Nanotechnology **15** (2004) 881–891 PII: S0957-4484(04)78404-4

CMOS-like logic in defective, nanoscale crossbars

Greg Snider, Philip Kuekes and R Stanley Williams

Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, USA

E-mail: snider.greg@hp.com

Received 22 March 2004
Published 9 June 2004
Online at stacks.iop.org/Nano/15/881
doi:10.1088/0957-4484/15/8/003

Abstract
We present an approach to building defect-tolerant, nanoscale compute fabrics out of assemblies of defective crossbars of configurable FETs and switches. The simplest structure, the complementary/symmetry array, can implement AND-OR-INVERT functions, which are powerful enough to implement general computation. These arrays can be combined to create logic blocks capable of implementing sum-of-product functions, and still larger computations, such as state machines, can be obtained by adding additional routing blocks. We demonstrate the defect tolerance of such structures through experimental studies of the compilation of a small microprocessor onto a crossbar fabric with varying defect rates and compiler mapping parameters.

(Some figures in this article are in colour only in the electronic version)

1. Introduction

Many proposed architectures for nanoscale electronics have focused on the crossbar (figure 1) because of its simplicity of fabrication and its inherent redundancy which supports defect tolerance [1–7]. All such architectures assume that each junction or crosspoint within the crossbar can be independently configured to activate an electronic device, such as a resistor, diode, or transistor. Such crossbars, if they can be built, would provide a building block for implementing very dense electronic systems.

A *crossbar* denotes both an interconnection topology and a fabrication geometry. Different device types (resistor, diode, transistor) can be incorporated into a crossbar, yielding different kinds of crossbars. Furthermore, micron-scale masking can be used to realize multiple device types in side-by-side regions within a single crossbar such that the nanowires maintain electrical conductivity across and between regions. We refer to these single-device type regions as *tiles*. Several adjacent tiles can be combined into larger function units, which are referred to as *composite tiles* or *blocks*. Alignment precision for masking, as compared to nanowire pitch, limits the minimum tile to roughly 10 × 10 junction regions of the crossbar.

Some devices have the desirable property of being *configurable*, which means that at some time after manufacturing, the devices at selected junctions may be independently *activated* or *deactivated*. An activated device (such as a resistor) functions as a normal electrical component in the crossbar circuit. A deactivated device appears to have vanished, functionally. A configurable resistor, for example, behaves as if it were a resistor in series with a switch, which may be opened or closed. When closed, the resistor connects the upper and lower wires at its junction in the crossbar; when the switch is open, the resistor has no effect on the circuit. *Configuring* the device is the act of setting the switch to be open or closed. This may be done electrically, optically, mechanically or by some other means. In the case of the crossbars fabricated at Hewlett-Packard Laboratories, configuration is done electrically, with particular voltages across the configurable device setting it to different configuration states. A device which is *reconfigurable* may be repeatedly activated and deactivated.

Nanoscale crossbars with reconfigurable resistors have been built and used to implement small memory and logic systems [1, 8]. Reconfigurable switch crossbars can be used to create latches capable of both inversion and signal restoration [9]. Inverting latches combined with configurable resistor and diode crossbars are sufficient to implement arbitrary logic and processing systems. Still, configurable, nanoscale transistors in a crossbar would provide a more

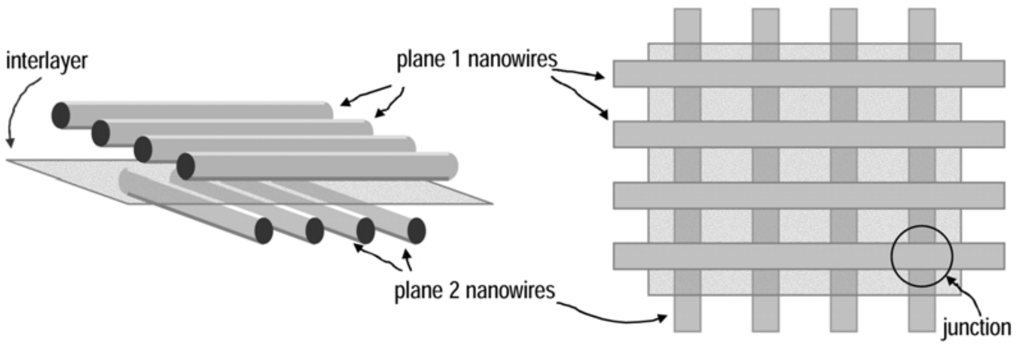

Figure 1. Schematic view of a nanoscale crossbar from two different perspectives. Junctions may be independently configured to behave as electronic devices. A chemical 'interlayer' between the two planes of parallel nanowires, along with the nanowire composition, determine the type of devices that may be configured.

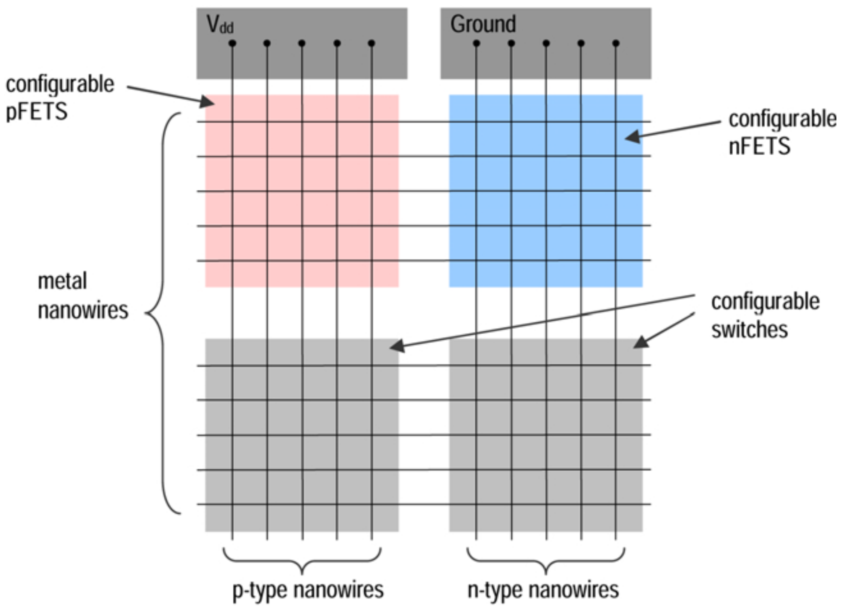

Figure 2. Logic block implemented with the complementary/symmetry array. Each junction in the pink quadrant may be independently configured to implement a p-FET, while each junction in the blue quadrant may be configured to implement an n-FET. The junctions in the two lower quadrants may be configured to be 'closed' switches, representing a low impedance path between the two nanowires defining the junction.

powerful foundation for electronic computation. A system built out of such crossbars might not require reconfigurability (which, to date, has been more difficult to implement reliably than simple configuration), and a vast fabric of transistors with gain would offer considerable flexibility to system architects.

