

Defect-tolerant Logic with Nanoscale Crossbar Circuits

Tad Hogg and Greg Snider
HP Labs
Palo Alto, CA

May 25, 2004

Abstract

Crossbar architectures are one approach to molecular electronic circuits for memory and logic applications. However, currently feasible manufacturing technologies introduce numerous defects so insisting on defect-free crossbars would give unacceptably low yields. Instead, increasing the area of the crossbar provides enough redundancy to implement circuits in spite of the defects. We identify reliability thresholds in the ability of defective crossbars to implement boolean logic. These thresholds vary among different implementations of the same logical formula, allowing molecular circuit designers to trade-off reliability, circuit area and the computational complexity of locating functional components. We illustrate these choices for an AND gate and, of more practical interest, binary adders.

1 Introduction

Molecular electronics offers the possibility of significantly denser circuits than is possible with current lithography-based manufacturing. Achieving this potential requires circuit designs that can exploit the capabilities of molecular electronics and compensate for limitations of current approaches to fabricating such circuits, particularly defects. In this paper, we consider defect-tolerant systems architecture in the context of a particular type of molecular circuit, the crossbar described in Section 2. With currently feasible technologies, nanoscale crossbars will contain numerous defective junctions. Thus as a practical matter for implementing logic operations, we need to create functioning circuits in spite of defects rather than simply discarding any circuit with even a single defect (which would give unacceptably low yield).

For nanoscale crossbar devices, the main type of defect is that introduced during manufacture (so-called “static defects”) rather than during operation. This is reasonable for plausible technologies, which involve high temperatures during manufacture, and hence a relative ease of introducing defects, but low

temperature during operation, with much less chance of creating new defects. In this situation, an appropriate systems architecture consists of a compiler to arrange for desired circuit behaviors by only using correctly functioning components of a given crossbar circuit, as determined from a testing phase after manufacture [10]. This approach of avoiding known defects gives a defect-tolerant system architecture. It contrasts with methods dealing with faults that may appear during the operation of the device, perhaps intermittently, e.g., using majority votes from replicated hardware [19].

We restrict attention to defects leading to inoperative connections (i.e., with no ability to activate them to make diode connections), rather than defects that short out a wire, or prevent routing the output of one gate to the input of another. In this scenario, we can test the circuits to determine which crosspoints are defective, and then use the remaining ones to implement the circuit. That is, a compiler uses the required logic formula and a table of defects to find a way to implement the formula.

This leads to the central question addressed in this paper: given a defect rate and a certain size crossbar, how likely is it we can find a way to implement a particular logical formula in the crossbar? Determining whether such a circuit exists, and if so, finding one, is a combinatorial search problem. Thus a related question is the computational difficulty for the compiler to identify an implementation, or conclude no implementation is possible. For a given desired circuit and crossbar size, decreasing the defect rate will generally require more difficult and costly manufacturing. On the other hand, increasing the allowable defect rate will make it less likely the desired circuit can be implemented and can also result in longer runtimes for the compiler to identify a way to implement the circuit while avoiding the defects.

As a further aspect of this problem, a logical formula can be written in various logically equivalent forms, e.g.,

$$(a \text{ OR } b) \text{ AND } c$$

$$(a \text{ AND } c) \text{ OR } (b \text{ AND } c)$$

are logically equivalent. These *rewrites* can involve different numbers of terms, and hence require different crossbar areas to implement. They can also differ in their likelihood of being implementable on crossbars with defects.

After describing the molecular crossbar hardware, we illustrate how a simple logic gate can be implemented using a crossbar in Section 3 and show how defects affect different implementation choices. We then turn to a more interesting circuit: a binary adder. We first describe two approaches to implementing adders using crossbars, and then show their feasibilities in the face of defects. This paper thus shows how crossbar architectures can be applied to create logic circuits, even with numerous manufacturing defects.

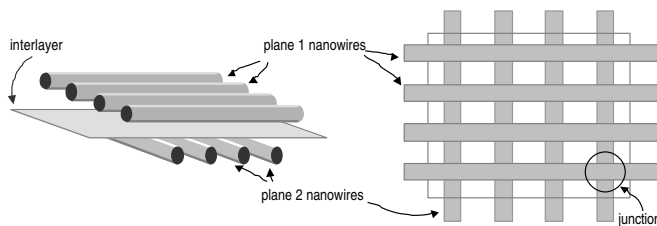


Figure 1: Schematic view of a molecular crossbar from two different perspectives. Each junction may be independently configured to behave as an electronic device.

2 Crossbar Architecture

The crossbar architecture is a general approach for molecular circuits [1, 2, 3, 12, 16, 5, 18, 9, 14]. Specifically, a molecular crossbar consists of two parallel planes of molecular wire arrays separated by a thin layer of a chemical species (called the “interlayer”) with particular electrochemical properties (Fig. 1). Each plane consists of a number of parallel molecular wires (also called “nanowires”), with each wire in a plane being of the same type. The wires in one plane cross the wires in the other plane at a right angle. The region where two perpendicular wires cross is called a junction or crosspoint. Depending on the nature of the interlayer and nanowires, each junction may be configured to implement an electronic device, such as a resistor, diode or field effect transistor [15], or may be left unconfigured so the two crossing wires forming the junction do not interact electrically. We consider crossbars whose junctions can only be configured as either resistors or diodes, since those are easier to fabricate with current technology than configurable transistors.

The crossbar structure is an attractive architecture for molecular electronics since it is relatively simple and inexpensive to fabricate using either chemical self-assembly or nanoimprint lithography. In practice, the crossbar molecular device can be connected to larger-scale external circuits (built using conventional lithography) to provide I/O to the molecular devices [13, 6].

By suitable selection of the type of connections at each crosspoint (e.g., no connection, or a diode in one direction or the other), crossbars can be set to evaluate any logical formula expressed as a combination of AND and OR operations. Fig. 2 shows one example. To see this, consider the output wire, labeled “X”. It is connected to ground through a resistor, and via diode junctions to the second and third vertical wires. If both vertical wires are at low voltage (“off”), then the output wire X will also be at low voltage due to its connection to ground. On the other hand, if either of the connected vertical wires is at high voltage (“on”), the diode connection from the high voltage vertical wire(s) will give a high voltage to the output wire (since, by design, the diode resistance in the forward direction is much smaller than the resistor connecting the output

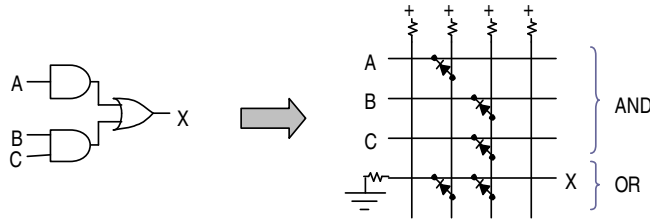


Figure 2: Implementing the AND/OR function $X = A + BC$ with a diode crossbar and resistors.

wire to ground). If only one of the vertical wires is on, the high resistance of the diode junction in the reverse direction ensures that the output wire remains at high voltage. Thus this combination of resistors and diode connections makes the output X equal to the logical-OR of the inputs on the two vertical wires. Similarly, the connections from the inputs A , B and C implement logical-ANDs.

The crossbar of Fig. 2 connects each column, through a *pullup* resistor, to a positive voltage source. With the diode directions shown here, each column implements the logical-AND of its inputs (the horizontal wires). Each output row, connected to ground through a *pulldown* resistor, implements the logical-OR of the columns connected to it through diode junctions. Although this is not the only way to configure crossbar circuits, it provides a simple functional form in which each output is the logical-OR of a number of terms, each of which is the logical-AND of some inputs.

An important limitation of diode/resistor logic is its inability to implement logical inversion (i.e., a NOT gate). However, by presenting the circuit with two wires for each input (i.e., one wire representing the true input value, the other representing its complement), the crossbars can produce internal signals in both the original and complemented forms. Combining these signals using just AND and OR operations then allows evaluating any logical formula. The complemented inputs to the crossbar are readily produced by the external circuit, fabricated using conventional technology, to which the crossbar is connected for input and output. Thus by doubling the number of wires and presenting all primary inputs in both true and complemented forms, the diode crossbar architecture can implement any logical formula just using combinations of AND and OR operations, as illustrated in Fig. 3.

