Seven Strategies for Tolerating Highly Defective Fabrication

André DeHon and Helia Naeimi
California Institute of Technology

Editor's note:
To tolerate defects in molecular electronics, the authors propose techniques to bypass defective resources during the logic mapping phase. These techniques take advantage of intrinsic redundancy in molecular crossbars to tolerate defective nanowires and nanocrossbars. The proposed greedy mapping algorithm can tolerate a defect density of 10% with very low area overheads.

—Mehdi B. Tahoori, Northeastern University

Bottom-up chemical synthesis techniques provide a promising path to nanometer-scale conductors and devices.[1] Coupled with suitable assembly procedures,[2] these techniques provide a way to integrate dense wiring and computation at nanometer pitches. At these nanometer scales, wires are only a few atoms in diameter and have cross-sectional areas of a few hundred atoms. This small cross section makes these wires fragile, increasing the likelihood that they will break during assembly. Moreover, the contact area between nanowires, and between nanowires and devices, may include only tens of atoms. Consequently, contact integrity depends on a few atomic-scale bonds. Because the atomic-scale features are not perfectly smooth, and the assembly and bond formation are based on statistical processes, some connections could be poor and effectively unusable. This all points to a relatively high defect rate, perhaps in the 1%-to-15% range, for wires and connections.[1,3]

A natural response to defective fabrication is to design architectures with a collection of interchangeable resources and employ post-fabrication programming to configure components so that they use only functional resources, thus avoiding the defective resources. As feature sizes shrink to reach these nanometer scales, however, defect rates inevitably increase. This necessitates reducing the size of the architectures' interchangeable units so that there's a reasonable chance of yielding each replaceable unit. At these high defect rates, it is necessary to employ fine-grained architectures, which use individual wires or wire pairs as the unit of interchange.

Nevertheless, an individual nanowire can have hundreds of junctions. If acceptance policies demanded that all junctions be nondefective for the nanowire to be usable, components with junction defect rates even as high as 1% would not be usable. Fortunately, the nature of logic and interconnect suggests that only a small fraction of the junctions must be connected for any particular, configured nanowire. Consequently, a wire with many nonprogrammable junction defects can still be useful as long as it serves a purpose compatible with its nondefective junctions. In architectures that place hundreds of interchangeable nanowires in arrays—for example, programmable logic arrays (PLAs)—post-fabrication mapping routines can assign nanowires to logical functions (for example, product terms) to avoid such defects. This final assignment then becomes a matching problem between yielded device capabilities and functional needs.

The architecture we present supports this kind of fine-grained sparing and resource matching. The base logic structure is a set of interconnected PLAs. The PLAs and their interconnect consist of large arrays of interchangeable nanowires, which serve as programmable product and sum terms and as programmable interconnect links. Each nanowire can have several defective programmable junctions. We can test nanowires for functionality and use only the subset that provides appropriate conductivity and electrical characteristics. We then perform a matching between nanowire junction programmability and application logic needs to use almost all the nanowires even though most of them have defective junctions. We employ seven high-level strategies to achieve this level of defect tolerance:

- lightweight configurable cross points,
- a reliable support superstructure,
- individual wire sparing,
- *M*-choose-*N* sparing on large sets of interchangeable resources,
- matching to use wires with defective cross points,
- transformations to guarantee cross-point sparseness that matches defect rates, and
- on-chip test and configuration support.

Most of these strategies are general and will apply to many different nanoscale technologies and architectural variations. We ground specifics in our own technology and architecture work to make the ideas concrete. A growing number of researchers are proposing and developing such strategies.[4,5]

Technologies

Our architecture builds on two key emerging technologies. The first lets us produce parallel rows of wires only a few nanometers apart, with diameters of only a few nanometers (say, 3 to 10 nm).[2] The second involves nonvolatile, programmable cross points that fit in the space of the wire crossings between orthogonal arrays of these tight-pitch wires and can be switched between a high and low resistance state.[1] These cross points enable our first strategy. They are significantly more lightweight than programmable cross points in conventional VLSI, which are an order of magnitude larger than the wire crossing area. Consequently, we can employ post-fabrication configurability at a far richer and finer-grained level than was viable in conventional VLSI architectures.

For our second strategy, we integrate these nanoscale conductors with reliable CMOS lithography. The lithography provides a reliable support superstructure for probing and configuring the nanoscale components. The lithographic components can have feature sizes an order of magnitude larger than the dense nanowires. Nevertheless, we can design the CMOS infrastructure so that it occupies only a modest fraction of the total component area. (A more detailed review of the technologies emerging to produce these dense nanowires and cross-point switches is available elsewhere.[6,7])

Architecture

We can fabricate parallel arrays of tight-pitched nanowires, but we cannot fabricate arbitrary geometries with equally tight conductor and device pitches. Consequently, our architecture's core logic and routing exploit crossed arrays of nanowires. Programmable junctions decorate nanowire crossings in select regions. At the larger lithographic scale, we define breaks between nanowires and define regions that can be processed differently using lithographic etching and masking. Each nanowire is accessible from lithographic support wires without relying on any other nanowires.