Several authors have proposed building field-effect transistors (FETs) out of carbon nanotubes [10–12] or silicon nanowires [13–15]. The nanowire approach uses one nanowire (perhaps made of metal) as the gate electrode, crossing over the other nanowire (perhaps a doped semiconductor) which forms the source and drain [2]. The channel would be formed in the semiconductor nanowire in the region near the junction. There are, of course, many hurdles to actually implementing an FET in such a tiny area. For example, a semiconductor nanowire with typical doping levels of 10^{18} atoms of boron or arsenic per cubic centimetre would have, on average, only 0.1 dopant atom in a 5 nm × 5 nm junction [16]. As a result, FETs at those dimensions might not behave predictably, if they would even function at all. Higher doping levels would increase the dopant density within the junction, but there are thermodynamic upper bounds on the effective level that can be achieved. Another strategy for injecting charge carriers will most likely be needed. Leakage currents due to quantum mechanical tunnelling of electrons through insulation between the gate and the channel must also be kept in check in order to hold power consumption to an acceptable level [17].

This paper focuses on the architectural possibilities that would arise from crossbars of configurable FETs, assuming both n-channel FETs (n-FETs) and p-channel FETs (p-FETs). In particular, we show how mosaics of such crossbars can implement CMOS-like logic capable of universal computation. The functional redundancy in such architectures also make them very forgiving of defects—a likely characteristic of fabrication at the nanoscale—providing that one can characterize and locate the defects prior to configuration. For simplicity, some important architectural considerations such as configuration and interfacing to a micron-scale environment are not addressed here—these have been covered to some degree by other authors [2, 18].

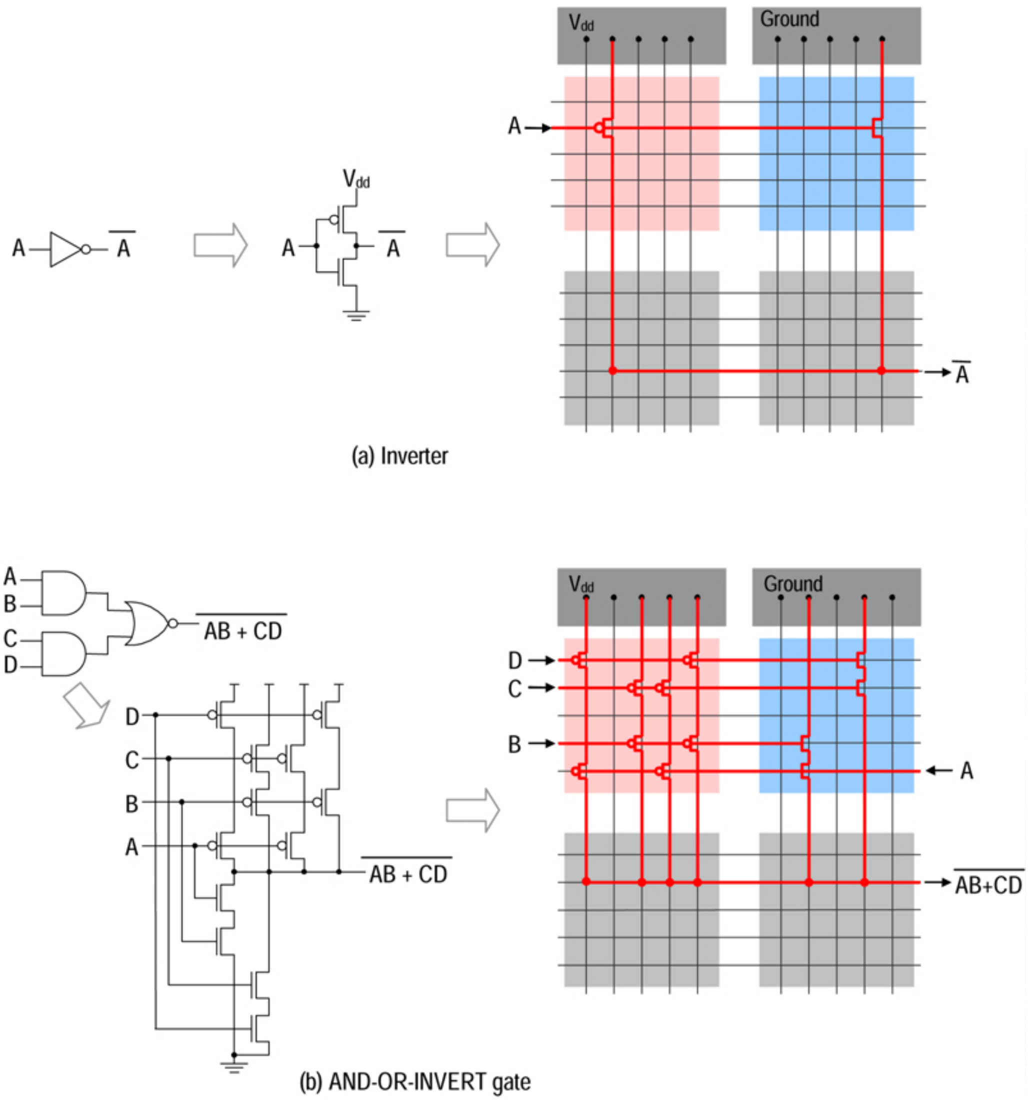

Figure 3. Implementing logic by selectively configuring junctions.

2. Logic block

The basic building block for creating logic out of n-FET and p-FET arrays, called the complementary/symmetry array, is shown in figure 2. The horizontal and vertical lines in that figure represent nanowires, with the horizontal wires in one plane and the vertical wires in another. The horizontal nanowires are metallic and the vertical are semiconductors, with the vertical wires in the left half of the structure p-type, and the remaining vertical wires n-type. The nanowires are divided into four different quadrants, each quadrant possessing different electrical properties due to the nature of the nanowire types in that quadrant and the chemical properties of the interlayer used in that region.

The dark grey rectangles at the top of figure 2 represent micron-scale structures (perhaps implemented on an underlying silicon substrate) that supply power and ground to the array. The small black circles represent connections between the structures in two different planes. In this case, they illustrate that the leftmost vertical wires are electrically connected to the V_{dd} power supply, while the rightmost vertical wires are electrically connected to ground. The light grey quadrants at the bottom of the figure represent configurable switches: each junction is normally 'open' with a high impedance path between the wires in the two planes, but can be electrically configured to be 'closed' (a low impedance path). The pink quadrant junctions are also normally open, but each junction here can be configured to be a p-FET, with the gate implemented with the horizontal wire and the source and drain implemented with the vertical wire. Each of the blue quadrant junctions, also normally open, can be configured to be an n-FET, with the gate implemented with the horizontal wire and the source and drain implemented by the vertical wire. Although each of the four quadrants is illustrated here as being sharply defined, this is not necessary; each quadrant may be 'fuzzily' implemented (due to difficulties in aligning micron-scale masks) with, say, defective wires near the boundaries, and each quadrant may be a different size. The only requirement is that junctions can be characterized and defective wires and junctions located. Note that the interlayer may necessarily be different in each quadrant through the use of micron-scale masking, though a later section will suggest a simpler fabrication scheme.