3 An AND Gate

A simple example of creating a logic circuit from the crossbar architecture is the logical AND of k inputs. One implementation is as a single k -input AND gate, i.e., using k connections to a single output wire. For k equal to a power of 2 this can also be implemented by a set of $k - 1$ 2-input AND gates, as illustrated for $k = 4$ in Fig. 4.

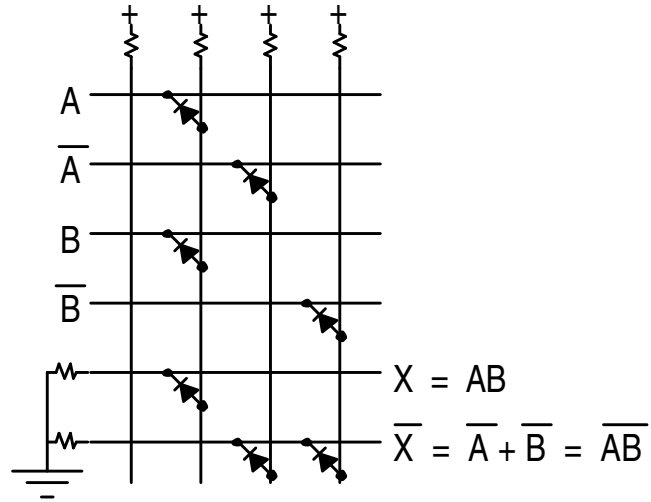


Figure 3: DeMorgan's theorem allows generating a given logical AND/OR function along with its complement. This generally requires all input signals to be presented in both original and complemented forms.

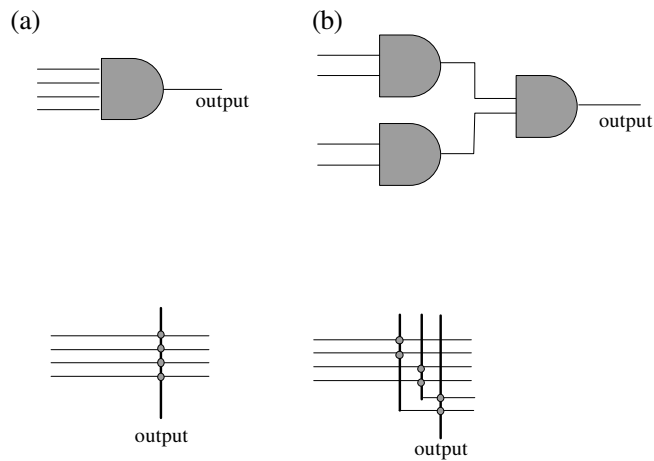


Figure 4: Logic gates: a 4-input AND, and the same function using 2-input AND gates. Also shown is an implementation of these circuits using parts of a crossbar network.

For this example we suppose the assignments of input signals to input wires are fixed, e.g., either from external connections or from outputs of another part of a larger overall circuit. More generally, the circuit design could also involve searching for a suitable choice of these input wires among a larger number of rows in the crossbar network, as we discuss for the adder circuit in Section 4.

If each connection in the crossbar is defective with independent probability p , the probability a given column wire in the crossbar can be used to form a gate with k given inputs (i.e., row wires) is $(1 - p)^k$. Thus in a crossbar with N columns, the probability that at least one column will be able to implement the k -input gate is

$$P_{\text{gate}}(k, N) = 1 - (1 - (1 - p)^k)^N \quad (1)$$

Computing the logical AND of k inputs using a single column, the probability to find a functional circuit is $P_{\text{circuit}} = P_{\text{gate}}$. If we use $k - 1$ 2-input gates to compute the same logical value, the probability to find a functioning circuit is *approximately*¹

$$P_{\text{circuit}}(k, N) = P_{\text{gate}}(2, N)^{k-1} \quad (2)$$

Fig. 5 shows the behavior of these expressions. In this example, with low p values, both circuits for evaluating the logical expression are very likely to be constructible. However, as p increases, the chance of being able to find a single gate circuit drops more rapidly. Thus, it is easier to find seven column wires with functioning connections for 2-input gates than to find one column to implement a single 8-input gate. This difference becomes more extreme as the size of the circuits increases. This figure also illustrates the threshold nature of the behavior: most of the drop in success probability occurs over a short range of p values. This drop is even more abrupt for larger circuits.

Another way to characterize the ability to find functional circuits in spite of defects is by the number of additional columns, or, equivalently, increased circuit area, necessary to give at least, say, a 95% probability of being able to find a functioning circuit. This amounts to inverting Eq. (1) and (2). Specifically, for a success probability of α , we require

$$N = \frac{\ln(1 - \alpha)}{\ln(1 - (1 - p)^k)} \quad (3)$$

columns to have probability α of being able to construct a single k -input gate². Such a circuit has k rows for the inputs, for a total area of kN . Similarly, Eq. (2)

¹The approximation arises from allowing two or more gates to be constructed from the same column, which is not possible for the crossbar circuit. Hence this approximation overstates the actual probability of finding a correct circuit. Alternatively, it is the exact result for a different crossbar architecture: i.e., in which the column wires are in distinct, unconnected segments, with each segment covering just k row wires. However, unless the number of columns is small, when there are columns with so few defects as to allow constructing multiple gates, there are even more likely to be some other columns, with more defects but still few enough to allow implementing a single gate. In these situations, the error from this approximation is small.

²In actual circuits, the number of columns must be an integer. So the actual minimum number of columns is the smallest integer greater than or equal to this expression.

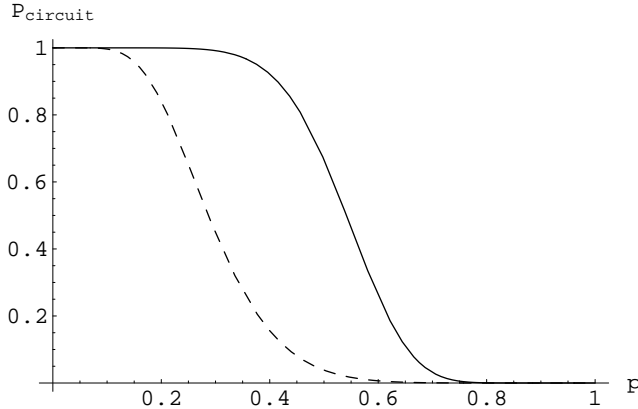


Figure 5: Probability to be able to find a correct circuit in a crossbar with 10 columns as a function of defect probability p . The dashed curve is for a single 8-input AND gate, and the solid curve is for the equivalent logical expression made with seven 2-input gates.

gives

$$N = \frac{\ln(1 - \alpha^{1/(k-1)})}{\ln(1 - (1-p)^2)} \quad (4)$$

required columns. Each gate has 2 inputs (distinct from all the rest) for a total circuit area of $2(k-1)N$ (not counting the additional area involved in the routing connections to take the outputs of some gates to the inputs of others).

Fig. 6 illustrates the behavior of the area requirements for the two implementations of the logical AND of k inputs. When p is low, both methods have a high chance of success (as also seen in the threshold behavior illustrated in Fig. 5). In this case, the smaller size of the k -input AND gate is more important than the slightly higher success probability with the combination of 2-input gates. As p increases, the success probability for the k -input gate drops more rapidly, leading to faster growth in area required, than the 2-input gates.

To see this tradeoff more generally, note the ratio of area required for $k-1$ 2-input gates to that for a single k -input gate is proportional to

$$(k-1) \ln(1 - \alpha^{1/(k-1)}) \ln(1 - (1-p)^k) \quad (5)$$

which grows as $(k-1) \ln(k-1)(1-p)^k$ as k gets large (for fixed α and p strictly between 0 and 1). This goes to 0 as k increases, exponentially fast. Thus for large circuits, it is better to use more gates with few inputs than fewer gates with more inputs.