Figure 1 shows a simple nanoPLA block organization with no interblock routing. The array forms a two-plane PLA cycle. Each plane consists of a programmable OR array followed by a restoration-and-selective-inversion array. Consequently, each plane is a programmable NOR. The combination of NOR-NOR planes is essentially an AND-OR PLA with suitable application of DeMorgan's laws and signal complementation.

As Figure 1 shows, each horizontal wire forms a wired OR. Crossed nanowire inputs connected to a horizontal nanowire through a junction programmed into the low-resistance *on* state can potentially pull up the nanowire, whereas inputs connected through high-resistance *off* junctions do not allow sufficient current to pass through the junction to pull up the nanowire. We precharge the nanowire to a low voltage, using the lithographic scale connections (right side of Figure 1) so that the output is appropriately low when none of the programmed input nanowires is high.

The vertical nanowires serve as buffers or inverters to restore and potentially invert the signals formed on the horizontal, wired-OR nanowires. Each horizontal nanowire can act as a field-effect gate for a vertical wire so that each vertical wire output is simply a restored, potentially inverted, version of a horizontal wire. To make a vertical wire sensitive to a single input, we use nanowires that are doped lightly only in the desired control region and heavily doped elsewhere. The heavily doped region is oblivious to the field of crossing nanowires, whereas the lightly doped region can be controlled by the field of the crossed nanowire. One way to form these arrays is to grow the doping profile into individual nanowires during construction and then assemble a random set of nanowires with their control regions in different positions.[8]

We form a decoder between a set of lithographic wires (see vertical microscale wires A0 through A3 in Figure 1) and the horizontal nanowires. Each nanowire is doped

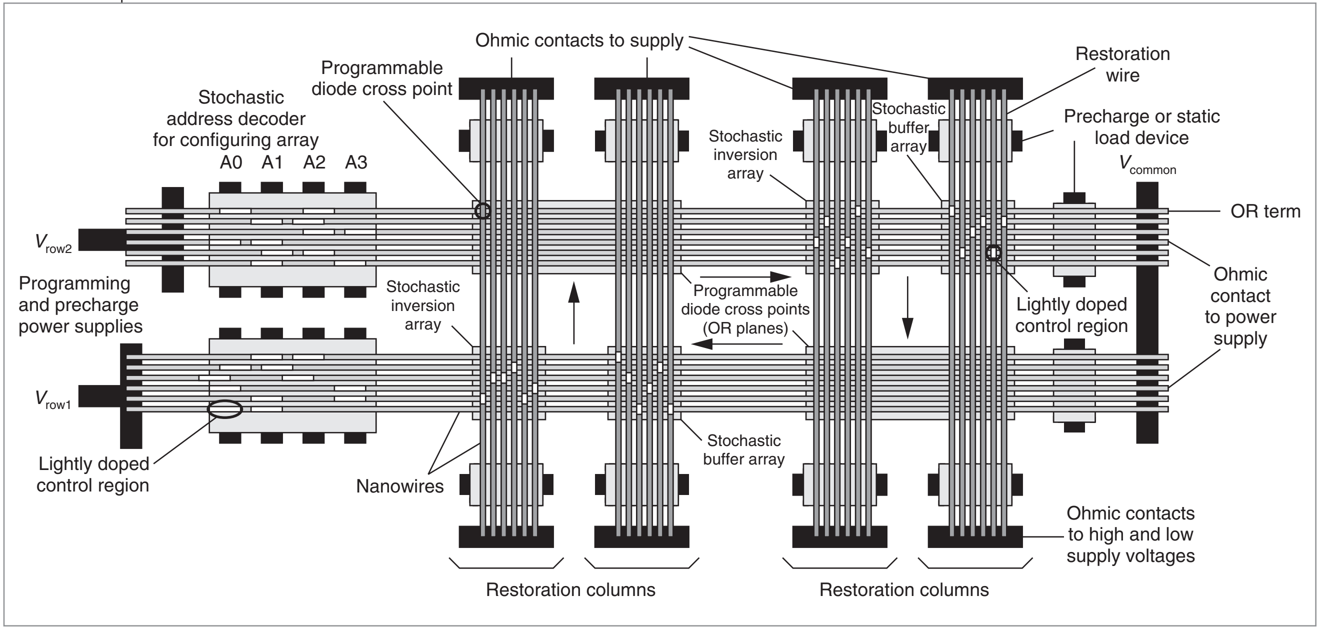
Ohmic contacts to supply
Programmable diode cross point
Stochastic address decoder for configuring array
A0 A1 A2 A3
Stochastic inversion array
Stochastic buffer array
Restoration wire
Precharge or static load device
V_{common}
OR term
V_{row2}
Programming and precharge power supplies
Stochastic inversion array
Programmable diode cross points (OR planes)
Lightly doped control region
Ohmic contact to power supply
V_{row1}
Lightly doped control region
Nanowires
Stochastic buffer array
Ohmic contacts to high and low supply voltages
Restoration columns
Restoration columns

Figure 1. Simple block of crossed nanowires forming a programmable-logic-array block (nanoPLA block) with no interblock routing.

