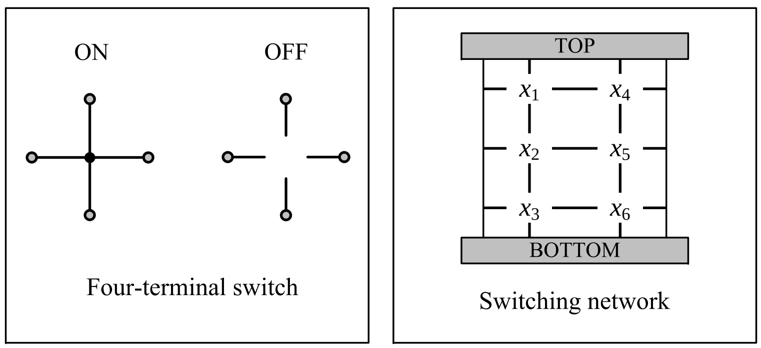
• A preliminary version of this paper appeared in [Altun et al. 2009].

• Author's addresses: M. Altun and M. Riedel, Electrical and Computer Engineering, University of Minnesota, 200 Union St S.E., Minneapolis, Minnesota 55455.

The paper is organized as follows. In Section 1.1, we discuss our circuit model. In Section 1.2, we discuss our defect model. In Section 1.3, we discuss the mathematics of percolation and how this phenomenon can be exploited for tolerating defects. In Section 2, we discuss potential technologies that fit our model. In Section 3, we present our main technical result: a method for assigning Boolean literals to sites in a switching lattice that optimizes the lattice area while meeting prescribed defect tolerances. In Section 4, we evaluate our method on benchmark circuits.

1.1. Circuit Model

Our circuit model consists of regular two-dimensional arrays of four-terminal switches. A four-terminal switch is shown in the top part of Figure 1. It has two states, ON and OFF, that are controlled by a Boolean literal. If the literal takes the value 1 then the four ends of the switch are mutually connected – the switch is ON. If the literal takes the value 0 then the four ends of the switch are mutually disconnected – the switch is OFF. A network of four-terminal switches is shown in Figure 1(b). The Boolean function for the network evaluates to 1 iff there is a closed path between the top and bottom plates of the lattice. It can be computed by taking the sum (OR) of the product (AND) of literals along each path. These paths are $x_1 - x_2 - x_3$, $x_1 - x_2 - x_5 - x_6$, $x_4 - x_5 - x_2 - x_3$, and $x_4 - x_5 - x_6$.



(a) (b)

Fig. 1. (a): Four-terminal switch with its ON and OFF states. (b): Four-terminal switching network implementing the Boolean function $x_1x_2x_3 + x_1x_2x_5x_6 + x_2x_3x_4x_5 + x_4x_5x_6$.

1.2. Defects and Defect Tolerance

We assume that defects cause switches to fail in one of two ways: they are ON when they are supposed to be OFF, i.e., the controlling literal is 0; or they are OFF when they are supposed to be ON, i.e., the controlling literal is 1. We assume that switches can fail in one of these two ways, or both. As we discuss in Section 3.2, we allow for different defect rates in both directions, ON-to-OFF and OFF-to-ON. Crucially, we assume that all switches fail with *independent* probability.

Defective switches can ruin the Boolean computation performed by a network. Consider the networks shown in Figure 2. The network in Figure 2(a) consists of a single switch. The networks in Figure 2(b) and Figure 2(c) consist of a pair of switches in series and in parallel, respectively. All switches are controlled by the literal x_1. Obviously, in each of these networks, the top and bottom plates are connected when $x_1 = 1$ and disconnected when $x_1 = 0$. Therefore they implement the function $f = x_1$.

Note that the three networks are not identical in their defect-tolerance capability. Suppose that exactly one switch in each network is defective when $x_1 = 1$ and exactly one is defective when $x_1 = 0$. When $x_1 = 1$, the networks in Figure 2(a) and Figure 2(b) compute the wrong value of $f = 0$; however, the network in Figure 2(c) computes the correct value $f = 1$. Similarly, when

$x_1 = 0$, the networks in Figure 2(a) and Figure 2(c) compute the wrong value of $f = 1$. However, the network in Figure 2(b) computes the correct value of $f = 0$. So the series and parallel networks in Figures 2(b) and 2(c) each tolerate up to one defective switch, but they tolerate different defect types. None of these networks tolerates defects for both cases $x_1 = 1$ and $x_1 = 0$.

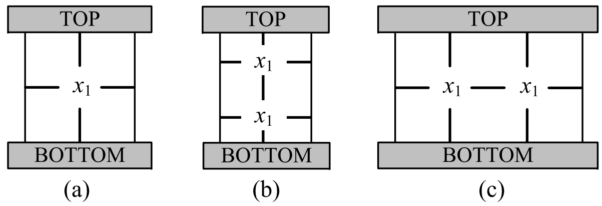


Fig. 2. Switching networks

Now consider the network in Figure 3. Compared to the networks in Figure 2, it has more switches. We expect that it will be superior in terms of its defect tolerance, for both the cases $x_1 = 1$ and $x_1 = 0$. But what is the relationship between the amount of redundancy and the defect tolerance that is achieved? As we discuss in Section 1.3, the relationship is non-linear. The explanation hinges on percolation.

Throughout the rest of the paper, we will use a lattice representation. White and black sites represent OFF and ON switches, respectively. If $x_1 = 1$, each four-terminal switch is ideally ON and represented by a black site. If $x_1 = 0$, each four-terminal switch is ideally OFF and represented by a white site. Due to defects, not all switches will behave in this way. Defective switches are represented by white and black sites while the switch is supposed to be ON and OFF, respectively. This is illustrated in Figure 3. Note that in spite of defects, the network in Figure 3 computes correctly for both the cases $x_1 = 0$ and $x_1 = 1$.