This discussion considers only two possible implementations of the logic formula. Additional possibilities include using a mixture of gates with different numbers of inputs, or combining 4-input, rather than 2-input, gates (so a k -input

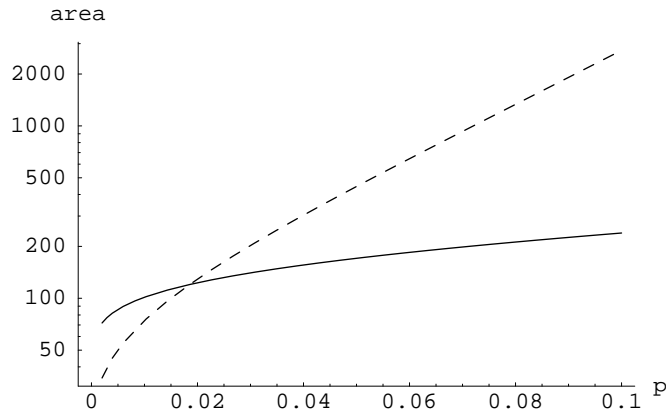


Figure 6: Area required to have at least 95% probability to be able to find a correct circuit as a function of the defect probability p . The dashed curve is for a single 32-input AND gate, and the solid curve is for the equivalent logical expression made with thirty-one 2-input gates. Actual circuits must have an integer number of columns, resulting in slightly larger areas and step-functions in these plots, but with qualitatively the same behavior.

AND operation would be built from $(k - 1)/3$ 4-input AND gates). This give qualitatively similar behaviors to those shown in Fig. 5 and additional choices for circuit implementations.

4 Adder Circuits

We now consider the mapping of small adder circuits onto crossbars. There are several different ways to implement such circuits. Fig. 7 shows a straightforward 3-bit, ripple-carry adder that is essentially a direct translation of the logic circuit shown at the top, producing four output bits $S_0 \dots S_3$ representing the sum of two 3-bit numbers. For instance the bottom wire of the crossbar and the leftmost two columns compute the least significant bit of the sum, S_0 , as $S_0 = A_0 \overline{B_0} + \overline{A_0} B_0$. This logical formula is equivalent to the exclusive-OR of the two least-significant bits of the numbers to be added, i.e., A_0 and B_0 .

Because this implementation uses several levels of logic, some of the intermediate output signals must be fed back to some of the inputs, possibly requiring signal regeneration in the process to compensate for degradation due to diode voltage drops. The circuit uses 12 input wires: each of the two numbers to be added has 3 bits, and each bit must be presented as original and complementary values. The circuit has 4 outputs. The addition of the feedback signals gives a total of 30 rows. Forming the required logical operations on these values uses 25 columns, as shown in Fig. 7. This implementation, which uses 78 junctions

configured as diodes, thus requires a minimum crossbar area of $30 \times 25 = 750$ junctions.

A second approach to the 3-bit adder is shown in Fig. 8. Here the entire circuit uses only two logic levels. It requires only enough rows to handle the input and output wires, i.e., 16 rows. The circuit requires 31 column wires to perform logic operations on the inputs, for a minimum crossbar area of $31 \times 16 = 496$. Again S_0 is computed as $S_0 = A_0\overline{B_0} + \overline{A_0}B_0$, using the bottom wire of the crossbar and the 2nd and 3rd columns from the right. This implementation eliminates the need for feedback and requires less area. On the other hand, it requires more diode junctions (147) and uses a greater number of diodes along some of the vertical and horizontal wires. For instance, the circuit in Fig. 7 never uses more than four diodes on any wire, while the circuit in Fig. 8 requires as many as 16. This corresponds to the observation in Section 3 for the AND gate: finding wires with many functioning junctions is more difficult than finding several wires each requiring fewer junctions. Thus we can expect this circuit, packing more diodes in a smaller area, will be more difficult to implement on a crossbar with defects than that of Fig. 7.

Both of these adder circuits produce output bits corresponding to the sum. This is suitable when these circuits are considered as stand-alone components whose outputs are delivered to an external circuit composed of conventional technology (which can implement inversion). If instead the crossbar adder is to be used as part of a larger molecular-scale circuit, the adder will need to be extended to also produce complement values of each of the output bits so subsequent crossbars, using the results of the adder, will have access to both original and complement values for their inputs. Such extensions are readily included, as described with the discussion of Fig. 3, and will result in doubling the number of output wires. For simplicity, we focus on the adders treated as stand-alone circuits without the need for complement values for the outputs.

In the remainder of this section, we first describe the algorithm used to find a mapping of an adder circuit implementation to a crossbar with a known set of defects. We then use this algorithm to produce implementations on simulations of defective crossbars to identify their ability to give functional adder circuits. Our focus is on crossbars whose configurable junction devices (e.g., diodes) have fixed parameters if they are functional (e.g., resistances in forward and reverse directions). A complementary analysis to that presented here could examine the consequences for circuit performance of various values of these parameters, as has been applied to another adder implementation, using a look-up table to evaluate all the combinations of inputs [20].

4.1 Allocation Algorithm

The diode/resistor fabric which we map onto is modeled as a set of four crossbars that have been tiled together to form a mosaic, illustrated schematically in Fig. 9 and more explicitly with the example circuits of Fig. 2 and 3. The pullup and pulldown “crossbars” are only one wire tall and wide respectively, but the resistors there can be defective just like the diodes in the diode crossbars, so it

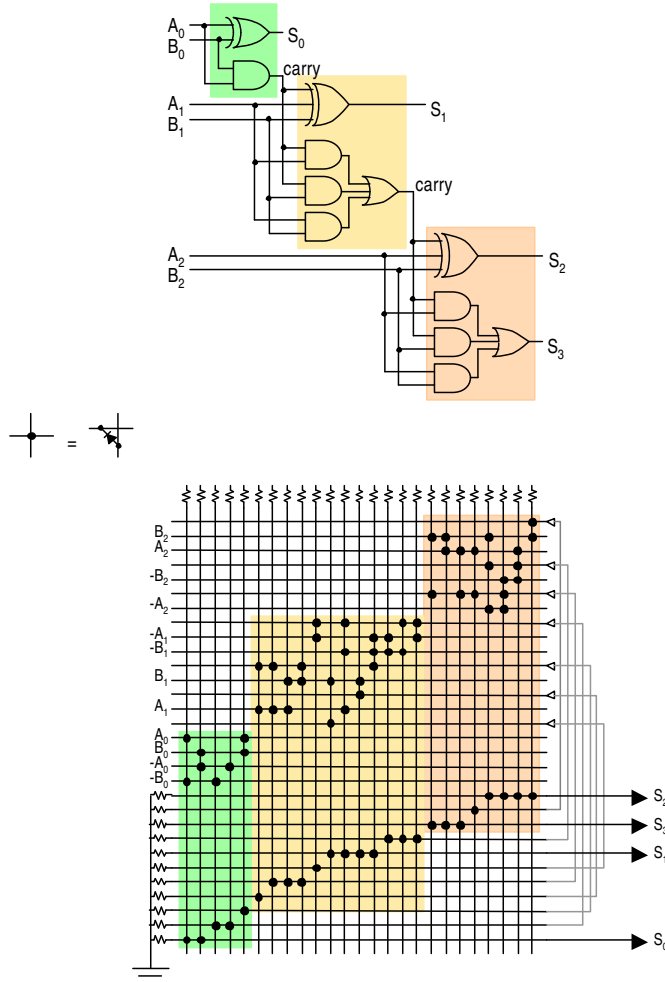


Figure 7: A 3-bit adder which adds two 3-bit numbers (denoted as the bits $A_2A_1A_0$ and $B_2B_1B_0$, respectively) to produce a 4-bit sum (with bits $S_3S_2S_1S_0$). The ripple-carry logic implementation (top) translates directly to a diode crossbar implementation (bottom) using feedback from some of the outputs to the inputs (gray lines). Regenerative buffers (left pointing triangles) between stages regenerate signals degraded by diode and resistor voltage drops. The input wire marked $-A_0$ gives the complement of input bit A_0 , and similarly for the other inputs. Note that the carry bit between successive stages of the crossbar implementation must be presented in both original and complemented forms.