Figure 3(a) illustrates how the array can implement an inverter by configuring a single junction in each quadrant. The red dots in the light grey quadrants represent junctions that have been configured to be closed crosspoint switches. The p-FET symbol in the pink quadrant represents a junction configured to be a p-FET, with the horizontal wire of the junction performing the function of the gate, and the vertical wire the source and drain. The n-FET symbol in the blue quadrant represents a junction configured to be an n-FET. With this configuration, an input signal, 'A', connects to the gates of the configured p-FET and n-FET. The two FETs are also connected with closed crosspoint switches in the two lower quadrants. The result is a circuit that implements a CMOS-like inverter. The inverted output is then available to be routed outside of the array.

An AND-OR-INVERT gate is shown in figure 3(b), implemented by selective configuration of junctions in each of the quadrants. Other logic functions can be implemented in a straightforward manner by decomposing the desired function into two sets of minterms (one set for the p-FET quadrant and one set for the n-FET quadrant), and then using conventional algorithms (such as Quine–McClusky) to minimize the FET implementation for each set. In general, such synthesized functions will also require the complement of each of its input signals, easily supplied by the inverter of figure 3(a). Note that all communication with the array is done through the horizontal metal wires, leaving the vertical semiconductor wires relatively short. This is desirable since the semiconductor wires are likely to be several orders of magnitude more resistive than the metal wires; keeping them short helps to minimize signal degradation and to minimize propagation delay due to resistive–capacitive (RC) time constants.

The array is flexible with respect to the routing of signals into and out of it. A particular input signal may be brought in at any of the horizontal nanowires in the top two quadrants, and an output driven out on any of the horizontal nanowires in the bottom two quadrants, limited only by the ability to allocate junctions within the array to implement the computation. If the array is defect-free, the inputs can be freely permuted, as can the outputs, and signals may be connected on either side of the array.

The array is also robust with respect to defective junctions since defects can be avoided by routing around them, provided that the defective junctions can be determined prior to configuration.

3. Compute fabrics

A useful modification of the basic logic block is shown in figure 4. Since logic functions implemented using a sum-of-products representation generally require input variables to be presented in both true and complemented forms, most signals produced by logic gates will, at some point, need to be inverted. Sending both the true and complemented forms of each signal through an external routing network will consume twice as many routing resources than one would prefer. This variation allows each signal to be inverted locally (and only if necessary for implementing the function), so that only a true form of each signal need be sent through the routing network. Hence, the block can implement any general logic function

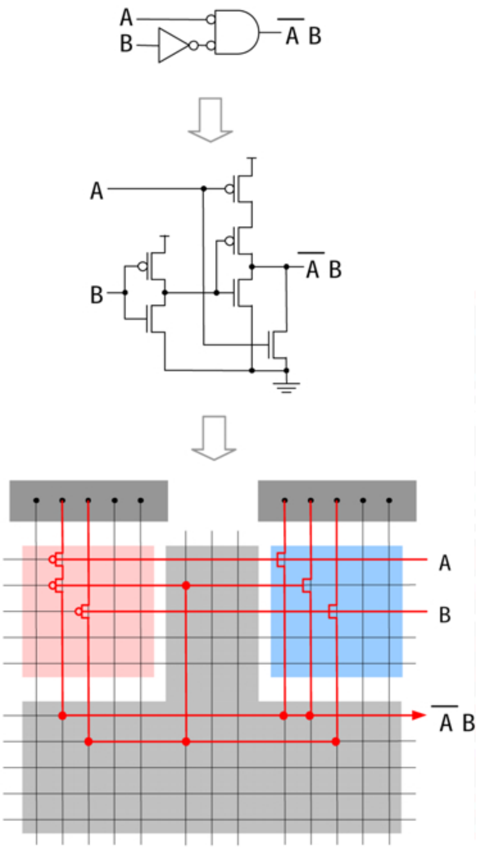

Figure 4. Modified logic block. The additional switches between the configurable p-FETs and n-FETs provides separation between the microwires supplying V_{dd} and ground (dark grey rectangles) and also supply internal routing for creating local inverters in the array. In this example, the input signal B is inverted within the block.

when presented only with the uncomplemented form of each input variable, limited only by the number of usable FETs and the fan-in and fan-out constraints of the block.

Using reflections and rotations, two basic arrays (figure 3) can be interleaved to form the configurable bidirectional buffer block shown in figure 5. This block can take a signal on any wire on either side, amplify it (with inversion), and drive it out on any wire on the other side. The choice of direction on each wire and the mapping of an input wire to an output wire are independent, making this a useful component in building a routing network.

Larger computational structures can be built by combining the arrays with configurable switch tiles for routing. One example is the logic fabric shown in figure 6. The routing structure on the right provides local communication between the logic arrays on the left as well as communication with external signals.

One can hierarchically combine structures, such as the logic block, bidirectional buffer block and routing tiles, to build still larger fabrics as shown in figure 7.

4. Defects

Because of the regular, symmetric structure of a crossbar and the large number of redundant components contained in its junctions, one would expect crossbar structures to be highly

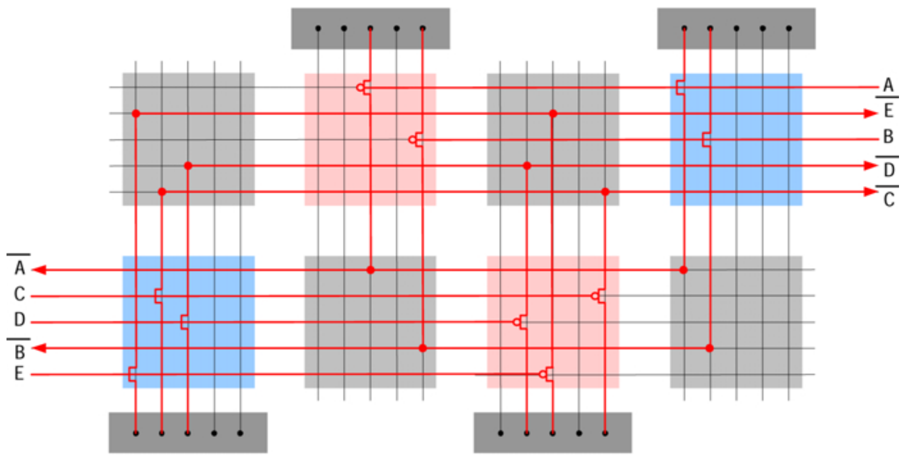

Figure 5. Bidirectional buffer block. Signals entering one side can be regenerated (with an inversion) and driven out the opposite side.