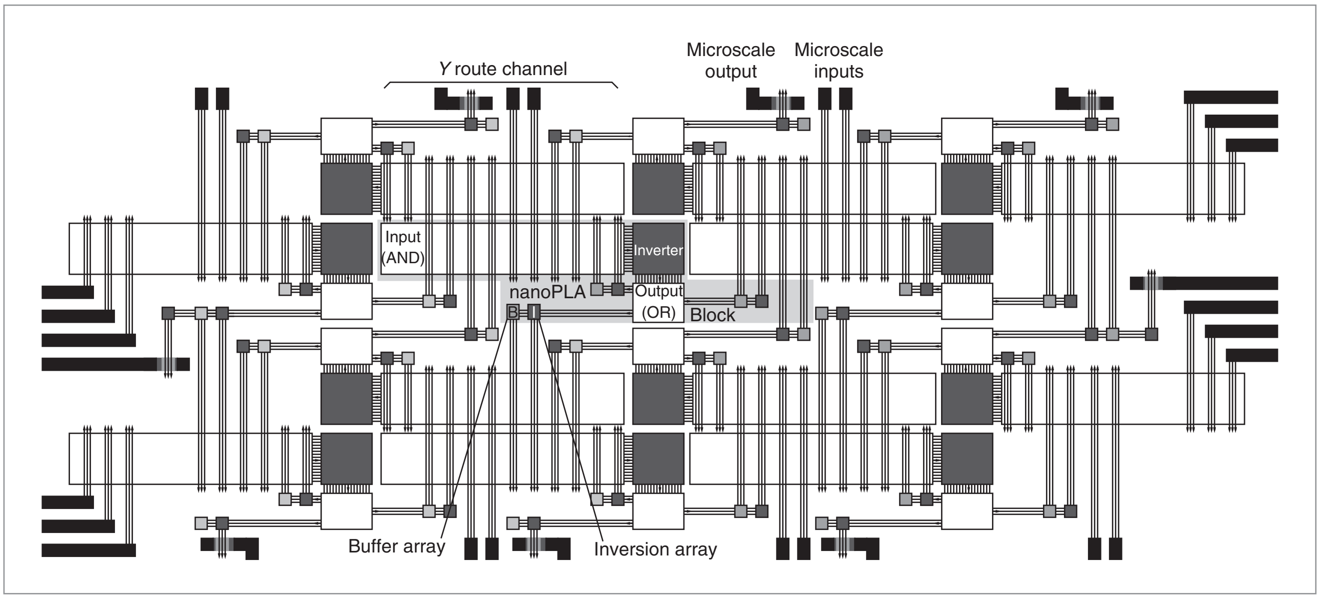

Figure 2. NanoPLA cluster tiling with edge I/O nanowires connected to lithographic scale wires on the edge of the array.

with a unique code, so that the lithographic wires can charge a single nanowire. This lets the lithographic superstructure test each nanowire's connectivity. Furthermore, we can exploit the restoration connections to set or reset a single cross point in either the top or bottom OR plane by using the decoders on these planes to place a differential voltage across this cross point.

In principle, all the nanowires should be equivalent. Therefore, we can assign a given OR function to any of the nanowires within the array, avoiding defective nanowires. A typical array has 100 nanowires in each plane. We can program junctions between broken nanowires and functional nanowires into the disconnected state so that broken wires can be ignored.

To build large components, we can extend these nanoPLA blocks to include I/O to other nanoPLA blocks and assemble them into a large array, as Figure 2 shows. The logic structure is basically the same. However, a given nanoPLA block now has horizontal input nanowires from arrays above and below it. By carefully arranging the overlap between these arrays, we can support arbitrary Manhattan routing (orthogonal routing on a 2D *x*, *y* grid).[7] At a high level, this provides a structure similar to conventional, Island-style FPGAs,[9] where logic blocks consist of lookup tables (LUTs) rather than PLAs but otherwise comprise bit-level logic blocks inside a bit-level, configurable network. Switching occurs through the nanoPLA logic blocks, and the OR arrays now serve as both wired-OR logic and crossbar switching points.

A more complete description of these architectures, their fabrication, and their operation, is available elsewhere.[7,8] The key features to note include the following:

- The entire architecture consists of straight nanowires assembled at tight pitch in regular arrays.
- These nanowires provide all the logic, restoration, and interconnect features needed to implement any computable function entirely at the nanoscale.
- Programming the inexpensive cross points inside the junctions between crossing nanowires personalizes the device to avoid defects and enable the desired logic operation.
- Each nanowire connects directly to a lithographic superstructure for testing and configuration.

Defect model

We abstract our fabrication defects into two groups:

- disconnected wire defects, and
- nonprogrammable cross-point defects.

A wire is either functional or defective. A functional wire has good contacts on both ends and conducts current with a resistance that falls within a designated range. Broken wires cannot conduct current. Poor contacts increase the wire's resistance. Excessive variation in nanowire doping from the engineered target places the wire out of the specified resistance range.

A cross point can be programmable, nonprogrammable, or shorted into the *on* state. A programmable junction can be switched between the *on*- and *off*-state resistance ranges. A nonprogrammable junction can be turned off, but cannot be programmed into the *on* state. A nonprogrammable junction can result from the statistical assembly of too few molecules in the junction or from poor contacts between some of the molecules in the junction and either of the attached conductors. A shorted junction cannot be programmed into the *off* state. Based on the physical phenomena involved, we consider nonprogrammable junctions far more common than shorted junctions. Furthermore, process engineers can tune fabrication to guarantee this. Consequently, we treat shorted junctions like a pair of defective wires, and thus avoid both wires associated with the short. In the remainder of the article, we will simply distinguish between programmable and nonprogrammable junctions.