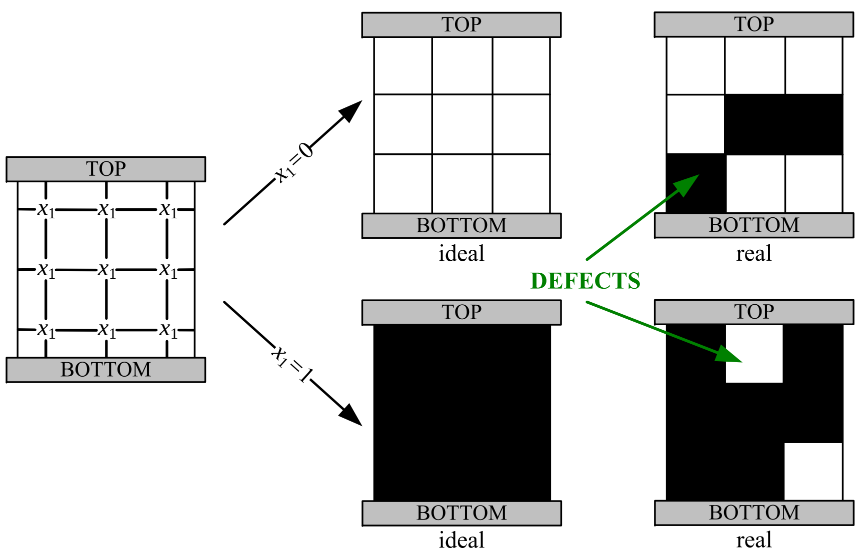


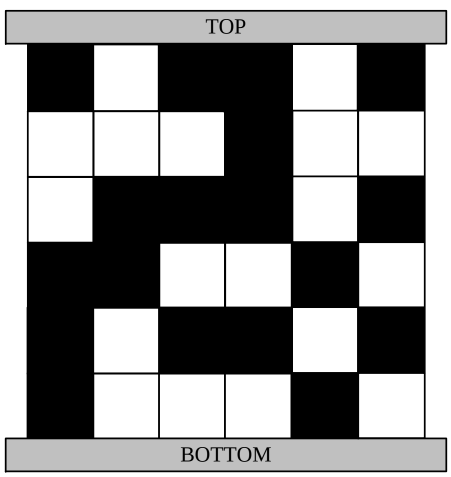
Fig. 3. Switching network with defects.

1.3. Percolation

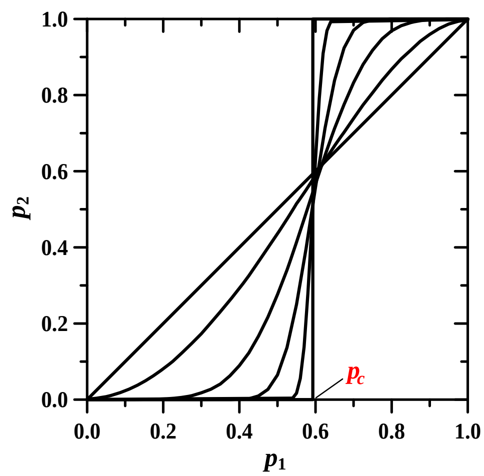
Percolation theory is a rich mathematical topic that forms the basis of explanations of physical phenomena such as diffusion and phase changes in materials. It tells us that in media with random local connectivity, there is a critical threshold for global connectivity: below the threshold, the probability of global connectivity quickly drops to zero; above it, the probability quickly rises to one.

Broadbent and Hammersley described percolation with the following metaphorical model [Broadbent and Hammersley 1957]. Suppose that water is poured on top of a large porous rock. Will the water find its way through holes in the rock to reach the bottom? We can model the rock as a collection of small regions each of which is either a hole or not a hole. Suppose that each region is a hole with independent probability p_1 and not a hole with probability $1 - p_1$. The theory tells us that if p_1 is above a critical value p_c, the water will always reach the bottom; if p_1 is below p_c, the water will never reach the bottom. The transition in the probability of water reaching bottom as a function of increasing p_1 is extremely abrupt. For an infinite size rock, it is a step function from 0 to 1 at p_c.

In two dimensions, percolation theory can be studied with a lattice, as shown in Figure 4(a). Here each site is black with probability p_1 and white with probability $1 - p_1$. Let p_2 be the probability that a connected path of black sites exists between the top and bottom plates. Figure 4(b) shows the relationship between p_1 and p_2 for different square lattice sizes. Percolation theory tells us that with increasing lattice size, the steepness of the curve increases. (In the limit, an infinite lattice produces a perfect step function.) Below the critical probability p_c, p_2 is approximately 0 and above it p_2 is approximately 1.



(a)



(b)

Fig. 4. (a): Percolation lattice with random connections; there is a path of black sites between the top and bottom plates. (b) p_2 versus p_1 for 1×1, 2×2, 6×6, 24×24, 120×120, and infinite-size lattices.

Suppose that each site of a percolation lattice is a four-terminal switch controlled by the same literal x_1. Also suppose that each switch is independently defective with the same probability. Defective switches are represented by white and black sites while the switch is supposed to be ON and OFF, respectively. Let's analyze the cases $x_1 = 0$ and $x_1 = 1$. If $x_1 = 0$ then each site is black with the defect probability, and the defective black sites might cause an error by forming a path between the top and bottom plates. In this case, p_1 and p_2 described in the percolation model correspond to the defect probability and the probability of an error in top-to-bottom connectivity, respectively.

If $x_1 = 1$ then each site is white with the defect probability and the defective white sites might cause an error by destroying the connection between the top and bottom plates. In this case, p_1 and p_2 in the percolation model correspond to $1-$(defect probability) and $1-$(probability of an error in top-to-bottom connectivity), respectively. The relationship between p_1 and p_2 is shown in Figure 5.

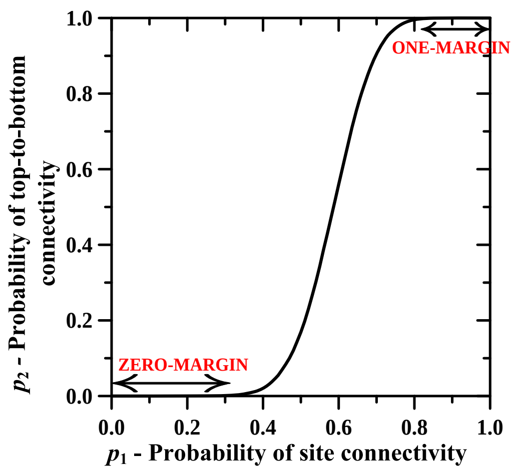


Fig. 5. Non-linearity through percolation in random media.

1.4. Definitions

Throughout the paper, we use the concept of defect *probability* and defect *rate* interchangeably. We assume that the lattice is large enough for this to hold true.