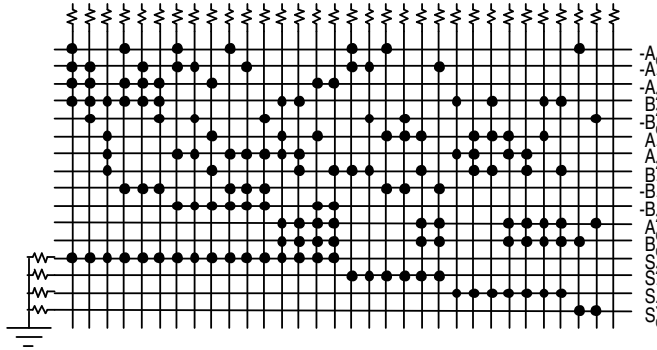


Figure 8: A 3-bit adder implemented as 2-level logic in a single diode crossbar. Although this approach uses more diodes, it consumes less area, avoids the feedback and regenerative buffers between stages, and will likely offer less propagation delay. Inputs and outputs are labeled as in Fig. 7. The rightmost column wire is not used in this circuit.

simplifies allocation to use a single model for all of the components. However, the consequence of a defective pullup or pulldown resistor is to disable the entire column or row, respectively, to which it is connected. Fortunately, these resistors, at the edge of the crossbar, are formed from junctions between the nanowires and much larger, microscale, wires. Thus the junction area per device is significantly larger than that for the diode junctions used in the AND and OR crossbars. This increased junction area means the chance of a defective resistor is far smaller than having a defective diode.

Even though the AND and OR crossbars share the same junction type and could be represented with a single crossbar, it is helpful to keep them separate since input signals may only be bound to horizontal wires entering the AND crossbar and output signals only bound to horizontal wires leaving the OR crossbar. The allocation problem, then, is to implement a circuit in the crossbars given a set of defective junctions in each. Unlike the discussion of the AND gate in Section 3, the allocation for the adder circuits also considers alternate choices for the input and output wires. However, input connections can only be made among wires preselected to be part of the AND crossbar, and output connections only among wires in the OR crossbar.

The allocation problem cannot be divided into separate crossbar allocation subproblems when the crossbars contain defects because a particular allocation of resources in one tile may actually preclude a successful allocation in another. We must also respect other constraints, such as input/output signal restrictions (in the case where the crossbars are embedded in a larger system) and asymmetric junctions, where component direction or polarity (such as for diodes) must be respected. We address the problem by searching globally for a solution that meets all constraints (defect avoidance, input/output constraints, junction

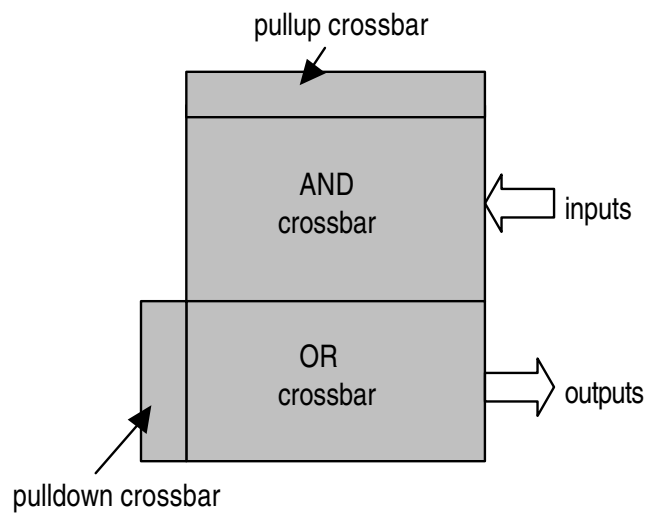


Figure 9: Model of diode array as a set of 4 connected crossbars. The AND and OR crossbars have configurable diode junctions, while the pullup and pulldown crossbars have configurable resistor junctions. Any junction in any crossbar may be defective, though the defect rate for junctions in the pullup and pulldown crossbars is much lower than for other junctions.

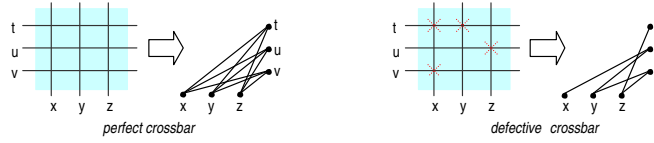


Figure 10: Representing a crossbar with a graph. Wires and junctions in the crossbar correspond to nodes and edges of the graph, respectively. Defective junctions are shown marked with an “X”.

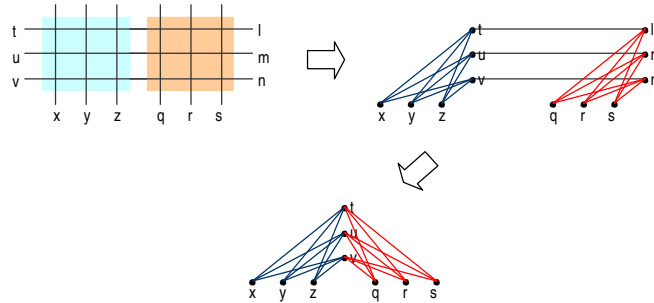


Figure 11: Representing composite crossbars with a graph. Edges are colored to represent the functionality of the junction that each one represents, since each crossbar might have different functionality.

polarity, and crossbar interaction) simultaneously.

Our allocation algorithm uses graphs with annotated edges and nodes to represent both the original circuit to be mapped onto a set of crossbars as well as the crossbars themselves. Fig. 10 shows how a crossbar is represented with a graph: a wire in the crossbar is represented by a node in the graph, and a junction is represented by an edge between the two nodes representing the wires that define the junction. A perfect crossbar (left) has an edge for every junction. A defective crossbar (right) has edges only for usable junctions.

Graphs for multiple crossbars are constructed by first creating a graph for each individual crossbar (Fig. 11, top left) and then interconnecting them (top right). The resulting graph may then be (optionally) optimized by merging identical nodes (bottom).

Allocation is accomplished by (1) creating graphs representing the desired circuit and compound crossbars; and (2) searching for an embedding or monomorphism between the circuit graph and the compound crossbar graph. Fig. 12 illustrates this in detail. The steps for allocation are:

1. For the desired circuit (Fig. 12a) create a circuit graph (Fig. 12b) representing it: wires and junctions in the circuit are represented by nodes and edges in the circuit graph, respectively.

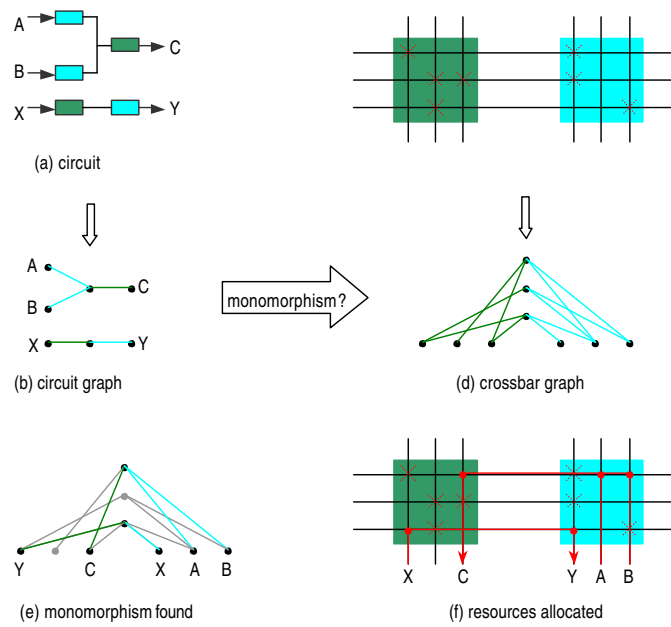


Figure 12: Resource allocation: searching for a monomorphism between a circuit graph and a crossbar graph. The corresponding algorithm steps are described in the text.