For the analysis that follows, we assume randomly distributed defects. This assumption is consistent with broken wires resulting from assembly strain, poor contacts made by statistical assembly, and statistical distributions of molecules and connections in junctions. The defect mechanisms anticipated here are very different from those typically in microscale assemblies, and there is insufficient experience in manufacturing these arrays to suggest more sophisticated models (for example, clustering) at this point.

Wire defects

Our third strategy is individual wire sparing. Tolerating wire defects simply involves provisioning adequate spares, separating the good wires from the bad, and configuring the nanoPLA blocks accordingly. For a given PLA design, each block must have a minimum number of usable wires (product terms and interconnect wires). Because wire losses will occur, we design the physical array to include more physical wires to ensure the yield of enough usable wires to meet our logic requirements.

For the detailed architecture just described, wires work in pairs. A horizontal OR term wire provides the programmable computation or programmable interconnect; a vertical restoration wire provides signal restoration and perhaps inversion. If the connections between each restoration wire and its associated OR term wire are not programmable, as with doped restoration nanowires, then a defect in either wire results in an unusable pair. Consequently, each logical OR term or output yields only when both wires yield. Let P_{wire} be the probability that a wire is not defective. The probability of yielding each OR term is

$$P_{or} = P_{wire}^{2} \tag{1}$$

Here, we employ our fourth strategy, M-choose-N sparing on large sets of interchangeable resources. We can perform an *M*-choose-*N* calculation to determine the number of wires we must physically populate (*N*)

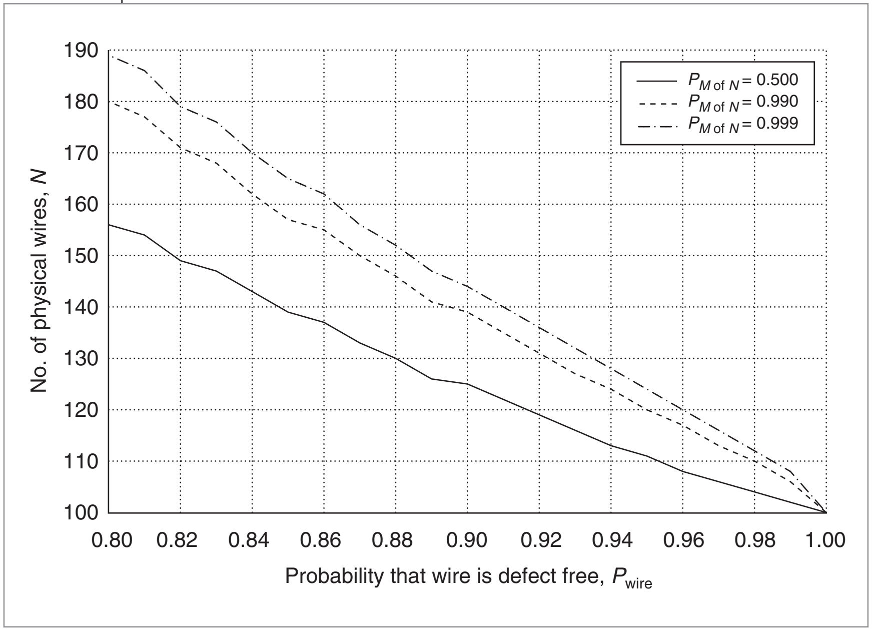

Figure 3. Physical population (*N*) of wires to achieve 100 restored OR terms (*M*).

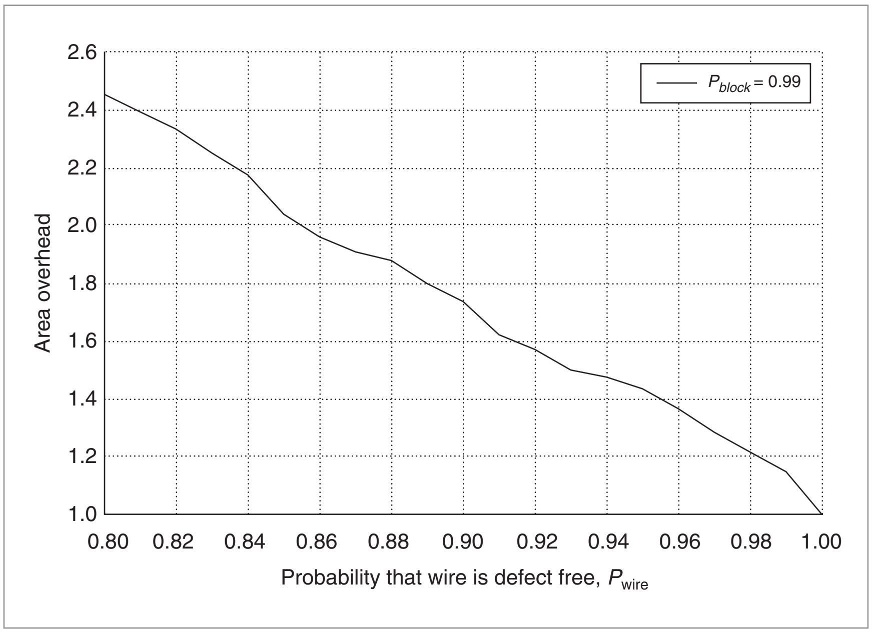

Figure 4. Effect of wire yield rate (P_{wire}) on area for an interconnected nanoPLA block with 80 net product terms and 25 net interconnect wires per channel (nanowire pitch is 10 nm; reliable superstructure pitch is 105 nm).