Definition 1 *We define the* ***one margin*** *and* ***zero margin*** *to be the ranges of p_1 for which we interpret p_2 as unequivocally 1 and 0, respectively.*

The percolation curve shown in Figure 5 tells us that unless the defect probability exceeds a zero margin (one margin), we achieve **robust connectivity**: the top and bottom plates remain disconnected (connected) with high probability. Therefore the one margin and zero margin are the indicators of defect tolerance while the switch is ON and OFF, respectively. In other words, the margins are the maximum defect probabilities (rates) that can be tolerated. For example, suppose that a network has 5% zero and one margins. This means that the network will successfully tolerate defects unless the defect probability (rate) exceed 5%.

What follows are some standard definitions from the field of logic synthesis. We will use these terms in Sections 3.

Definition 2 *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 3 *A* **product (P)** *is an AND of literals, e.g., $P = x_1\bar{x}_3x_4$. A* **sum-of-products (SOP)** *expression corresponds to an OR of products.*

Definition 4 *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 5 *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. Among the SOPs for f, one with the minimum number of products is a* **minimum sum-of-products (MSOP)** *expression.*

Definition 6 *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 also can 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. APPLICABLE TECHNOLOGIES

The main contributions of this paper are conceptual. Our circuit and defect models are simple and broadly applicable to different types of emerging technologies. A schematic for the realization of our circuit model is shown in Figure 6. 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 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. One can sense the output 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 this circuit model.

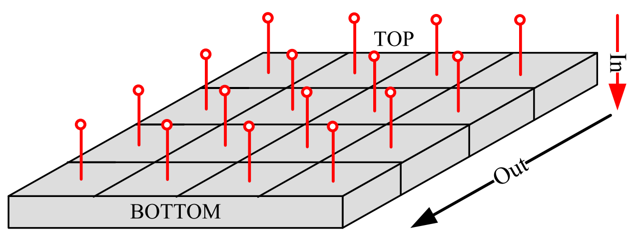


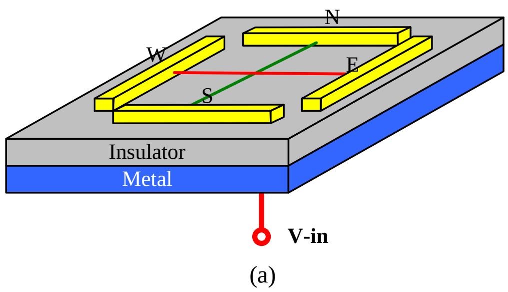
Fig. 6. 3D realization of our circuit model with the inputs and the output.

In their seminal work, Yi Cui and Charles Lieber investigated crossbar structures for different types of nanowires including *n*-type and *p*-type nanowires [Cui and Lieber 2001]. 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.

Cui and Lieber implemented a four-terminal device with crossed *n*- and *p*-type nanowires, illustrated in Figure 7(a). The device works as follows. When a high V-in is applied, a *p*-type nanowire (green) behaves like a short circuit, so the N and S terminals are connected, and an *n*-type nanowire (red) behaves like an open circuit, so the W and E terminals are disconnected. When a low V-in is applied, a *p*-type nanowire behaves like an open circuit, so the N and S terminals are disconnected, and an *n*-type nanowire behaves like a short circuit, so the W and E terminals are connected.

One could easily implement a four-terminal switch with similar techniques, as illustrated in Figure 7(b). Here the switch has crossed p-type nanowires. When a high V-in is applied, the nanowires behave like short circuits. Also a resistor-like junction is formed between them, meaning that the nanowires are connected through a low-valued resistor. Thus, all the four-terminals are connected; the switch is ON. When a low V-in is applied, the nanowires behave like open circuits. Thus, all the four-terminals are disconnected; the switch is OFF. The result is a four-terminal switch that corresponds to our model.



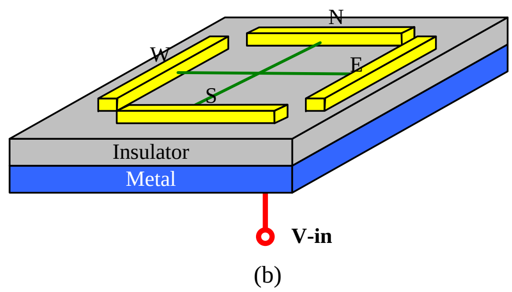


Fig. 7. Nanowire four-terminal switch.

Nanowire switches, of course, would be assembled in large arrays. Indeed, the impetus for nanowire-based technology is the potential density, scalability and manufacturability [Huang et al. 2001; Luo et al. 2002; DeHon 2005]. 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 Figure 8. Here the black and white sites represent crosspoints that are ON and OFF, respectively.

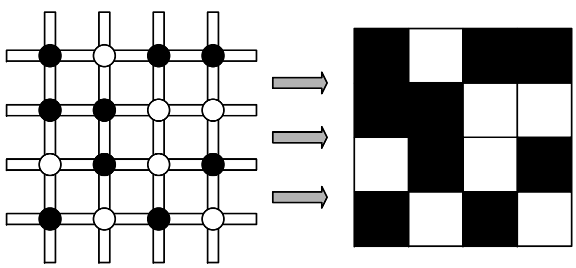


Fig. 8. Nanowire crossbar array with random connections and its lattice representation.

Many other novel and emerging technologies fit the general model of four-terminal switches. For instance, researcher are investigating *spin waves* [Eshaghian-Wilner et al. 2006]. 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 Figure 9. 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 the four-terminals are connected. In the OFF state the switch reflects any incoming spin waves; all the four-terminals are disconnected. Spin-wave switches, like nanowire switches, are also configured in crossbar networks [Khitun et al. 2008].

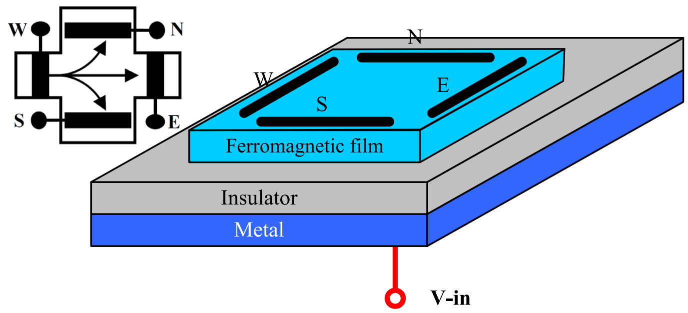


Fig. 9. Spin-wave switch.