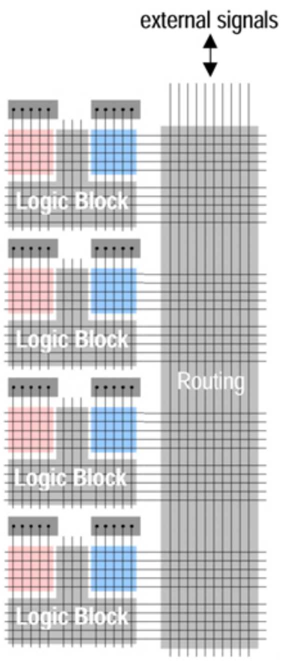

Figure 6. Logic fabric: logic and routing resources combined to form a larger computational structure.

tolerant of defects, *if* those defects can be determined prior to the mapping of a circuit onto the crossbar. 'Stuck open' junctions—junctions that are incapable of being configured as an electronic component but that do not short out the pair of wires that form it—have the smallest impact since they generally do not restrict the use of the wire pair. 'Stuck closed' junctions—junctions that are either shorted or permanently configured as an electronic component—are more problematic, since they require either the mandatory use of that junction in a circuit mapped onto the crossbar, or that the wires making up that junction be marked unusable. Wires may also be broken or shorted, easily handled by marking such wires as unusable during circuit mapping.

This section explores the effect that defects have on compute fabric utilization and on the algorithms needed to map circuits onto the fabric. We assume that defects can be located prior to mapping using a separate test procedure.

4.1. Methodology

We performed a set of simulations that explored the layout of a single application onto a particular compute fabric while varying the defect probability and two layout algorithm parameters. A hardware compiler was used to automatically map the application (written in the C language, which we use as our hardware description language) onto the target fabric.

The target fabric consisted of 64 identical logic blocks implementing the complementary/symmetry array (figure 8) combined with a routing network similar to the fabric in figure 7. The sizes of the n-FET and p-FET regions were intentionally quite large to ensure that they could feasibly be fabricated using conventional masking processes. The n-FET and p-FET arrays differ in size due the approach we have taken in clustering functions within arrays (discussed in section 4.2). Abundant routing resources were supplied to ensure that any failure to map the application onto the fabric was due to an inability to allocate resources within the logic blocks and not because of an inability to route. Defects were limited to the 'stuck-open' type, randomly distributed throughout all junctions formed by the crossing of two nanowires. Junctions formed by a nanowire crossing a microwire were assumed to be non-defective.

The application was a simple, 4-bit microprocessor implemented in 143 lines of C code. The instruction memory for the processor contained a short program for implementing a two-pole, low pass filter using 18 words of 6 bits; this memory was included in the compiled circuit.

Three parameters distinguished a single compilation:

(1) *Defect probability (p)*: the probability of a nanowire/nanowire junction being 'stuck open'; this varied from 0% to 20% in increments of 2%.
(2) *Maximum function inputs (m)*: essentially the maximum of number of input variables that synthesis was allowed in creating a sum-of-product representation of a logic function. For example, AND gates with up to m inputs could be fabricated while AND gates with $(m + 1)$ inputs could not. The value of m also equals the maximum number of p-FETs (or n-FETs) that could be chained together (with the source of one connected to the drain of the next) along a single nanowire. Larger values of m

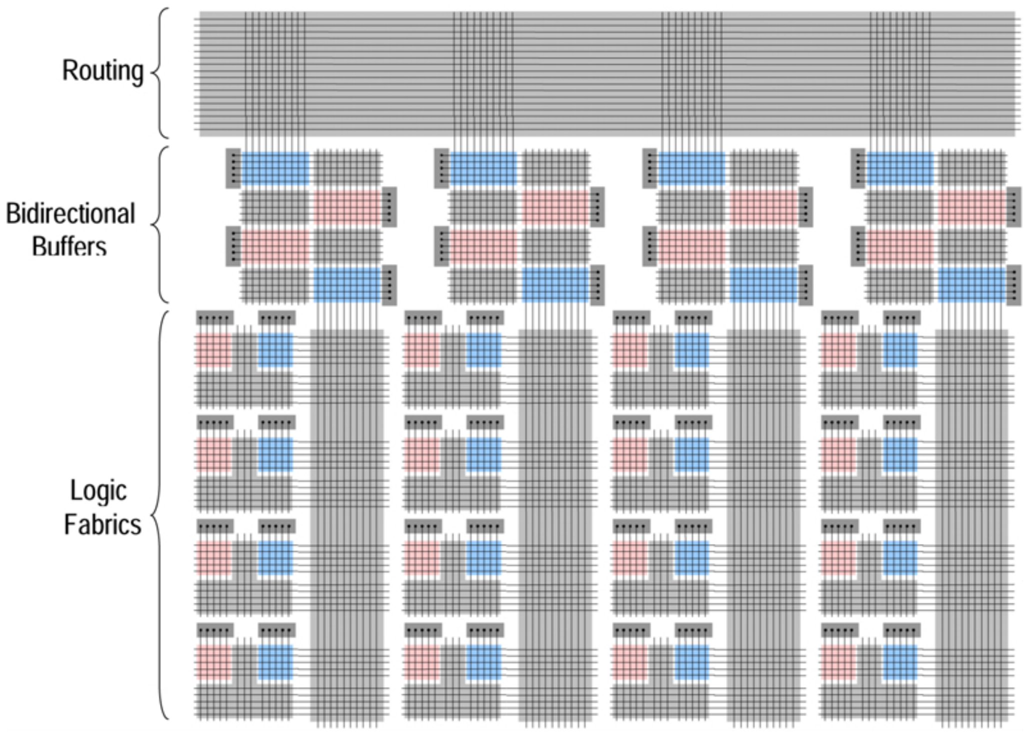

Figure 7. Larger computing fabrics may be built out of combinations of logic and routing blocks, hierarchically organizing them in a fractal-like pattern.

produce denser logic implementations, but these are also more difficult to map onto defective crossbars.

(3) *Cluster packing density (d)*: the maximum percentage of resources within a logic block that could be allocated for implementing logic gates. This varied from 100% (which would give maximum utilization on a defect-free fabric) down to 40% in steps of 20%. Reducing the density, d, provides more flexibility in mapping synthesized functions onto defective crossbars, but tends to increase the number of logic blocks required to implement the entire circuit.

An experiment consisted of 50 compilation attempts with the same set of experimental parameters (defect probability, maximum function inputs, cluster packing density). Each compile used a different pattern of random defects when the defect probability was non-zero. There were three possible outcomes of a single compilation.