to achieve a given number of functional wires (M) in the array. The probability that we will yield exactly i restored OR terms is

$$P_{\text{yield}}(N, i) = \binom{N}{i}(P_{\text{or}})^i(1 - P_{\text{or}})^{N-i} \qquad (2)$$

That is, there are $\binom{N}{i}$ ways to select i functional OR terms from N total wires, and the yield probability of each case is $P_{\text{or}}{}^i(1 - P_{\text{or}})^{N-i}$. The nanoPLA architecture has an ensemble with at least M items whenever M or more items yield, so system yield is actually the cumulative distribution function:

$$P_{M\ of\ N} = \sum_{M \le i \le N}\left[\binom{N}{i}(P_{or})^i(1-P_{or})^{N-i}\right] \qquad (3)$$

Equation 3 determines the number of physical wires, N, that we must populate to achieve the desired probability of yielding at least M functional OR terms, $P_{M\text{ of }N}$. For our interconnected nanoPLA blocks, the product terms and interconnect wires are the minimum yield targets, M, and we calculate a corresponding N to determine the number of physical wires we must place in the fabricated nanoPLA block. Figure 3 plots the N required to achieve 50%, 99%, and 99.9% yield rates ($P_{M\text{ of }N}$) as a function of P_{wire} when building $M = 100$ wire arrays.

Once we know the number of physical wires to populate, we can build physical area models to calculate the size of a given array. We can then calculate the area overhead associated with sparing for a given wire defect rate. Figure 4 plots the area overhead as a function of P_{wire} for a typical array, assuming that the reliable, lithographic substrate uses 105-nm pitch wires (for example, a 45-nm technology node) and that the nanowires have a 10-nm pitch. If the design consisted only of nanowires, we'd expect the area to scale as $(N/M)^2$. However, because the nanowires make up only a fraction of the area (see Figure 1), the area overhead scales more slowly; at 80% wire yield rate, we see an overhead of only around 2.4 instead of $(1.8)^2 = 3.2$.

The microscale addressing units and microscale contacts let us identify the addresses of individual, restored OR term wires. A typical testing routine would sequentially test all possible addresses for nanowires in an array, attempting to charge one wire at a time for conduction and restoration. By monitoring the voltage on the microscale supply contact attached to the far end of a restoration column (see Figure 1), we can determine if the address is present and properly restored, and identify the resistance associated with the nanowire's signal path. We record the addresses of each discovered nondefective wire, and then we use only those addresses during subsequent device configuration.

Cross-point defects

Because each nanoPLA block wire has around 100 logical junctions, it's unlikely that any single wire is free of defective junctions. For example, at a 10% nonprogrammable cross-point defect rate, the likelihood of a wire with 100 junctions having no defects is $(0.9)^{100} = 3 \times 10^{-5}$. This suggests only a 0.3% chance that there is even one defect-free wire in an array of 100 nanowires. Consequently, to cope with these high junction defect rates, the nanoPLA must use wires even when they contain defective junctions.

In practice, OR terms are sparsely programmed. Each OR term receives both the true and complement sense of each input, and few product terms use every input to the nanoPLA block. Consequently, most product terms need fewer than 50% of their junctions enabled, and the product term can tolerate nonprogrammability defects in the rest.

Our fifth strategy is to match an OR term's logic to a nanowire's defect pattern. An OR term is compatible with the nanowire's defect pattern if and only if the OR term's inputs are a subset of the nanowire's nondefective junctions. For example, if a nanoPLA's logic array (AND or OR plane) has defective junctions as Figure 5a shows, we can assign OR term $f_1 = a + b + c + e$ to nanowire w_4 even though it has a defective (nonprogrammable) junction at (w_4, d); that is, OR term f_1 is compatible with the defect pattern of nanowire w_4. Because OR term inputs are sparse, a nanowire with defective cross points may still be compatible with many OR terms. Thus, nanowire assignment becomes a matching problem.