3. LOGIC SYNTHESIS THROUGH PERCOLATION

We implement Boolean functions with a single lattice of four-terminal switches, as illustrated in Figure 10. There are $R \times C$ regions $r_{11}, \ldots, r_{RC}$ in the lattice. Each region has $N \times M$ four-terminal switches. We assign Boolean literals $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_k, \bar{x}_k$ to regions as controlling inputs. If an input literal is logic 1 then all switches in the corresponding region are ideally ON; if the literal is logic 0 then all switches in the corresponding region are ideally OFF. This is illustrated in Figure 11.

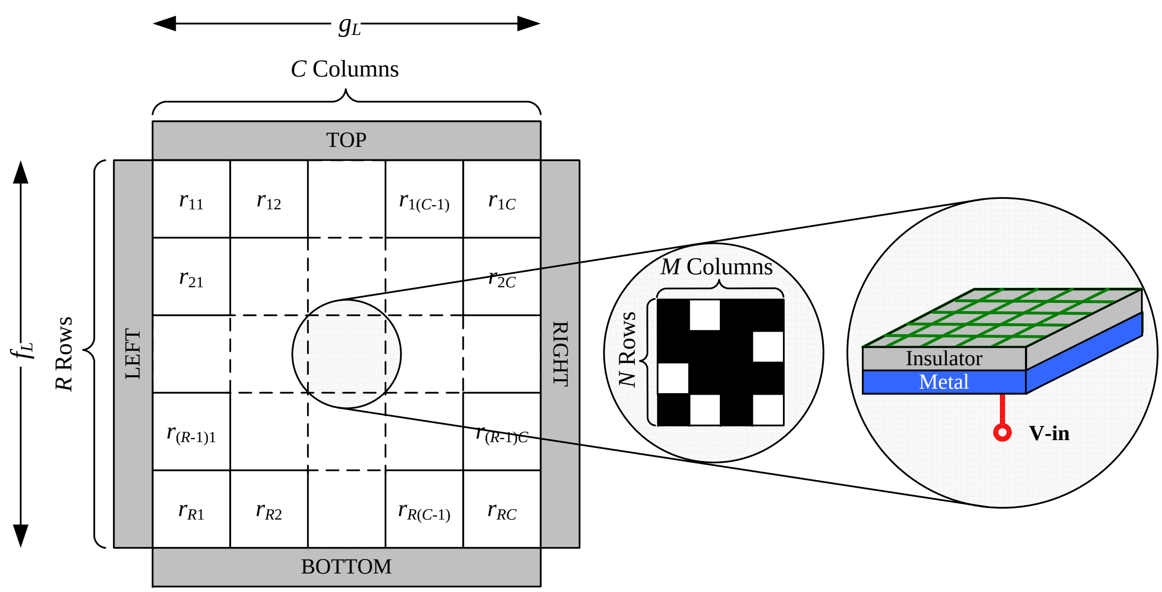


Fig. 10. Boolean computation in a lattice, i.e., each region has $N \times M$ four-terminal switches. Each region can be realized by an $N \times M$ nanowire crossbar array with a controlling voltage V-in.

In our synthesis method, a Boolean function is implemented by a lattice according to the connectivity between the top and bottom plates. For the purpose of elucidating 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. (Note, however, that our design method does not aim to implement separate top-to-bottom and left-to-right functions. As we explain below, f_L and g_L are related.)

As shown in Figure 11, each Boolean function evaluates to 1 if there exists a path between corresponding plates and evaluates to 0 otherwise. Thus, the Boolean functions f_L and g_L can be computed as 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.

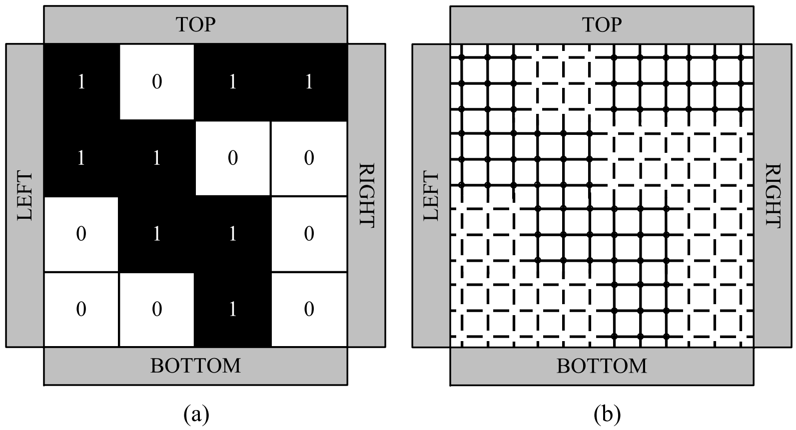


Fig. 11. Relation between Boolean functionality and paths; $f_L = 1$ and $g_L = 0$. (a) Each of the 16 regions is assigned logic 0 or 1; $R = 4$ and $C = 4$. (b) Each region has 9 switches; $N = 3$ and $M = 3$.

Note that the values of N and M do not affect the Boolean functionality between plates; they determine the defect tolerance capability of the lattice. Therefore, for simplicity, let's set $N = 1$ and $M = 1$ while computing the Boolean functions f_L and g_L. In this way, there are fewer paths to count between the corresponding plates. Consider the lattice shown in Figure 12(a): here there are 6 regions each of which is controlled by a Boolean literal. With $N = 1$ and $M = 1$, there are 3 top-to-bottom paths and 4 left-to-right paths, as shown in Figure 12(b). Here 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 $g_L = \bar{x}_1x_3x_4 + x_2x_3$.