2. For the desired target compound crossbar (Fig. 12c), create a compound tile graph (Fig. 12d) representing it. As in circuit graph case, wires and junctions in the crossbars are represented by nodes and edges in the compound tile graph, respectively. A defective junction in a crossbar is represented by the absence of its corresponding edge in the crossbar graph.
3. Annotate the edges of the circuit graph and the crossbar graph with annotations representing the functionality of those edges (junctions in the circuit represented by the graph). For example, edges in both graphs representing resistors would all be tagged with identical annotations.
4. Annotate the nodes of the circuit graph and crossbar graph with annotations to constrain matching between the two graphs. As will be shown later, this is done to either (a) enforce input/output constraints between the desired circuit and other circuitry that has been or will be mapped to other areas of a large compound tile graph; or (b) enforce directionality constraints on asymmetric junctions, such as diodes, that must have, for example, an input delivered on a horizontal wire and an output driven on a vertical wire; or (c) enforce both.
5. Search for a monomorphism (Fig. 12e) between the annotated circuit graph and the annotated target crossbar graph to do allocation (Fig. 12f), subject to the constraints that node and edge annotations must match. In other words, a node in the circuit graph can only be matched with a node in the crossbar graph if they both have identical annotations or both have no annotations. Similarly, edges can only be matched if they both have identical (or non-existent) annotations. Efficient algorithms for searching for a graph monomorphism are well known [17, 4, 7].
6. Use the monomorphism to complete the allocation or mapping of wires and junctions in the desired circuit graph onto wires and junctions of the crossbar. For example, a node, A , in a circuit graph matched to a node, B , in the crossbar graph will be used to allocate the crossbar wire represented by B in the crossbar graph to carry the signal represented by A in the desired circuit. Similarly, an edge, X , in a circuit graph matched to an edge, Y , in the crossbar graph will be used to allocate the junction in the crossbar represented by Y in the crossbar graph for the electrical component represented by X in the desired circuit.

Edges in the circuit graph and crossbar graph are “colored” to reflect the component and junction functionality respectively. These edge colors are additional constraints when searching for a monomorphism or embedding, since an edge in the circuit graph may only be matched with an edge in the compound tile graph with the same color. The edge coloring and the implied matching constraint are referred to as “edge annotation.” Similarly, nodes may be annotated with the same matching constraint, namely that a node in a circuit graph may only be matched to a node in the crossbar graph with the same annotation. As shown in Fig. 13, this is useful when trying to meet input/output constraints

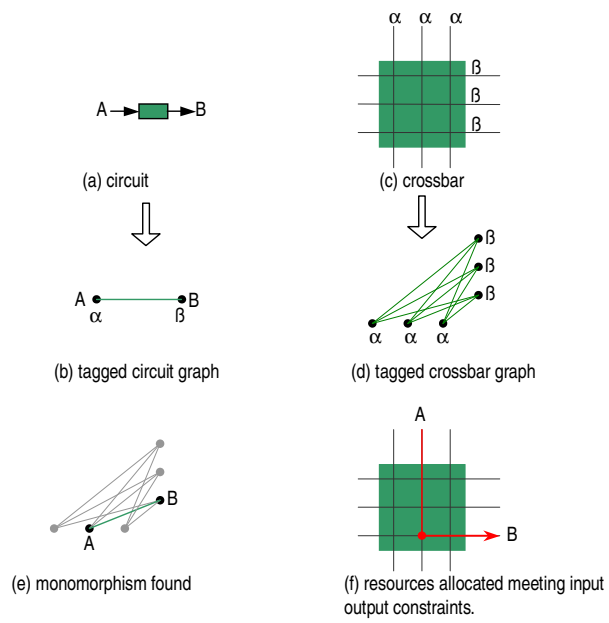


Figure 13: Nodes (wires) and edges (junctions) may be annotated with a matching constraint.

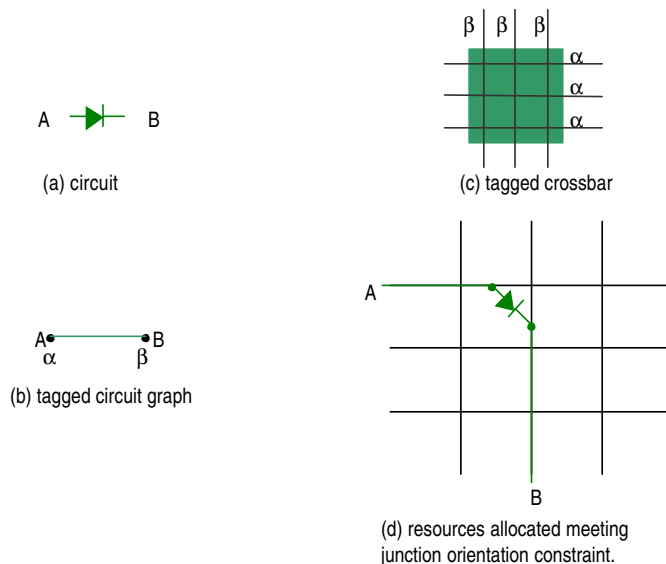


Figure 14: Node annotation to support allocation of asymmetric junctions.

for a circuit. In this example the simple circuit (Fig. 13a) is to be mapped onto a set of crossbars (Fig. 13c). External constraints may require that the A signal be mapped onto a vertical wire and the B signal be mapped onto a horizontal wire. This constraint can be met by tagging the computation graph vertices (Fig. 13b) and the crossbar graph vertices (Fig. 13d) with legal matching tags. In this case, the input signal, A , is tagged with the α tag as are the three vertical wires in the crossbar graph. Similarly, B is tagged with a β tag as are the three horizontal wires in the crossbar graph. When a monomorphism is found (Fig. 13e), it meets the input/output constraints for the allocation (Fig. 13f).

Node annotation is also useful for asymmetric junctions. The diode in Fig. 14 may only be configured correctly in the target compound tile in one direction. Annotating the vertices of the circuit graph and the compound tile graph appropriately assures that the diode will be allocated with the proper orientation.

This allocation algorithm is a *complete* search method: if it does not find a possible circuit implementation in the crossbar with given defects, then no such implementation exists. In practice, the computational cost of such search methods can be prohibitive, especially for circuits with many components. In that case, one could instead use an *incomplete* search method, which often solves combinatorial problems more rapidly than complete methods, but offers no guarantee that failure to find a solution means no solution exists. For the simulation results discussed below, we employed an incomplete search method obtained by simply imposing a bound on the number of graph matching attempts allowed for the allocation algorithm. If a match is not found within this number of at-

tempts, we consider the defective crossbar to be a failure. In a mass-production manufacturing context, the choice of the bound on the algorithm results in a trade-off between increasing the computing and testing time spent determining whether a crossbar is functional and decreasing the yield of functional circuits.

4.2 Adder Circuit Implementation Behaviors

To examine the behavior of implementing adder circuits on defective crossbars, we created a number of simulated test cases. Specifically, for a given adder implementation (e.g., single or multiple stage) and crossbar size, we mark each junction of the AND and OR crossbar as defective independently with probability p . Because the pullup and pulldown resistors have much lower defect rates, for simplicity, we restrict our attention to cases with no defective resistors. We then run the allocation algorithm, recording whether it found an allocation (within at most 30 seconds of CPU time, corresponding to about 7×10^7 graph matchings) and the number of steps required to reach a decision. We repeat this procedure on new crossbars with randomly selected defects (using the same parameters).

The limit on search time for the allocation algorithm was significantly larger than the typical number of matchings needed in the cases that produced a successful match. From a pragmatic standpoint, the need to rapidly test circuits after fabrication would preclude spending an inordinate amount of time trying to distinguish a crossbar with a possible, but difficult to find, circuit implementation from one with no possible implementations. Thus, imposing a bound on CPU time is a simple approach to avoiding this excessive search cost, with the tradeoff of slightly reducing the yield.

4.2.1 Threshold Behavior

From the set of simulation trials, we estimate P_{circuit} , the probability an implementation exists for this set of parameters. Specifically, suppose out of n trials we successfully find an allocation s times. Because each trial uses an independently generated crossbar, the probability to observe s successful circuits out of n trials is given by the binomial distribution $\text{Bi}(n, s; P_{\text{circuit}})$ where

$$\text{Bi}(n, k; p) \equiv \binom{n}{k} p^k (1 - p)^{n-k} \quad (6)$$

and P_{circuit} is the (unknown) actual value of the probability a circuit exists. Thus the likelihood that P_{circuit} equals the value f is proportional to $f^s (1 - f)^{n-s}$. Maximizing this gives $f = s/n$ as the maximum-likelihood estimate of p . Evaluating the range of f values accounting for, say, 95% of the likelihood gives an indication of how well our simulation determines the value.