(1) The application was successfully placed and routed on the target.
(2) Compilation failed because of insufficient resources within at least one logic block for allocating a logic function. This occurred when there were too many defects for the allocation algorithm to find a legal configuration.
(3) Compilation failed because of an insufficient number of logic blocks to hold all synthesized logic functions for the application. This could only occur with a cluster packing density of less than 100%.

4.2. Compilation

For compilation we used the Xax hardware compiler [19, 20] which compiles programs written in a subset of the C language

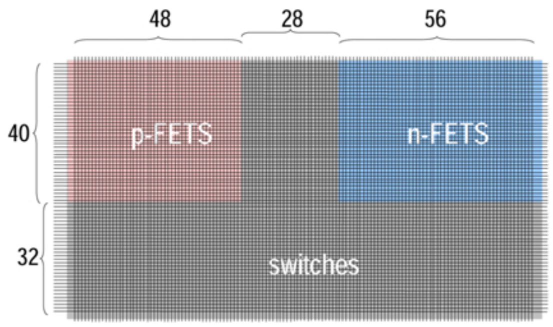

Figure 8. The size of the complementary/symmetry array used in the experiments.

down to circuits. We added additional modules to handle compilation onto various nanoscale targets, including those built from the complementary/symmetry array. Xax is also part of a larger design environment, called Nano, which provides a testbed for creating hypothetical computing fabrics out of mosaics of multiple types of tiles, compiling onto them, and doing logic- and SPICE-level simulation of the resulting circuits. Nano also allows us to inject random defects into our modelled architectures.

Figure 9 is a screenshot of our design environment, showing a half-adder circuit (written in C) compiled onto a logic block augmented with a small routing block on the right. The p-FET crossbar is coloured pink as in previous figures, and the n-FET crossbar is coloured blue. Light grey crossbars represent configurable switches. Junctions marked with red 'X's represent 'stuck open' junctions that the mapping software must avoid; in this particular case, 10% of the junctions in all crossbars have been randomly marked as defective. The

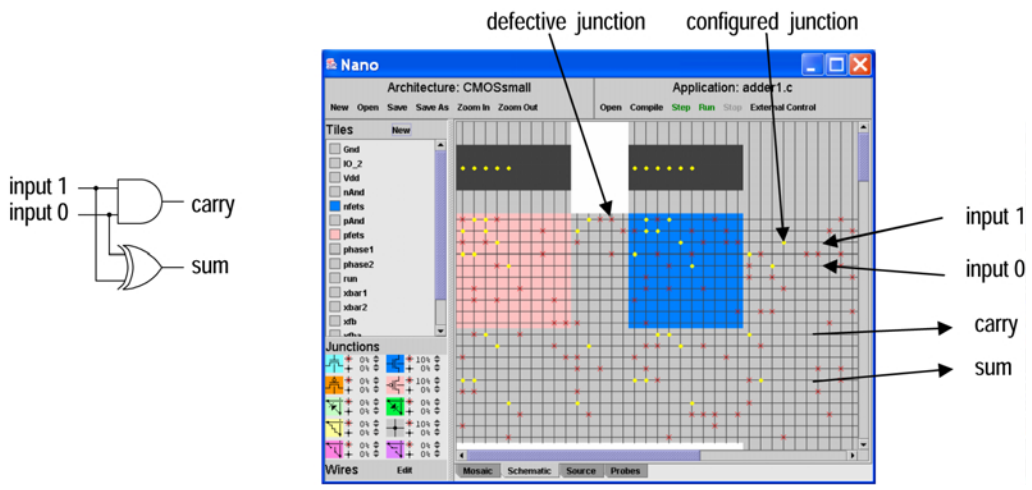

Figure 9. A half-adder implemented in the complementary/symmetry array. Red X's represent defective, 'stuck-open' junctions, and yellow dots represent configured junctions. The half adder was compiled from C, using a resource allocation algorithm to avoid defective junctions.

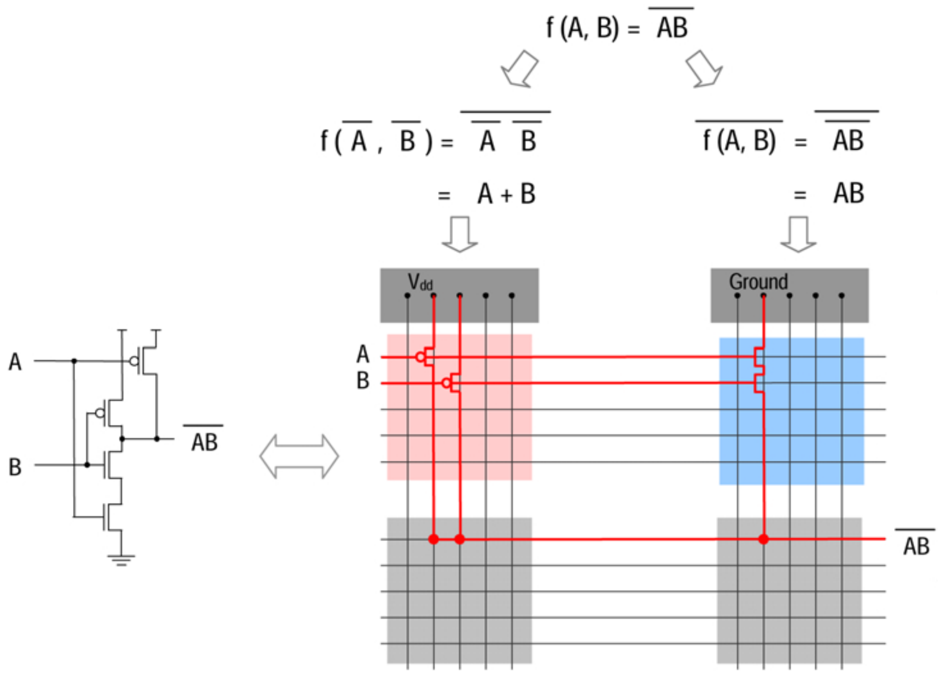
$f(A, B) = \overline{AB}$
$f(\overline{A}, \overline{B}) = \overline{\overline{A}\ \overline{B}}$
$= A + B$
$f(A, B) = \overline{\overline{AB}}$
$= AB$
Vdd
Ground
A
B
A
B
$\overline{AB}$
$\overline{AB}$

Figure 10. Implementing a logic function (a NAND gate) in the complementary/symmetry array.

environment displays configured junctions with small yellow circles, with the type of component created determined by the type of crossbar that the junction resides in (n-FET, p-FET, or switch).