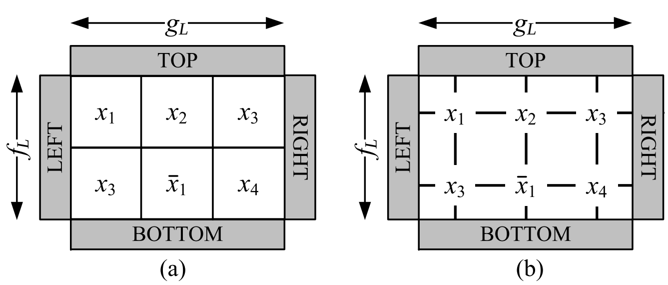


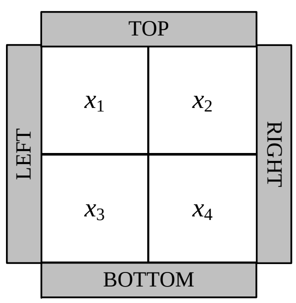
Fig. 12. (a) A lattice with assigned inputs to 6 regions. (b) Switch-based representation of the lattice; $N = 1$ and $M = 1$.

In the following section, we study the robustness of the lattice computation. We investigate the computation, implemented in terms of connectivity across the lattice, in the presence of defects.

3.1. Robustness

An important consideration in synthesis is the quality of the margins, defined in Definition 1. Suppose that the one and zero margins are the ranges of values for p_1 for which p_2 is always above $(1-\epsilon)$ and below ϵ, respectively, where ϵ is a very small number. For what follows, we will use a value $\epsilon = 0.001$. The margins correlate with the degree of defect tolerance. For instance a 10% one margin means that a defect rate of up to 10% can be tolerated while the corresponding Boolean function evaluates to 1. In other words, although each switch is defective with probability 0.1, the circuit still evaluates to 1 with high probability ($p_2 > 0.999$). The higher the margins, the higher the defect tolerance that we achieve.

Different assignments of input variables to the regions of the lattice affect the margins. Consider a 4-input 2×2 lattice shown in Figure 13(a). Suppose that $N = 8$ and $M = 8$ for this lattice. Figure 13(b) shows Boolean functionalities and margins for different input assignments. Since the lattice has 4 input variables x_1, x_2, x_3, x_4 there should be 16 different input assignments. However, there are only 7 rows in the table. Some input assignments produce the same result due symmetries in the lattice: flipping the lattice vertically or horizontally gives us two different input assignments that are identical in terms of margins as well as the Boolean functionality. Note that each margin value in the table corresponds to either a one margin (if the corresponding Boolean function is 1) or a zero margin (if the corresponding Boolean function is 0). We define the **worst-case** one and zero margins to be the minimum one and zero margins of all input assignments. For example, the table shown in Figure 13(b) states that f_L has a 14% worst-case one margin and a 0% worst-case zero margin.



(a)

x_1	x_2	x_3	x_4	f_L	Margin	g_L	Margin
0	0	0	0	0	40%	0	40%
0	0	0	1	0	25%	0	25%
0	1	0	1	1	14%	0	23%
0	0	1	1	0	23%	1	14%
0	1	1	0	0	0%	0	0%
0	1	1	1	1	14%	1	14%
1	1	1	1	1	25%	1	25%

(b)

Fig. 13. (a): A lattice with assigned inputs; $R = 2$ and $C = 2$. (b): Possible 0/1 assignments to the inputs (up to symmetries) and corresponding margins for the lattice ($N = 8, M = 8$).

The row highlighted in grey has very low margins – indeed, these are nearly zero – so the circuit is likely to produce erroneous values for this input combination. Let's examine why. Assignments that evaluate to 0 but have diagonally adjacent assignments of blocks of 1's could be problematic because there is a chance that a weak connection will form through stray, random connections across the diagonal. This is illustrated in Figure 14. In this example, f_L and g_L both evaluate to 0; however the top-to-bottom and left-to-right connectivities evaluate to 1 if a defect occurs around the diagonal 1's. In effect, such defective switches are "shorting" the connection. So in this case f_L and g_L both evaluate to 1, incorrectly.

Note that diagonal paths are only problematic when the corresponding Boolean function evaluates to 0 because the diagonal paths can only cause $0 \rightarrow 1$ errors. If the Boolean function evaluates to 1, these diagonal paths do not cause such an error; at best they strengthen the connection between plates. This is illustrated in Figure 15. In the figure, there are both top-to-bottom and left-to-right diagonal paths shown with red lines. However, only the top-to-bottom diagonal path is destructive because only f_L evaluates to 0 ($g_L = 1$).

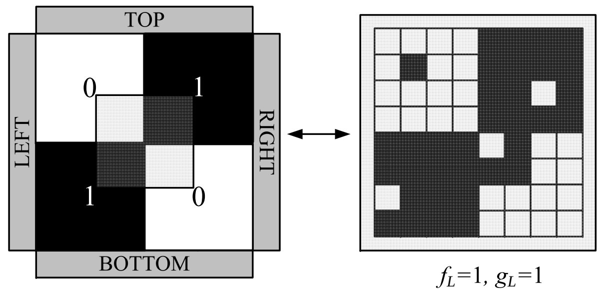


Fig. 14. An input assignment with a low zero margin.

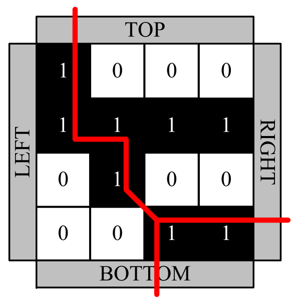


Fig. 15. An input assignment with top-to-bottom and left-to-right diagonal paths.

Definition of Robustness: We call a lattice robust if there is no input assignment for which the top-to-bottom function evaluates to 0 that contains diagonally adjacent 1's.

The following theorem tells us the necessary and sufficient condition for robustness.

Theorem 1 *A lattice is robust iff the top-to-bottom and left-to-right functions* f_L *and* g_L *are dual functions:* $f_L(x_1, x_2, \ldots, x_k) = \bar{g}_L(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k)$.

(See Definition 6 for the meaning of *dual*.)

PROOF. In the proof, we consider two cases, namely $f_L = 1$ and $f_L = 0$.

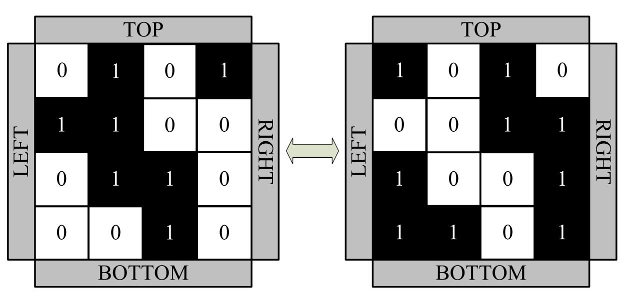


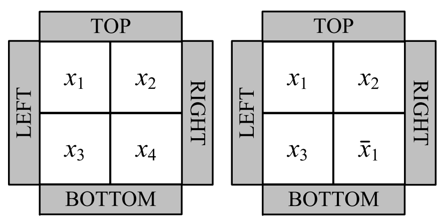
Fig. 16. Illustration of Theorem 1.