Algorithms

Assume we have a logic array with a defect pattern similar to Figure 5a, and we want to program it to the set of OR terms in Figure 5b. We first determine which OR terms are compatible with each nanowire's defect pattern. We then make a graph, as in Figure 5c, showing which OR terms can be assigned to which nanowires. The nodes on the right side of Figure 5c are the OR terms; the nodes on the left are the nanowires. An edge between an OR term and a nanowire indicates that the OR term is compatible with the nanowire's defect pattern. Next, we try to find a complete assignment from the OR terms to the nanowires. Figure 5d shows one possible assignment. Finding an assignment is equivalent to identifying a bipartite graph match. Although a standard, optimal bipartite matching algorithm could find the match, a linear-time greedy heuristic provides reasonably good results for these defect rates while running in substantially less time.[10]

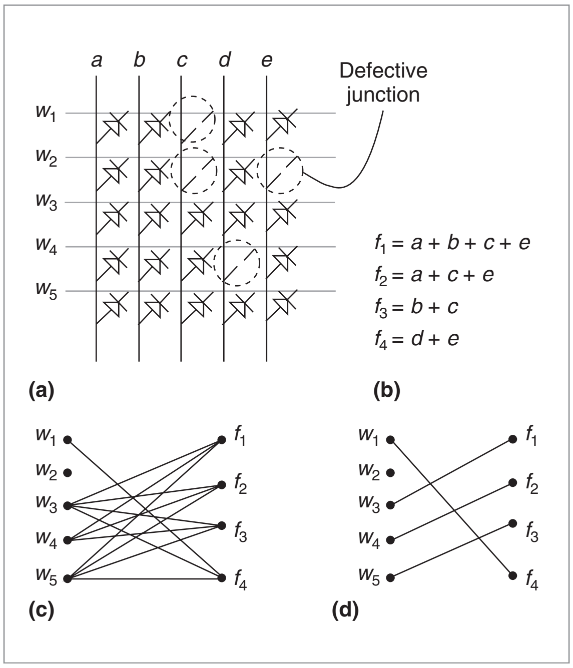

Figure 5. Logic array with defective junctions (a); example set of OR terms (b); bipartite graph with OR terms on the right and nanowires on the left (an edge shows that an OR term can map to a nanowire) (c); and a possible assignment of the OR terms to nanowires (d).

Let F be the set of OR terms, and W the set of nanowires. Let f_i represent an OR term in F, and w_j a nanowire in W. Our heuristic algorithm (shown in Figure 6) picks the f_i terms in decreasing order of their fan-in size (because larger fan-in OR terms are harder to map), and chooses the w_j terms randomly. When the number of *on* junctions per nanowire is bound to a constant, the number of wires tested in line 5 for each OR term f_i is a constant. Consequently, this algorithm runs in linear time, $O(|F|)$. This is even smaller than the $O(|F| \times |W|)$ operations that the optimal algorithm would need to diagnose the programmability of the cross points in the array to construct the full graph for matching.

```
do {
  $f_i$ = unmapped OR term in $F$ with largest fan-in
  do {
      $w_j$ = nanowire randomly selected from unused nanowires in $W$
      if ($f_i$ can be mapped to $w_j$)
          assign $f_i$ to $w_j$
          mark $f_i$ as mapped
          mark $w_j$ as used
  } while ($f_i$ unmapped)
} while (there are unmapped $f_i$ in $F$)
```

Figure 6. Greedy algorithm for matching OR terms to nanowires with potentially defective cross points.

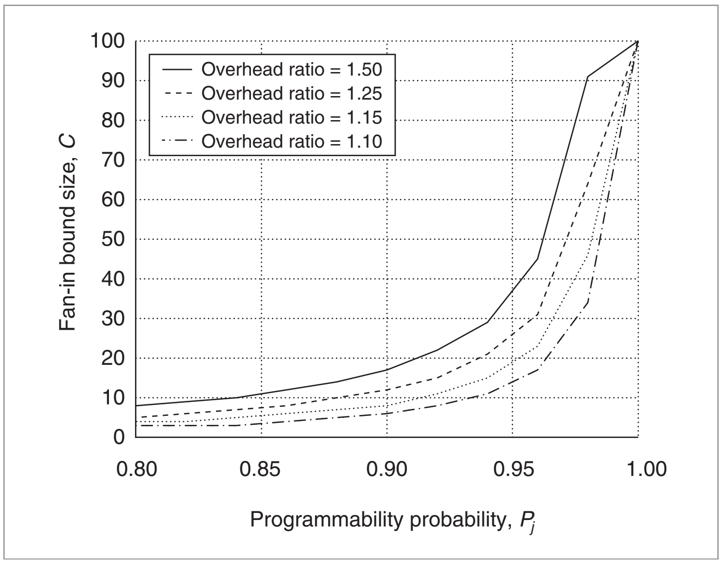

Figure 7. Fan-in limit to tolerate different $|W|/|F|$ overhead ratios versus P_j values for $|F| = 100$.

Fan-in bounding

For the algorithm in Figure 6 to succeed, there must be enough nanowires in the logic array so that every OR term can be assigned to a nanowire. We can derive a lower bound on the number of spare nanowires we need in the array based on the OR term fan-in. The probability that OR term f_i can map to a nanowire in the graph is $(P_j)^{c_i}$ (where c_i is the number of inputs to OR term f_i), because all the c_i junctions must be programmable. The expected number of nanowires connected to f_i in the graph is $|W|(P_j)^{c_i}$, where $|W|$ is the number of nanowires. To find a complete assignment from OR terms to nanowires, the expected size of each OR term's node degree must be at least 1. Assuming the c_i terms are bounded by the maximum fan-in, C:

$$|W|(P_j)^C > 1 \tag{4}$$

This implies that $|W|$ should be greater than $(P_j)^{-C}$. When C is large, $|W|$ must be unacceptably large. For example, with $P_j = 0.85$ and $C = 40$, the above bound suggests $|W| > 665$. Therefore, to allow reasonably sized logic arrays, C must be bounded. For example, to map $|F| = 100$ with little overhead ($|W| \approx 100$), we must keep $C < 28$, because $(0.85)^{-28} \approx 95$.