The compiler front-end goes through a series of passes to translate an algorithm to a circuit consisting of simple logic gates and flip-flops. This circuit is then optimized using SIS logic synthesis software [21] to create a more efficient circuit built out of sum-of-product nodes and flip-flops. This representation is nearly ideal for our purposes since the complementary/symmetry array naturally implements an inverted sum-of-products (AND-OR-INVERT) gate.

The implementation of an AND-OR-INVERT gate in the fabric requires synthesis of an n-FET structure and p-FET structure. The strategy for doing so may be understood by analogy with conventional CMOS: for all possible input values, the output of the gate must either

- pull the output line to V_{dd}, or
- pull the output line to ground,

but never both at the same time (that would be a short from V_{dd} to ground) and never neither (that would leave the output floating). Since the p-FETs connect to V_{dd}, at least one p-FET path must turn on for $\overline{f(A, B, C)}$, and at least one n-FET path must turn on for $f(A, B, C)$. In other words, the functions implemented by the n-FET structure and the p-FET structure are complements of each other. We use the output from SIS to build the p-FET structure, inverting all of the inputs (since p-FETs invert) and implementing each product term with a single vertical chain of p-FETs, and then create the final sum by 'wired-ORing' the products (figure 11). To derive the n-FET structure, we take the function produced

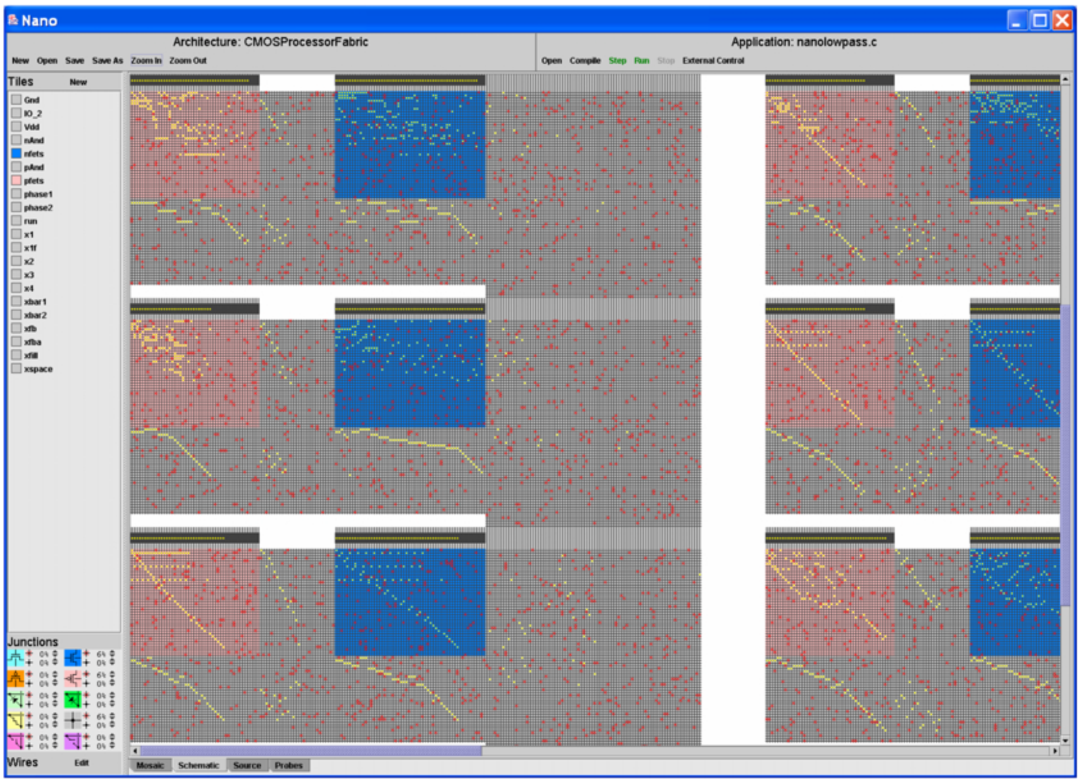
Nano
Architecture: CMOSProcessorFabric
Application: nanolowpass.c
New Open Save Save As Zoom In Zoom Out
Open Compile Step Run Stop External Control
Tiles New
Gnd
IO_2
Vdd
nAnd
nfets
pAnd
pfets
phase1
phase2
run
x1
x1f
x2
x3
x4
xbar1
xbar2
xfb
xfba
xfill
xspace
Junctions
Wires Edit
Mosaic Schematic Source Probes

Figure 11. A portion of the experimental fabric configured with the microprocessor application. Junctions marked with red 'x's are assumed defective; junctions with yellow dots are those which have been configured by the compiler.

by SIS, complement its output rather than its inputs, then invoke SIS again to produce an optimized sum-of-products representation of the complemented function. For example, if SIS produced the function $f(a, b, c)$, we would create the function $\overline{f(a, b, c)}$ and use SIS to produce an optimized sum-of-products implementation (figure 10). Each product term of this result would be implemented with a single chain of n-FETs, and the final sum would be 'wire-ORed' to the output of the p-FET structure.

It is unfortunate that the number of products for a function and its inverse are not generally the same, and the disparity between the two can grow as the number of variables increases; this puts a practical limit to the number of function variables that we can handle for any given function. The compiler can force SIS to respect an upper bound on the number of input variables using the *maximum function inputs* parameter.

After producing the n-FET and p-FET structures for each of the functions in our optimized circuit, we then group the functions together into logic blocks to minimize the number of arrays needed to implement the entire circuit. This process, known as clustering, is implemented using a simple greedy algorithm that groups functions together based on the number of input and output signals that they share—the more signals they have in common, the more attraction they share to the same cluster. The experimental parameter *cluster packing density* can be used to force sparser packing of gates within a cluster, thereby generally increasing the number of clusters, by limiting the percentage of resources within an array that are available for allocation. The clusters are then placed into logic blocks using simulated annealing.

Allocation of p-FETs, n-FETs and switches within a logic block is done using a pruned exhaustive search algorithm, described by Hogg and Snider [22].

Figure 11 shows an example of a compilation of the 4-bit microprocessor onto the crossbar fabric. In this particular case, 6% of all junctions have been randomly marked defective (red X's). The compiler routes around the defects, implementing logic functions by allocating working junctions (yellow dots) and wires.

4.3. Experimental results

The raw experimental results are shown in figure 12. Each graph shows the results of compiling the application onto the fabric with function inputs limited to 10, 8, 6 and 4. A single data point on a graph corresponds to 50 compiles of the application onto the target fabric with the parameters that distinguish it (defect probability, maximum function inputs, cluster packing density). Within a single graph, the effects of different cluster packing densities are shown as separate curves. If a graph is missing a curve for a particular cluster density (such as for densities less than 1.0 for 10 inputs) it is because there were insufficient resources in that case for the compiler to successfully map the application. Note that cluster packing densities less than 1.0 will tend to cause the implementation to 'spread out' in space, requiring more clusters (and logic blocks to contain them); hence the number of clusters required for each packing density is also shown in parentheses.