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

Case 2: If $f_L(x_1, x_2, \ldots, x_k) = 0$ and there are no diagonally connected top-to-bottom paths, there must be a path of 0's between left and right. If we complement all the inputs, these connected 0's become 1's, i.e., $g_L(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k) = 1$. As a result, $\bar{g}_L(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k) = f_L(x_1, x_2, \ldots, x_k) = 0$

Figure 16 illustrates the two cases. Taken together the two cases prove that for robust computation, f_L and g_L must be dual functions. For both cases it is trivial that we can do the same reasoning in an inverse way: if f_L and g_L are dual functions then every input assignment is robust. $\square$

Example 1 *Consider the lattices shown in Figure 17. For both lattices, $R = 2$ and $C = 2$. Let's analyze the robustness of these two lattices using Theorem 1.*



(a) (b)

Fig. 17. (a) An example of non-robust computation; (b) An example of robust computation.

Example (a): *The Boolean functions implemented by the lattice are $f_L = x_1x_3 + x_2x_4$ and $g_L = x_1x_2 + x_3x_4$. Since $f_L^D = (x_1 + x_3)(x_2 + x_4) = x_1x_2 + x_1x_4 + x_2x_3 + x_3x_4 \neq g_L$, so f_L and g_L are not dual functions. Theorem 1 tells us that if f_L and g_L are not dual then there exists an non-robust input assignment. We can easily identify it: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$.*

Example (b): *The Boolean functions implemented by the lattice are $f_L = x_1x_3 + \bar{x}_1x_2$ and $g_L = x_1x_2 + \bar{x}_1x_3$. Since $f_L^D = (x_1 + x_3)(\bar{x}_1 + x_2) = x_1x_2 + \bar{x}_1x_3 = g_L$, so f_L and g_L are dual functions. Theorem 1 tells us that if f_L and g_L are dual then every assignment is robust. One can easily see that none of the input assignments cause diagonal 1's while the corresponding function evaluates to 0.*

We conclude that, in order to achieve robust computation, we must design lattices that have dual top-to-bottom and left-to-right Boolean functions.

3.2. Logic Optimization Problem

This gives rise to an interesting problem in logic optimization: given a target function f_T in SOP form, how should we assign the input literals such that $f_L = f_T$ and $g_L = f_T^D$? In other words, how should we assign literals so that the lattice implements the target function between the top and bottom plates, and implements the dual of the function between the left and right plates? As described in the previous section, having dual functions ensures robustness.

While maximizing the margins, we also need to consider the area of the lattice; this can be measured by the total number of switches $R \times C \times N \times M$ in the lattice. Here $R \times C$ and $N \times M$ represent the number of regions and the number of switches for each region, respectively.

We suggest a four-step algorithm for optimizing the lattice area while meeting prescribed worst-case margins for a given target function f_T.

Algorithm:

(1) Begin with the target function f_T and its dual f_T^D both in MSOP form.
(2) Find a lattice with the smallest number of regions that satisfies the conditions: $f_L = f_T$ and $g_L = f_T^D$. This determines $R \times C$.
(3) Dependent on the defect rates of the technology, determine the required worst-case one and zero margin values.
(4) Determine the number of switches required in each region in order to meet the prescribed margins. This determines $N \times M$.

The first step is straightforward. The dual of the target function can be computed from Definition 6. Exact methods such as Quine-McCluskey or heuristic methods such as Espresso can be used to obtain functions in MSOP form [McCluskey 1986; Brayton et al. 1984].

For the second step of the algorithm, we point the reader to our prior work. In [Altun and Riedel 2010; 2011], we addressed the problem of assigning literals to switches in a lattice in order to implement a given target Boolean function. The goal was to minimize the number of regions. We presented an efficient algorithm that produces lattices with a size that grows linearly with the number of products of the target Boolean function. Suppose that f_T and f_T^D in MSOP form have A and B product terms, respectively. Our algorithm produces lattices with $B \times A$ regions ($R = B$ and $C = A$) for which $f_L = f_T$ and $g_L = f_T^D$ [Altun and Riedel 2010].

For the third step, we assume that the defect rates of the switches are known or can be estimated. Recall that we consider two types of defects: those that result in switches being OFF while they are supposed to be ON (call these "ON-to-OFF" defects), and defects that result in switches being ON while they are supposed to be OFF (call these "OFF-to-ON" defects). We allow for different rates for both types of defects. Based upon the ON-to-OFF and OFF-to-ON defect rates, we establish the worst-case one and zero margins, respectively.

For the fourth step, we need to determine N and M such that the lattice meets the prescribed margins. Figure 18 shows the general relationship between margins and N and M. It suggests how we should select values of N and M. For instance, suppose that we require a 20% one margin and a 5% zero margin. Figure 18 tells us that we need to select a larger value of M than that of N. Also, from the figure, we observe that regardless of whether we increase N or M, the sum of the margins always increases. This is due to the percolation phenomenon: the larger the lattice, the steeper the non-linearity curve. Based upon these considerations, we use a simple greedy technique to set the required values of N and M. The method tries worst-case margins for different values of N and M until the prescribed margins are met.

We elucidate our algorithm with the following examples. For all of the examples, we use 10% worst-case one and zero margins.

N	M	One Margin	Zero Margin	Sum of Margins
$\uparrow$	$\bullet$	$\downarrow$	$\uparrow$	$\uparrow$
$\bullet$	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$

Fig. 18. Relationship between margins, and N and M.

Example 2 *Suppose that we are given the following target function f_T in MSOP form:*

$$f_T = x_1 x_2.$$

First, we compute its dual f_T^D in MSOP form:

$$f_T^D = x_1 + x_2.$$

The number of products in f_T and f_T^D are 1 and 2, respectively, i.e., $A = 1$ and $B = 2$.
Then, we construct a lattice such that $f_L = f_T = x_1x_2$ and $g_L = f_T^D = x_1 + x_2$. The lattice is illustrated in Figure 19. Note that $R = B = 2$ and $C = A = 1$.
Finally, we find that $N = 4$ and $M = 6$ in order to satisfy 10% worst-case one and zero margins.
As a result, the lattice area $= R \times C \times N \times M = 2 \times 1 \times 4 \times 6 = 48$.