Ultimately, we must map all the OR terms in each array; hence, Equation 4 is a weak bound. In an earlier work,[10] we derived tighter bounds useful for design. Figure 7 shows the fan-in limits for different area overhead ratios based on these tighter bounds.

Guaranteeing sparseness during mapping

Our sixth strategy involves using transformations to guarantee sparseness that matches defect rates. We control fan-in bound C during logic mapping. A typical step in mapping for clustered PLA designs such as these nanoPLA blocks is to group the logic into clusters that the base logic blocks can implement. For example, a PLA mapping algorithm (PLAmap) can use a netlist of PLA block logic clusters to cover the logic while ensuring that none of these clusters exceeds architectural limitations,[11] including the maximum number of

- inputs to a cluster, I;
- product terms in each cluster, P;
- outputs from a cluster, O; and
- product terms that fan in to any OR term, $P_{\max}$.

In our nanoPLAs, the critical OR term fan-in limit, C, corresponds to the number of inputs that fan in to any particular AND term, $I_{\max,}$ for the AND plane, and $P_{\max}$ for the OR plane. Consequently, once we know the effective defect rate for a given fabrication technology, we can correctly set $P_{\max}$ and $I_{\max}$ and synthesize logic clusters guaranteed to meet the appropriate fan-in bounds (that is, using Equation 4 and the extensions to that equation presented in our earlier work[10]). Restricting $I_{\max}$ and $P_{\max}$ could increase both the number of blocks that the circuit needs and the logic depth; we must compare this situation with the alternative of using less sparse logic and paying a larger mapping overhead.

Experimental overheads

To characterize the impact of defective cross points, we mapped 16 of the benchmarks in the Toronto 20 place-and-route challenge suite (http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html) to nanoPLA arrays with varying cross-point and wire defect rates. We used PLAmap to create clusters.[11] To control I_{max}, we asked PLAmap to generate ($I = I_{max}$, $P = P_{max}$, $O = 1$) single-output covers. We then used T-Vpack (a standard clustering tool from the University of Toronto originally developed for packing LUTs into clusters with limited logic, input, and output capacity) to combine the single-output covers into $I = 20$, $P = 64$ clusters.[9]

Table 1 shows relative area versus defect rate for the 16 designs. For all designs, the additional overhead of fan-in bounding and defect tolerance was below 115%, up to a defect rate of 10% ($P_j \geq 0.9$).

The overhead for most designs was modest because we could map them to bounded fan-in without significantly increasing the number of clusters required to cover the logic task. Table 2 shows how the design sizes scale as we map to different fan-in bounds, $C = I_{max} = P_{max}$. Many of the designs show no increase as we tighten the fan-in bound. In many cases, in fact, a larger fan-in bound can lead to excessive logic duplication and increased area.[11] For a few designs, block count increases considerably as we tighten the fan-in bound, and this is the major factor accounting for the area overhead jumps for the ex1010, pdc, s298, and spla benchmarks.

Table 3 summarizes the composite area overhead of pdc, the design with the largest overhead, as a function of both wire and junction defects. The composite area overhead at $P_{wire} = 0.90$ and $P_j = 0.90$ is less than a factor of 3. After accounting for overheads at these defect rates, the design achieves more than 100 times greater density than an FPGA implementation based on 4-input LUTs in 22-nm CMOS. This density benefit is typical of these benchmark designs.[7]

Bootstraps and testing

Our seventh strategy is to provide on-chip test and configuration support. With larger raw capacities, high defect rates, stochastic assembly, and fine-grained resource sparing, the sheer volume of testing required for nanoPLA arrays would be too large to economically test with conventional, off-chip testers. Consequently, an on-chip microprocessor responsible for both manufacturing tests and local defect remapping must be an integral part of the component architecture. The microprocessor can be implemented in reliable CMOS and need only occupy a tiny fraction of the die area on the chip. The on-chip microprocessor can also play an important role during device configuration. We give the device the logical configuration for each nanoPLA block (the bitstream). The on-chip microprocessor then runs the algorithm to discover each nanoPLA block's present and functional addresses and to perform the local matching necessary to assign logical product and sum terms to physical

Table 1. Relative area versus programmability probability P_j, with 10-nm nanowire pitch and 105-nm reliable superstructure pitch.