Fig. 15 is an example of the behavior of implementing a 3-bit adder circuit, using the two rewrites discussed in Section 4: a single stage and three stages. As with the AND-gate discussed above, we see a threshold behavior as P_{circuit}

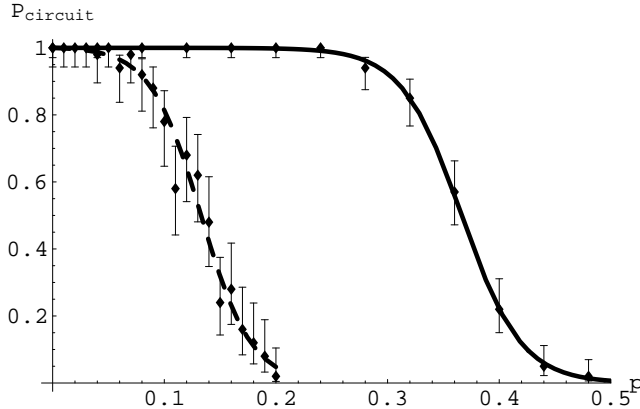


Figure 15: Probability P_{circuit} to be able to find a correct circuit for a 3-bit adder in a crossbar as a function of defect probability p . The points show the estimates from the simulation runs, with error bars indicating the 95% confidence intervals in the estimates of P_{circuit} . The curves show the maximum-likelihood fits of a sigmoid function, for the single and three-stage adders (dashed and solid, respectively). The crossbar sizes for the two circuits are 768 and 896 for the single and three-stages adders, respectively.

drops abruptly over a fairly short range of p values. The 3-stage adder can tolerate higher defect rates than the single-stage implementation.

As p increases, the probability of being able to find a given circuit monotonically decreases. As a simple summary of the results from multiple sets of trials, each using a different value of p to generate defective crossbars, we use a two-parameter sigmoidal form to relate P_{circuit} to p . Specifically, let

$$S(x) = \frac{1}{1 + e^{a(x-b)}} \quad (7)$$

Here the parameter a determines the sharpness of the threshold and b its location. Since P_{circuit} is zero when $p = 1$, and $P_{\text{circuit}} = 1$ when $p = 0$ (provided the crossbar is at least the minimum size required for the circuit), we shift and scale this sigmoid to match these extremes. Thus we use

$$P_{\text{circuit}} = \frac{S(p) - S(1)}{S(0) - S(1)} \quad (8)$$

With multiple sets of trials, we select a and b to maximize the likelihood of obtaining the observed results from the simulation. Since each trial is independent of the others, this amounts to maximizing the product of the individual likelihoods, described above, for each p value. Fig. 15 shows examples of the resulting fits.

To understand the existence of this threshold behavior, and how it depends on the circuit and crossbar area, consider a simplified version of the allocation in which we ignore all constraints on the locations of the functioning diodes. By ignoring these constraints, we obtain an upper bound on P_{circuit} and rough guides to the location and steepness of the threshold. Specifically, a crossbar with area A has k defective junctions with probability $\text{Bi}(A, k; p)$ given by Eq. (6). The expected number of defects is $\mu = Ap$ with standard deviation $\sigma = \sqrt{Ap(1-p)}$. A circuit requiring d diode junctions is very likely to exist when the number of defects is likely to be less than $A - d$. Conversely, when the number of defects is usually larger than this value, the crossbar is unlikely to be able to implement the circuit. More precisely, when $A - d$ is several standard deviations above or below μ , $P_{\text{circuit}} \approx 1$ or 0 , respectively. This discussion predicts the threshold near the value of p for which $\mu = A - d$, i.e.,

$$p \approx 1 - d/A \tag{9}$$

The change between these extremes takes place mainly over a range of p values corresponding to about a standard deviation around the mean, i.e., from the value where $\mu + \sigma = A - d$ to that where $\mu - \sigma = A - d$. The corresponding range in p values is $\frac{1}{A+1} \sqrt{1 + 4d - 4d^2/A}$ or $\sim \sqrt{1 + 4d}/A$ for large areas.

These specific values, derived by ignoring all constraints on the locations of the functioning devices, differ from the location and width of the threshold seen in the simulations. Nevertheless, they give some qualitative insight into the behaviors we observe. For instance, as the crossbar area increases, the threshold moves to larger values of p : as one would expect, larger crossbars provide more chances to find the required number of functioning junctions. The threshold width is small when the area is close to the minimum possible (i.e., just enough to hold the required number of functioning diodes) or when the area is very large. For a given area, comparing two implementations with different numbers of diodes, we see the implementation with more diodes, i.e., larger d , has a lower threshold: it is less tolerant of defects. As a final observation, consider the scaling to larger circuits (e.g., k -bit adders for $k > 3$). Taking the crossbar area to be a fixed factor larger than the required number of diodes, i.e., $A = rd$ gives a fixed threshold location of $1 - r$ while threshold width decreases as $O(1/\sqrt{d})$. That is, for larger circuits the threshold behavior becomes sharper.

4.2.2 Crossbar Area

Fig. 15 shows the behavior for a fixed size crossbar. Since the single and 3-stage adder circuits require different areas, it is also useful to compare the area required to obtain a fixed value of P_{circuit} , as we showed for the AND gate in Fig. 6. This raises the question of whether to adjust the shape of the crossbar as well as its area. A successful circuit requires not only enough functioning junctions, but also the ability to properly connect them to each other and the inputs and outputs. Thus P_{circuit} is, in general, a function separately of the three numbers characterizing the crossbar: the number of columns, the number of rows to be allocated for forming AND operations on inputs, and the number

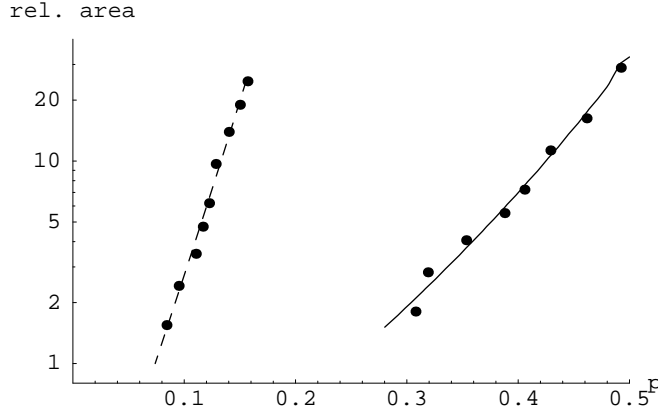


Figure 16: Relative area required to have at least 90% probability to be able to find a correct 3-bit adder circuit as a function of defect probability p , based on interpolating the 3-parameter fit to the simulation results. The dashed and solid curves correspond to single and three-stage adders, respectively. The points correspond to values estimated from individual sigmoid fits to results from each crossbar area. Areas are relative to that required for a single-stage adder on a defect-free crossbar.

of rows to form OR's on the results of the ANDs as outputs of the circuit. For example, the one-stage 3-bit adder uses 12 input wires (two for each of the 3 input bits for each of the two numbers to be added) and 4 outputs, for a total of 16 rows. It uses 31 columns to form the required logic operations. Thus a crossbar with fewer than 16 rows could never implement this circuit, no matter how many columns or how low a defect rate it had. More generally, scaling up the number of rows and columns by the same factor may not be the best way to improve performance from a given crossbar area. For example, it may be better to increase number of rows more than number of columns for an implementation requiring many diodes on the same row, vs. a different implementation using many diodes on a single column.

For simplicity, we focus on the behavior of P_{circuit} among crossbars of approximately the same shape (i.e., ratio of number of columns to numbers of rows allocated for the AND and OR operations) but with different areas and defect rates.

To estimate the behavior of the adder circuit for different areas, we used the sigmoidal fit of Eq. (8). This fit has two parameters, a and b , characterizing the width and location of the transition from $P_{\text{circuit}} \approx 1$ to $P_{\text{circuit}} \approx 0$. For definiteness, we consider arrays of fixed shapes as we vary their size, and show the resulting behavior in Fig. 16. Specifically, for the single-stage adder, we examined arrays whose numbers of columns, input rows and output rows had the ratios 4 : 2 : 1. By comparison, the minimum size array that can implement

the circuit, with 31 columns, 12 inputs and 4 outputs, has ratios $31 : 12 : 4 = 7.75 : 3 : 1$. Thus the shape we use allocates relatively more of the array area to increasing the number of rows, especially those for the outputs, than would be the case from uniformly scaling up the minimum array. This allocation is beneficial because the single-stage circuit is particularly sensitive to defects in the rows due to the need for large numbers of functioning diodes, especially for the outputs, as seen in Fig. 8. For the three-stage adder, various shapes close to scaled-up versions of the minimum size array (25 columns, 19 inputs, 11 outputs) had similar behaviors. As a convenient ratio to simulate with different sizes, we used $28 : 20 : 12$.