Compilation time for successful compiles was largely independent of defect rate, cluster density, and function inputs.

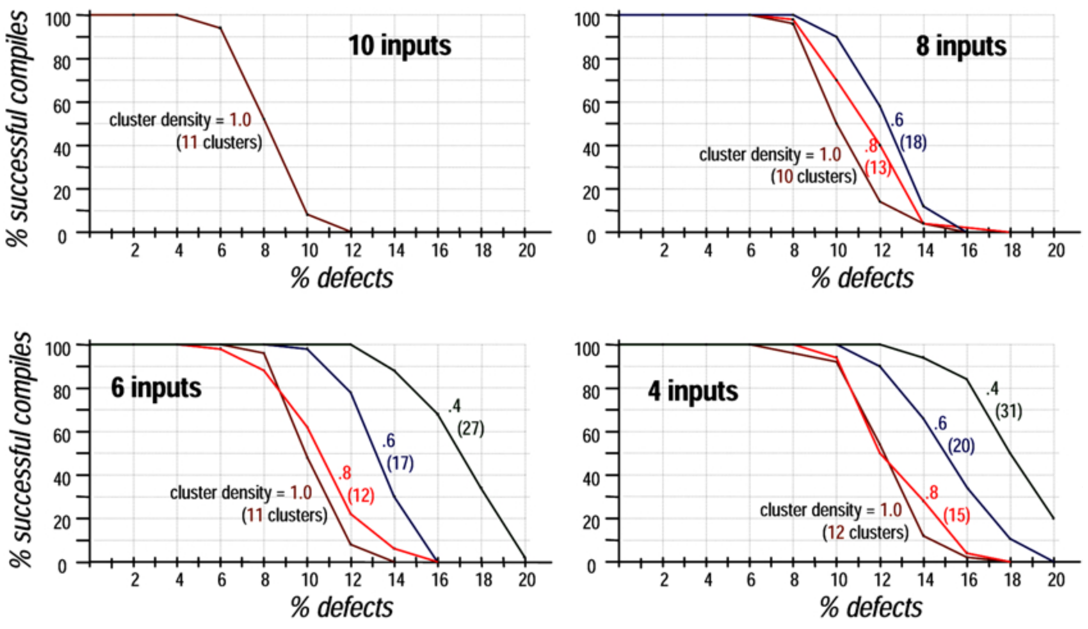

Figure 12. Exploration of mapping a 4-bit microprocessor onto a nano compute fabric, varying the defect density in the fabric and the mapping parameters (cluster density, maximum function inputs).

Table 1. Typical breakdown of time spent in each compiler pass.

Compiler pass	Time (s)
Parsing	1.0
Translation	1.0
Optimization	0.4
Synthesis (SIS)	18.6
Technology mapping	0.1
Clustering/placement	41.3
Resource allocation	5.1
Routing	5.0

Table 1 shows a typical breakdown of time spent on each of the compiler passes for the experiments. Note that only the final two passes of the compiler were dependent on a specific configuration of defects.

The application was also mapped, using SIS, to 4-input lookup tables (LUTs) as are commonly found in field-programmable gate arrays (FPGAs). The mapping required 95 LUTs.

4.4. *Analysis and discussion*

To compare the compilation results for different mapping parameters at different defect rates, we made the simplifying assumption that the area required by any single mapped circuit was proportional to the number of clusters generated by the compiler for that case. This is crude but will certainly be true to first order, especially given the small size of the generated circuits (between 10 and 31 clusters).

The 'optimal' choice of the mapping parameters (cluster density, function inputs) for a given defect rate was chosen from the graphs in figure 12 by selecting the parameter tuple that yielded the smallest area while still successfully compiling all 50 attempts. The results are in shown in figure 13. It is interesting that the area required was largely independent of the function inputs parameter, up to a defect rate of about 6%. One might expect a larger number of function inputs to produce fewer functions, thus requiring less area. Although fewer functions are indeed produced in that case, the disparity between the sizes of each function's n-FET and p-FET networks makes them more difficult to pack into clusters, negating some of their potential area advantages. Beyond a defect rate of 6%, fewer function inputs generally produced a smaller area penalty.

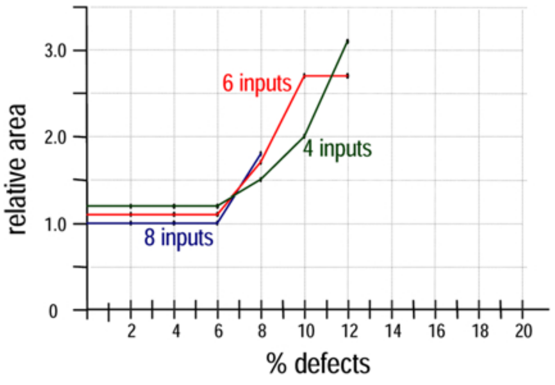

Figure 13. Area required as a function of defect rate, using the 'best' mapping parameters for that rate.

By making some very rough estimates, we may compare these results with a conventional FPGA. If we assume 30 nm centre-to-centre spacing of nanowires, then a complementary/symmetry logic block occupies about 10^{-11} m^2. If we multiply this by a factor of 5 to account for the routing network, a factor of 10 to account for configuration overhead, and another factor of 10 for the 10 arrays needed for the application, this results in a total area for the circuit of about 5×10^{-9} m^2. If we assume that an FPGA (using present technology) can pack 100 000 LUTs (and the supporting routing network) onto a 2 cm $\times$ 2 cm die, then the 95 LUTs for this application would require an area of approximately

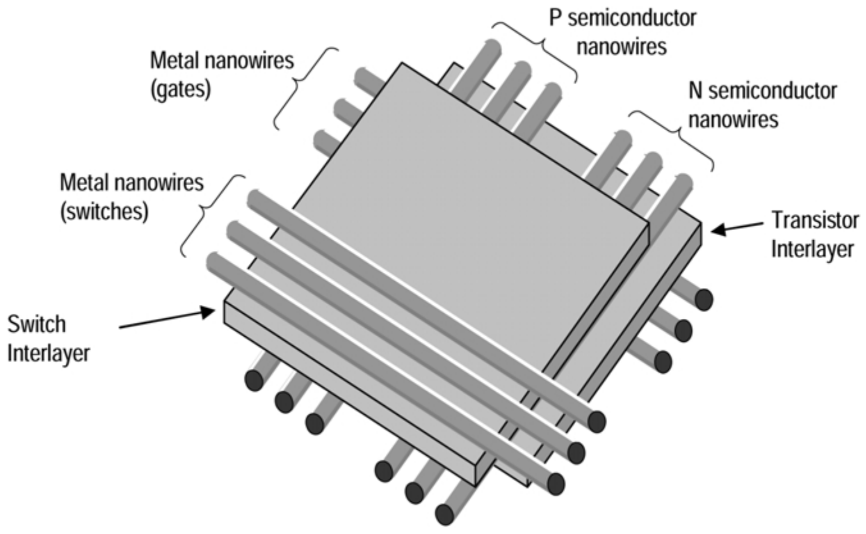
P semiconductor nanowires
Metal nanowires (gates)
N semiconductor nanowires
Metal nanowires (switches)
Transistor Interlayer
Switch Interlayer

Figure 14. If the same interlayer can be used for both p-FET and n-FET arrays, then three layers of wires can be separated with two different interlayers. Each interlayer between adjacent planes would be uniform, eliminating the need for interlayer regions.