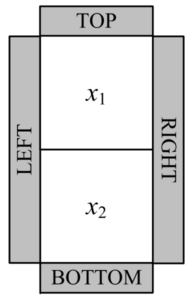


Fig. 19. A lattice that implements $f_L = x_1x_2$ and $g_L = x_1 + x_2$.

Example 3 *Suppose that we are given the following target function f_T in MSOP form:*

$$f_T = x_1\bar{x}_2 + \bar{x}_1x_2.$$

First, we compute its dual f_T^D in MSOP form:

$$f_T^D = x_1x_2 + \bar{x}_1\bar{x}_2.$$

We have that $A = 2$ and $B = 2$.
Then, we construct a lattice such that $f_L = f_T$ and $g_L = f_T^D$. The lattice is illustrated in Figure 20. Note that $R = B = 2$ and $C = A = 2$.
Finally, we find that $N = 4$ and $M = 6$ in order to satisfy 10% worst-case one and zero margins.
As a result, the lattice area $= R \times C \times N \times M = 2 \times 2 \times 4 \times 6 = 96$.

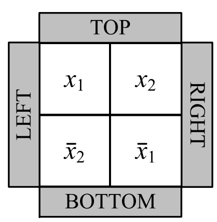


Fig. 20. A lattice that implements $f_L = x_1\bar{x}_2 + \bar{x}_1x_2$ and $g_L = x_1x_2 + \bar{x}_1\bar{x}_2$.

Example 4 *Suppose that we are given the following target function f_T in MSOP form:*

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

First, we compute its dual f_T^D in MSOP form:

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

We have that $A = 5$ and $B = 4$.
Then, we construct a lattice such that $f_L = f_T$ and $g_L = f_T^D$. The lattice is illustrated in Figure 21.
Note that $R = B = 4$ and $C = A = 5$.
Finally, we find that $N = 4$ and $M = 5$ in order to satisfy 10% worst-case one and zero margins.
As a result, the lattice area $= R \times C \times N \times M = 4 \times 5 \times 4 \times 5 = 400$.

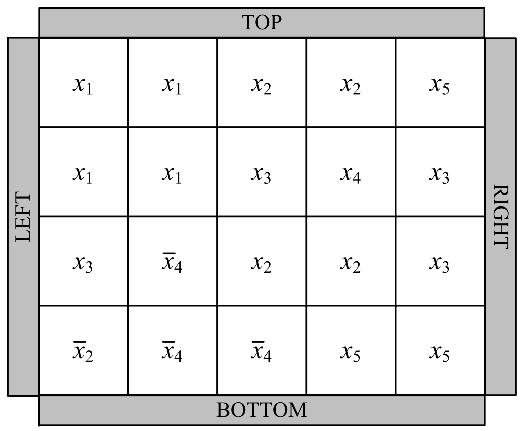


Fig. 21. A lattice that implements $f_L = x_1\bar{x}_2x_3 + x_1\bar{x}_4 + x_2x_2\bar{x}_4 + x_2x_4x_5 + x_3x_5$ and $g_L = x_1x_2x_5 + x_1x_3x_4 + x_2x_3\bar{x}_4 + \bar{x}_2\bar{x}_4x_5$.

We implement the target functions with specified margins. Note that because of the lattice duality, the one and zero margins of target functions become the zero and one margins of their duals, respectively.

4. EXPERIMENTAL RESULTS

We report synthesis results for some common benchmark circuits [McElvain 1993]. We consider each output of a benchmark circuit as a separate target Boolean function. Figure 22 lists the required lattice areas for the target functions meeting 10% worst-case one and zero margins. Recall that the lattice area is defined as the number of switches in the lattice. It can be calculated as $R \times C \times N \times M$ where $R \times C$ and $N \times M$ represent the number of regions and the number of switches for each region, respectively.

In order to obtain the lattice areas, we follow the steps of the proposed algorithm in Section 3.2. We first obtain values for A and B, the number of products in the target functions and their duals, respectively. Our algorithm sets $R = A$ and $C = B$, so produces lattices with $B \times A$ regions. We calculate values of N and M that satisfy the prescribed 10% worst-case margins.

Figure 22 reports the lattice areas, calculated as $A \times B \times N \times M$. Examining the numbers in the table, we see that number of switches needed per region, $N \times M$, is negatively correlated with the number of regions, $A \times B$. That is to say, Boolean functions with more products (larger $A \times B$ values) need smaller regions (smaller $N \times M$ values) to meet prescribed margins. This indicates a positive scaling trend: the lattice size grows more slowly than the function size. This key behavior is due to the percolation phenomena.

5. DISCUSSION

The two-terminal switch model is fundamental and ubiquitous in electrical engineering [Bryant 1987]. Either implicitly or explicitly, nearly all logic synthesis methods target circuits built from 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 model consists

Circuit	A	B	Number of regions	N	M	Lattice area
alu1	3	2	6	5	6	180
alu1	2	3	6	4	6	144
alu1	1	3	3	4	6	72
clpl	4	4	16	4	5	320
clpl	3	3	9	4	5	180
clpl	2	2	4	4	6	96
clpl	6	6	36	4	5	720
clpl	5	5	25	4	5	500
newtag	8	4	32	5	5	800
dc1	4	4	16	4	5	320
dc1	2	3	6	4	6	144
dc1	4	4	16	4	5	320
dc1	4	5	20	4	6	480
dc1	3	3	9	4	5	180
misex1	2	5	10	4	7	280
misex1	5	7	35	4	6	840
misex1	5	8	40	4	6	960
misex1	4	7	28	4	6	672
misex1	5	5	25	4	5	500
misex1	6	7	42	4	5	840
misex1	5	7	35	4	6	840
b12	4	6	24	4	6	576
b12	7	5	35	5	5	875
b12	7	6	42	5	5	1050
b12	4	2	8	5	6	240
b12	4	2	8	5	6	240
b12	5	1	5	6	5	150
b12	9	6	54	5	5	1350
b12	6	4	24	5	5	600
b12	7	2	14	6	5	420
newbyte	1	5	5	4	7	140
newapla2	1	6	6	4	7	168
c17	3	3	9	4	5	180
c17	4	2	8	5	6	240
rd53	5	10	50	4	6	1200
rd53	10	10	100	4	5	2000
rd53	16	16	256	3	5	3840