Over the range of crossbar areas we examined via simulation, the transition width had only small variation with area. On the other hand, the location of the threshold increased with area. Motivated by the discussion leading to Eq. (9), we suppose the sigmoid parameters a, b vary with area A according to

$$\begin{aligned} a &= \alpha A^\delta \\ b &= 1 - \beta A^{-\eta} \end{aligned}$$

where α, δ, β and η are parameters with nonnegative values. We fit four parameters, α, δ, β and η , to the simulation results from crossbars of various areas and defect rates to produce a single functional form relating P_{circuit} to A and p for a given circuit and array shape. The resulting functional form fits the results for single areas about as well as sigmoid functions optimized individually for each area. This fit then allows interpolating the behavior for other areas and defect rates than those evaluated via the simulation. In particular, it allows estimating the crossbar area required to have at least a given desired yield, i.e., value of P_{circuit} . For instance, Fig. 16 shows the resulting estimates for the area required to achieve $P_{\text{circuit}} = 90\%$. We see the single-stage adder is much more sensitive to defects, so requires larger areas to compensate.

The lowest points on the curves in Fig. 16 correspond to the minimum crossbar size that can implement the adder: 496 and 750 for the one and three-stage circuits, respectively. These values do not occur at $p = 0$ because the minimum areas are determined by the required numbers of logic operations, inputs and outputs rather than the number of diode junctions. Hence even the minimum area crossbars can tolerate some defective junctions. Comparing the two adder implementations, we see that for $p < 0.085$, the single-stage adder gives 90% success with crossbars whose size is too small to also implement the three-stage adder. For $0.085 < p < 0.3$, the three-stage adder on the smallest possible crossbar that can implement it gives over 90% success, while the area required for the single-stage adder increases significantly. With $p > 0.3$, required areas of both circuits increase, though the three-stage implementation requires much less area. This discussion illustrates how achievable defect rate, choice of circuit implementation and available crossbar area interact to determine the circuit yield.

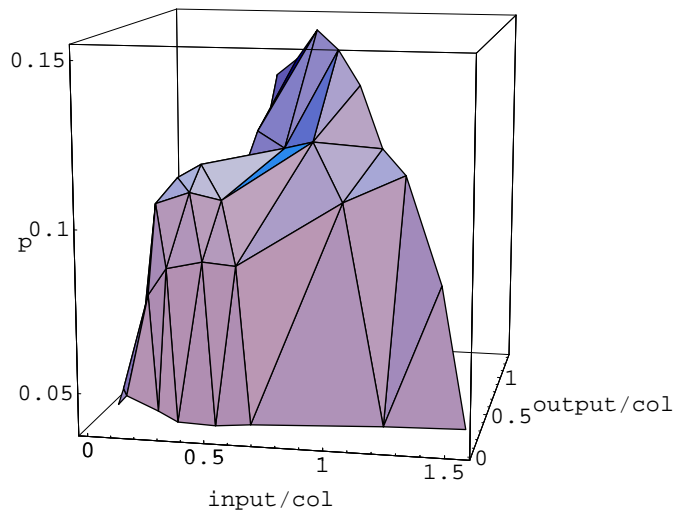


Figure 17: Defect rate, p , at which a single-stage 3-bit adder circuit can be found with 90% probability as a function of array shape for area equal to 1728. The shape is specified by the ratios of numbers of input and output rows to the number of columns.

4.2.3 Array Shape

In addition to the area of a crossbar array, its shape also influences the likelihood of being able to implement a circuit in spite of defects. Fig. 17 shows an example of the effect of different shapes for implementing the single-stage 3-bit adder, shown in Fig. 8. In this example, we examined various shaped arrays, all with the same area, 1728, which is about 3.5 times the minimum array size for this circuit. For each shape, we fit the sigmoid of Eq. (8) to the simulation results and used this fit to estimate the defect rate at which there was 90% probability to be able to implement the circuit. The shapes cover a range from a very wide array (108 columns, 12 inputs, 4 outputs) to the narrowest possible arrays with area 1728 still capable of implementing the circuit even when there are no defects, which have 32 columns and range from having 12 inputs, 42 outputs to 50 inputs and 4 outputs.

The best performance in Fig. 17 is for the array with 32 columns, 24 inputs and 30 outputs, giving at least 90% probability to produce a circuit with $p \leq 0.15$. This shape tolerates about 3 times as many defects as the worst shapes shown in the figure (which allocate only 4 wires to the outputs). Because the one-stage adder requires many diode connections on the outputs, this best performing array shape devotes a relatively large portion of the area to redundant output wires. By contrast, the shape used in Fig. 16 to illustrate behavior as a function of area arrays whose numbers of columns, input rows

and output rows had the ratios 4 : 2 : 1, a shape with intermediate performance among those shown in Fig. 17, i.e., allowing $p \leq 0.11$.

We also found variation due to shape in the three-stage adder, and that the best choice of shape varies somewhat with array size.

4.2.4 Scaling and Allocation Run Time

For comparison, we also examined 1, 2 and 4-bit adder circuits. They gave the same qualitative behaviors, and show the threshold behavior becomes more abrupt as circuit size increases. Thus the threshold becomes more significant as circuit size increases, thereby giving a useful design criterion for the maximum allowable defect rate for a given desired circuit and crossbar area.

For a given circuit, we found the typical run time of the allocation algorithm increases with p up to the threshold region. For even higher p values, most crossbars cannot implement the circuit and the algorithm terminates by reaching the bound on its run time. Nevertheless, to examine the allocation cost behavior for larger p , we also ran the algorithm to completion (i.e., with no bound on the run time) for a smaller circuit, namely a 1-bit adder. We found that the median allocation cost peaked near the threshold at which $P_{\text{circuit}} \approx 0.5$ and then decreased for larger p values, as the additional constraints introduced with additional defects allows the allocation algorithm to prune large sets of possibilities and more rapidly conclude no implementation is possible. This behavior is consistent with that seen in many other studies of combinatorial search problems [11]. That is, as problems become more constrained, there is an abrupt transition from almost always to almost never having a solution, and the typical search cost for a variety of search methods peaks near this transition.

5 Discussion

In summary, we examined the ability to map logic circuits, such as adders, onto a crossbar architecture in spite of numerous independent defects. The crossbar is a feasible approach to molecular electronics so our results show how they could be used to create molecular-scale logic. We also showed how different, logically-equivalent, choices for the mapping differ in their tolerance for defects and use of circuit area. We thus illustrate a range of design trade-offs involved in developing systems based on molecular electronics.

Quantitatively, for both a single gate operation and adder circuits, the likelihood of being able to tolerate defects shows a threshold behavior, i.e., changing abruptly from near one to near zero over a relatively small range of defect rates (for a given crossbar size) or over a small range of crossbar area (for a given defect rate). These thresholds become sharper as larger circuits are considered. Thus identifying the threshold locations gives useful design guidelines for the complexity of circuits feasible to implement at a given level of ability to fabricate molecular-scale crossbars (i.e., level of defects and size of crossbar possible to achieve during manufacture).

More generally, thresholds in behavior are seen in many combinatorial structures, such as random graphs [8], and related search algorithms [11]. These illustrate how definite behaviors can arise from unreliable, statistical systems. Hence identifying the locations of relevant performance thresholds is a useful design principle for molecular circuits, which can have many components (exploiting their high density) but are also likely to have relatively high defect densities.

These thresholds are also associated with large computational costs to determine whether a consistent structure exists. In the context of molecular circuits, identifying the defect rates corresponding to the thresholds provides a practical limit above which not only is yield low but it is likely that attempting to allocate a circuit would be computationally expensive, with this cost growing exponentially with the size of the circuit.