4×10^{-7} m^2. The nano crossbar implementation under these assumptions would thus be about 100 times denser than the FPGA.

The study we have done here is only a starting point due to a number of limitations and assumptions.

(1) Only one application, which may not be representative, was mapped onto a single target, which was not optimized.
(2) We have assumed that defects in nanowire/nanowire junctions are uniformly distributed throughout the fabric, neglecting any defect clustering. We have also assumed that nanowire/microwire junction defects are negligible.
(3) We have presumed that all junction defects may be found prior to compilation using tests that run in a reasonable amount of time.
(4) Only 'stuck-open' defects were considered; 'stuck-closed' and other defects such as open or shorted nanowires were ignored.
(5) We have not addressed transient faults or other electrical issues.

5. Fabrication issues

One challenge in the layout of the complementary/symmetry array shown in figure 4 is the need to create different regions within the interlayer with different chemical compositions. For example, the n-FET region is likely to require a different separating chemical layer than the configurable switch regions. This can likely be accomplished through lithographic processes, and the resulting structure will be tolerant of errors in registration or alignment as long as the wires and junctions can be analysed for defects.

Another possibility presents itself if the same chemical interlayer can be used for both the p-FET and n-FET regions. This is at least plausible—the interlayer in both cases may be nothing more than a thin oxide insulator separating the gate from the FET channel in each junction. In this scenario, the complementary/symmetry array and other structures described in this paper can be implemented using three parallel planes of wires, with adjacent planes separated by a uniform interlayer (figure 15) [18]. This would eliminate the need for multiple interlayer regions at the cost of an additional interlayer and an additional plane of wires.

6. Summary

Configurable crossbars of n-FETs and p-FETs may be combined with configurable switch arrays to create a composite structure capable of implementing AND-OR-INVERT gates. Such gates are powerful enough to implement general computation. We have shown how these structures can be combined with routing to create general compute fabrics.

The uniformity and redundancy of crossbars makes possible defect tolerance. Resource allocation algorithms can be used to allocate junctions for logic gates in the presence of defects, and simple routing algorithms can be used to interconnect them. The single interlayer version of these structures is tolerant of variations in layout of different interlayer regions—different regions do not need to be sharply defined nor of the same size as long as the resulting system can be characterized. The double interlayer version is simpler in that each interlayer is homogeneous with no need for different regions, but requires an additional wire layer and interlayer.

References

[1] Chen Y, Jung G, Ohlberg D, Li X, Stewart D, Jeppesen J, Nielsen K, Stoddart J and Williams R 2003 Nanoscale molecular-switch crossbar circuits *Nanotechnology* **14** 462–8
[2] DeHon A 2003 Array-based architecture for FET-based nanoscale electronics *IEEE Trans. Nanotechnol.* **2** 23–32
[3] Stan M, Franzon P, Goldstein S, Lach J and Ziegler M 2003 Molecular electronics: from devices and interconnect to circuits and architecture *Proc. IEEE* (November) 1940–57
[4] Goldstein S and Budiu M 2001 Nanofabrics: spatial computing using molecular electronics *ISCA: Proc. 28th Int. Symp. on Computer Architecture*
[5] Kuekes P J, Williams R S and Heath J R 2000 Molecular wire crossbar memory *US Patent Specification* 6,128,214 (issued October 3)

[6] Kuekes P J, Williams R S and Heath J R 2001 Molecular-wire crossbar interconnect (MWCI) for signal routing and communications *US Patent Specification* 6,314,019 (issued on November 6)

[7] Chen Y and Williams R S 2003 Configurable nanoscale crossbar electronic circuits made by electrochemical reaction *US Patent Specification* 6,518,156 (issued February 11)

[8] Luo Y *et al* 2002 Two-dimensional molecular electronics circuits *Chem. Phys. Chem.* **3** 519–25

[9] Kuekes P 2003 Molecular crossbar latch *US Patent Specification* 6,586,965 (issued July 1)

[10] Tans S J, Verschueren A R M and Dekker C 1998 Room-temperature transistor based on a single carbon nanotube *Nature* **393** 6680

[11] Avouris P, Hertel T, Martel R, Schmidt T, Shea H R and Walkup R E 1999 Carbon nanotubes: nanomechanics, manipulation, and electronic devices *Appl. Surf. Sci.* **141** 201

[12] Nygard J, Cobden D H, Bockrath M, McEuen P L and Lindelof P E 1999 Electrical transport measurements on single-walled carbon nanotubes *Appl. Phys.* A **69** 297

[13] Guo L J, Krauss P R and Chou S Y 1997 Nanoscale silicon field effect transistors fabricated using imprint lithography *Appl. Phys. Lett.* **71** 1881

[14] Kuekes P J and Williams R S 2003 Molecular wire transistor (MWT) *US Patent Specification* 6,559,468 (issued May 6)

[15] Huang Y, Duan X F, Cui Y, Lauhon L J, Kim K H and Lieber C M 2001 Logic gates and computation from assembled nanowire building blocks *Science* **294** 1313

[16] Heath J and Ratner M 2003 Molecular electronics *Phys. Today* (May) 43–9

[17] Packen P 1999 Pushing the limits *Science* **24** 2079

[18] Williams S and Kuekes P 2001 Demultiplexer for a molecular wire crossbar network *US Patent Specification* 6,256,767 (issued July 3)

[19] Snider G, Shackleford B and Carter R 2001 Attacking the semantic gap between application programming languages and configurable hardware *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays (Monterey, CA, Feb. 2001)*

[20] Snider G 2002 Performance-constrained pipelining of software loops onto reconfigurable hardware *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays (Monterey, CA, Feb. 2002)*

[21] Sentovich E *et al* 1992 SIS: a system for sequential circuit synthesis *Technical Report* Memorandum No. UCB/ERL M92/41 University of California, Berkeley

[22] Hogg T and Snider G 2004 Defect-tolerant logic in reconfigurable crossbars *IEEE Trans. Nanotechnol.* to be submitted