Fig. 22. Lattice areas for the output functions of benchmark circuits in order to meet 10% worst-case one and zero margins.

of a regular lattice of multi-terminal switches, each controlled by a Boolean literal. This model is conceptually general and applicable to a range of emerging technologies, including nanowire crossbar arrays [Cui and Lieber 2001] and magnetic switch-based structures [Khitun et al. 2008]. We are investigating its applicability to DNA nanofabrics [Pistol et al. 2006; Rothemund 2006]. In this paper, we focused on four-terminal switches. In future work, we will the extend the results paper to lattices of eight-terminal switches, and then to 2^k-terminal switches, for arbitrary k.

Particularly with self-assembly, nanoscale lattices are often characterized by high defect rates. Significantly, unlike many other strategies for defect tolerance, our method does not require defect identification followed by reconfiguration. Our method provides *a priori* tolerance to defects of any kind, both permanent and transient, provided that such defects occur probabilistically and independently. Indeed, percolation depends on a random distribution of defects. If the defect probabilities are correlated across regions, then the steepness of the percolation curve decreases; as a result, the defect tolerance diminishes. In future work, we will study this tradeoff mathematically and develop synthesis strategies to cope with correlated probabilities in defects.

REFERENCES

ALTUN, M. AND RIEDEL, M. D. 2010. Lattice-based computation of Boolean functions. In *Design Automation Conference*. 609–612.

ALTUN, M. AND RIEDEL, M. D. 2011. Logic synthesis for switching lattices. *Will appear in IEEE Transactions on Computers*.

ALTUN, M., RIEDEL, M. D., AND NEUHAUSER, C. 2009. Nanoscale digital computation through percolation. In *Design Automation Conference*. 615–616.

BRAYTON, R. K., MCMULLEN, C., HACHTEL, G. D., AND SANGIOVANNI-VINCENTELLI, A. 1984. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers.

BROADBENT, S. R. AND HAMMERSLEY, J. M. 1957. Percolation processes I. crystals and mazes. In *Proceedings of the Cambridge Philosophical Society*. 629–641.

BRYANT, R. E. 1987. Boolean analysis of MOS circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 6,* 4, 634–649.

CUI, Y. AND LIEBER, C. M. 2001. Functional nanoscale electronic devices assembled using silicon nanowire building blocks. *Science 291,* 5505, 851–853.

DEHON, A. 2005. Nanowire-based programmable architectures. *ACM Journal on Emerging Technologies in Computing Systems 1,* 2, 109–162.

ESHAGHIAN-WILNER, M. M., KHITUN, A., NAVAB, S., AND WANG, K. 2006. A nano-scale reconfigurable mesh with spin waves. In *International Conference on Computing Frontiers*. 5–9.

HOGG, T. AND SNIDER, G. 2007. Defect-tolerant logic with nanoscale crossbar circuits. *Journal of Electronic Testing 23*, 117–129.

HUANG, J., TAHOORI, M., AND LOMBARDI, F. 2004. On the defect tolerance of nano-scale two-dimensional crossbars. In *International Symposium on Defect and Fault Tolerance of VLSI Systems*. 96–104.

HUANG, Y., DUAN, X., CUI, Y., LAUHON, L. J., KIM, K., AND LIEBER, C. M. 2001. Logic gates and computation from assembled nanowire building blocks. *Science 294,* 5545, 1313–1317.

KHITUN, A., BAO, M., AND WANG, K. L. 2008. Spin wave magnetic nanofabric: A new approach to spin-based logic circuitry. *IEEE Transactions on Magnetics 44,* 9, 2141–2152.

KUEKES, P., ROBINETT, W., SEROUSSI, G., AND WILLIAMS, R. 2005. Defect-tolerant interconnect to nanoelectronic circuits: Internally redundant demultiplexers based on error-correcting codes. *Nanotechnology 16,* 6, 869–882.

LUO, Y., COLLIER, C. P., JEPPESEN, J. O., NIELSEN, K. A., DELONNA, E., HO, G., PERKINS, J., TSENG, H., YAMAMOTO, T., STODDART, J. F., AND HEATH, J. R. 2002. Two-dimensional molecular electronics circuits. *ChemPhysChem 3,* 6, 519–525.

MCCLUSKEY, E. J. 1986. *Logic Design Principles with Emphasis on Testable Semicustom Circuits*. Prentice-Hall.

MCELVAIN, K. 1993. IWLS93 benchmark set: Version 4.0, distributed as part of the IWLS93 benchmark distribution, http://www.cbl.ncsu.edu:16080/benchmarks/lgsynth93/.

PISTOL, C., LEBECK, A. R., AND DWYER, C. 2006. Design automation for dna self-assembled nanostructures. In *Design Automation Conference*.

ROTHEMUND, P. W. K. 2006. Folding dna to create nanoscale shapes and patterns. *Nature 440,* 7082, 297–302.

SNIDER, G. AND WILLIAMS, R. 2007. Nano/CMOS architectures using a field-programmable nanowire interconnect. *Nanotechnology* 18, 1–11.

SUN, F. AND ZHANG, T. 2006. Two fault tolerance design approaches for hybrid CMOS/nanodevice digital memories. In *International Workshop on Defect and Fault Tolerant Nanoscale Architectures*.

WHITESIDES, G. M. AND GRZYBOWSKI, B. 2002. Self-assembly at all scales. *Science 295,* 5564, 2418–2421.

YAN, H., PARK, S. H., FINKELSTEIN, G., REIF, J. H., AND LABEAN, T. H. 2003. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science 301,* 5641, 1882–1884.

ZIEGLER, M. M. AND STAN, M. R. 2003. CMOS/nano co-design for crossbar-based molecular electronic systems. *IEEE Transactions on Nanotechnology 2,* 4, 217–230.

ZOMAYA, Y. 2006. Molecular and nanoscale computing and technology. In *Handbook of Nature-Inspired and Innovative Computing*. Springer, Chapter 14, 478–520.