The map from logical circuit to the crossbar is an example of a constraint-satisfaction problem. In general, this consists of a set of variables, with associated values, and a set of constraints. A solution consists of a value for each variable such that all constraints are satisfied. Since the number of possible assignments grows rapidly with the size of the problem (e.g., number of variables), the computational cost of finding a solution can be prohibitive for large problems. In the context of designing molecular circuits, this observation highlights the importance of identifying not only whether an implementation exists but also the computational effort required to find such an implementation for a given set of defects. As with other constraint satisfaction problems, typically, circuits near the thresholds require the most search to find a viable circuit or conclude there are no viable circuits. Finally, logic formulas have a variety of equivalent expressions – so even if one fails to produce a circuit, others might work, as shown in Section 3 for a single k -input AND gate vs. $k-1$ 2-input gates evaluating the same logic formula. As with other combinatorial searches, reducing this computational cost, e.g., by using a greedy local search (e.g., taking the gates in some fixed order and always using the first available column wire to implement the next gate in the list) will sometimes fail to find an implementation even though one exists. This choice thus amounts to another design trade-off between computational cost to respond to defects and the cost of reducing the defect rate in fabrication.

Several extensions to this work would be interesting to investigate. First, we examined behavior with respect to independent point defects in the crossbar. It would be useful to examine how defects actually arise in manufacturing and whether they differ significantly from the independence we assumed. For example, defects might short out an entire row or column of the crossbar, or appear with strong spatial correlations. As with studies of other stochastic systems, these different error models will likely give rise to similar qualitative behaviors, including the existence of thresholds in implementation feasibility, but with different quantitative values. From an design perspective, it would be useful to identify how the threshold locations using realistic models of defect locations compare with the currently achievable defect rates as an indication of the range of logic circuits likely to be implementable in the near future.

A second extension to this work involves the rewrites for the circuit. As described with the AND gates and adder circuits, we examined two quite different implementations for each logic circuit. We showed these rewrites have different resource requirements (i.e., crossbar area) and tolerance for defects. Another application for rewrites would be to make only minor modifications in a given circuit with the goal of removing just those parts of the original circuit that are most difficult to map to a defective crossbar. For instance, the fourth from last row of the adder in Fig. 8, computing S_2 , requires 16 functioning diode junctions on the row. This is more than is required on any other row or column of the circuit. In a defective crossbar, the need to find one row with at least 16 functioning devices is a major limitation that could be removed by simply rewriting just that part of the circuit. For instance, instead of using one row to implement the logical-OR of 16 inputs, we could split the function into two rows, each performing a logical-OR of 8 inputs, followed by a logical-OR of those results to form the final value for S_2 . This would improve the chances of mapping the circuit to a defective crossbar (as we saw for similar rewrites with AND gates in Section 3). This rewrite also increases the size and delay of the circuit, though not as much as the entirely different rewrite of Fig. 7. This example illustrates how rewrites could be targeted to remove just the parts of the circuit least tolerant of crossbar defects, thereby giving a range of options suitable for crossbars with different defect rates.

We only considered logically-equivalent rewrites: e.g., the different implementations of the AND-gate or adder circuit compute exactly the same logical function of their possible inputs. An additional possibility is allowing some non-equivalent rewrites, particularly in conjunction with fault tolerance in a higher-level system architecture of which the molecular device is only one part. In this case, an occasional error in the circuit (e.g., due to undetected or new defects, or noisy operation) may be tolerable. As another application, logic circuits used for pattern recognition based on combinations of different sensors (e.g., each detecting a different chemical or concentration in the environment) could perform quite well even with a non-equivalent rewrite that gives different outputs from the ideal circuit on a small subset of inputs. For instance, suppose a device consists of three sensors for different concentrations of a chemical (“low”, “medium” and “high”) based on receptors with differing affinities and saturation levels. Here the low concentration sensor will be active whenever the medium or high are, and the medium one will be active whenever the high one is. Thus a rewrite of a pattern-recognition circuit that changes the output for the input corresponding to the low and high sensors on but the medium sensor off would have no practical significance since that combination of inputs would not arise in its use with this particular set of sensors. More generally, extending consideration to a limited set on non-equivalent rewrites could provide substantial improvements in ability to implement the circuit (by being below a higher threshold) than would be possible among logically equivalent rewrites.

We focus on circuit area as the main cost criterion in our discussion. In practice, other properties may also be important to consider in developing practical molecular circuit designs. For instance, rewrites not only differ in the circuit

area they require but also in propagation delay and their need for additional components such as restoring latches. Moreover, we have not included the cost to test the crossbar for defects and the possibility that this cost could vary with accuracy of this test. In particular, false positives unnecessarily reduce the available circuit area and false negatives could result in erroneous outputs for some inputs. Including these criteria as part of the overall system manufacturing cost could alter the choice of the best design.

Acknowledgements

We thank Phil Kuekes and Li Zhang for helpful discussions.

References

- [1] Yong Chen et al. Nanoscale molecular-switch crossbar circuits. *Nanotechnology*, 14:462–468, 2003.
- [2] Yong Chen and R. Stanley Williams. Nanoscale patterning for the formation of extensive wires. US Patent 6,294,450, September 25 2001.
- [3] C. P. Collier et al. Electronically configurable molecular-based logic gates. *Science*, 285:391–394, 1999.
- [4] L. P. Cordella et al. An improved algorithm for matching large graphs. In *Proc. of the 3rd IAPR-TC-15 Intl. Workshop on Graph-Based Representations*, pages 149–159, 2001.
- [5] A. DeHon. Array-based architecture for FET-based nanoscale electronics. *IEEE Trans. on Nanotechnology*, 2:23–32, 2003.
- [6] Andre DeHon, Patrick Lincoln, and John E. Savage. Stochastic assembly of sublithographic nanoscale interfaces. *IEEE Trans. on Nanotechnology*, 2:165–174, 2003.
- [7] Yasser El-Sonbaty and M. A. Ismail. A graph-decomposition algorithm for graph optimal momomorphism. In A. F. Clark, editor, *Proc. of the 8th British Machine Vision Conference (BMVC97)*, 1997.
- [8] P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–61, 1960.
- [9] S. Goldstein and M. Budiu. Nanofabrics: Spatial computing using molecular electronics. In *Proc. of the 28th Intl. Symposium on Computer Architecture (ISCA)*, 2001.
- [10] James R. Heath, Philip J. Kuekes, Gregory S. Snider, and R. Stanley Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280:1716–1721, 1998.

- [11] Tad Hogg, Bernardo A. Huberman, and Colin P. Williams, editors. *Frontiers in Problem Solving: Phase Transitions and Complexity*, volume 81, Amsterdam, 1996. Elsevier. Special issue of *Artificial Intelligence*.
- [12] Yu Huang et al. Logic gates and computation from assembled nanowire building blocks. *Science*, 294:1313–1317, 2001.
- [13] P. J. Kuekes and R. S. Williams. Demultiplexer for a molecular wire crossbar network. US Patent 6,256,767, July 2001.
- [14] P. J. Kuekes, R. S. Williams, and J. R. Heath. Molecular wire crossbar memory. US Patent 6,128,214, Oct. 2000.
- [15] P. J. Kuekes, R. S. Williams, and J. R. Heath. Molecular-wire crossbar interconnect (mwci) for signal routing and communications. US Patent 6,314,019, Nov. 2001.
- [16] Nicholas A. Melosh et al. Ultrahigh-density nanowire lattices and circuits. *Science*, 300:112–115, 2003.
- [17] SIVALab. VF graph matching library. University of Naples “Federico II”, 2001.
- [18] M. Stan, P. Franzon, S. Goldstein, J. Lach, and M. Ziegler. Molecular electronics: From devices and interconnect to circuits and architecture. *Proc. of the IEEE*, 91:1940–1957, 2003.
- [19] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, volume 34 of *Ann. of Math. Stud.*, pages 43–98. Princeton University Press, 1956.
- [20] Matthew M. Ziegler and Mircea R. Stan. Design and analysis of crossbar circuits for molecular nanoelectronics. In *Proc. 2nd IEEE Conf. on Nanotechnology (NANO-2002)*, 2002.