Benchmark design	Programmability probability P_j			
	0.85	0.90	0.95	1.00
alu4	1.81	1.64	1.00	1.00
apex2	1.19	1.19	1.00	1.00
apex4	1.30	1.16	1.00	1.00
bigkey	1.00	1.00	1.00	1.00
clma	1.00	1.00	1.00	1.00
des	1.00	1.00	1.00	1.00
dsip	1.00	1.00	1.00	1.00
elliptic	1.00	1.00	1.00	1.00
ex1010	3.81	2.15	1.00	1.00
ex5p	1.00	1.00	1.00	1.00
frisc	1.00	1.00	1.00	1.00
misex3	1.31	1.31	1.00	1.00
pdc	4.75	1.79	1.00	1.00
s298	1.84	1.84	1.00	1.00
seq	1.20	1.12	1.00	1.00
spla	3.46	1.83	1.00	1.00

Table 2. Number of mapped nanoPLA logic blocks for different limitations of fan-in bound C.*

	NanoPLA-mapped block count for fan-in bound C of						
Benchmark design	4	6	8	10	12	16	48
alu4	107	110	98	81	52	26	19
apex2	136	150	152	167	179	178	183
apex4	108	115	123	47	48	34	36
bigkey	99	111	139	147	132	165	168
clma	454	513	557	634	663	646	520
des	102	110	114	130	149	158	120
dsip	70	88	82	108	87	108	79
elliptic	162	209	225	289	338	397	262
ex1010	380	404	402	233	272	351	81
ex5p	88	98	50	50	46	30	8
frisc	213	229	244	285	326	376	276
misex3	97	109	105	102	83	44	37
pdc	291	322	267	292	304	160	41
s298	81	85	84	88	90	102	57
seq	121	135	140	143	138	111	76
spla	211	235	201	219	221	109	33

* $C = I_{max} = P_{max}$, where I_{max} is the maximum number of inputs that fan in to any particular AND term, and P_{max} is the maximum number of product terms that fan in to any OR term.

Table 3. Relative area for the pdc benchmark as a function of P_j and P_{wire}.

	Probability that wire is not defective, P_{wire}				
P_j	0.80	0.85	0.90	0.95	1.00
0.85	10.75	9.17	7.34	6.28	4.71
0.90	4.06	3.46	2.79	2.39	1.80
0.95	2.26	1.91	1.55	1.31	1.00
1.00	2.26	1.91	1.55	1.31	1.00

nanowires. Thus, the device never stores the entire defect map for the component, but simply rediscovers it one nanoPLA block at a time. Moreover, it never exposes the user to the array's defect details.

NanoPLA block sparing

Atop the individual wire sparing just described, it is probably still necessary to spare entire nanoPLA blocks. Larger scale contaminants during assembly or large cluster faults could leave an entire nanoPLA block unrepairable; these defects are not appropriately modeled as independent, random junction, or wire defects. Moreover, as we discussed earlier, we can guarantee high statistical yield of each nanoPLA block, but with millions of nanoPLA blocks in an array, some blocks will not be repairable.

Now that we have used the wire-sparing techniques to bring the yield of the nanoPLA blocks to a respectable level (for example, $P_{M \text{ of } N} = 99\%$), we can use the *M*-choose-*N* sparing strategy at a higher level on the nanoPLA blocks. For example, Lach et al. describe a strategy for tolerating FPGA defects by omitting one logic block from a $k \times k$ tile of FPGA logic blocks and generating logic configurations that accommodate the failure of each physical logic block in the tile.[12]

WE EXPECT that most, if not all, of these strategies will be essential to almost any technology in which wires and devices are built from only a few atoms. Our results suggest that 10% defect rates are tolerable. Higher defect rates may be manageable but will likely require additional techniques in the mapping stage. So far, we have addressed static defects that occur before we map and perform a computation. At this scale, lifetime defects (where cross points or wires fail during operation) could also occur. Consequently, additional techniques will be needed to detect these new defects as they occur, guard the integrity of the computation, and rapidly reconfigure around them. ■

Acknowledgments

We thank the Defense Advanced Research Projects Agency for supporting this research under Office of Naval Research contract N00014-04-1-0591. This material is based on work supported by the Department of the Navy, Office of Naval Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Office of Naval Research.

References

1. Y. Chen et al., “Nanoscale Molecular-Switch Crossbar Circuits,” *Nanotechnology*, vol. 14, no. 4, Apr. 2003, pp. 462-468.
2. D. Whang et al., “Large-Scale Hierarchical Organization of Nanowire Arrays for Integrated Nanosystems,” *Nanoletters*, vol. 3, no. 9, Sept. 2003, pp. 1255-1259.
3. Y. Huang et al., “Logic Gates and Computation from Assembled Nanowire Building Blocks,” *Science*, vol. 294, no. 4545, 9 Nov. 2001, pp. 1313-1317.
4. J.R. Heath et al., “A Defect-Tolerant Computer Architecture: Opportunities for Nanotechnology,” *Science*, vol. 280, no. 5370, 12 June 1998, pp. 1716-1721.
5. S.C. Goldstein and M. Budiu, “NanoFabrics: Spatial Computing Using Molecular Electronics,” *Proc. Int'l Symp. Computer Architecture* (ISCA 01), ACM Press, 2001, pp. 178-189.
6. M. Butts, A. DeHon, and S. Goldstein, “Molecular Electronics: Devices, Systems and Tools for Gigagate, Gigabit Chips,” *Proc. Int'l Conf. Computer Aided Design* (ICCAD 02), ACM Press, 2002, pp. 433-440.
7. A. DeHon, “Design of Programmable Interconnect for Sublithographic Programmable Logic Arrays,” *Proc. Int'l Symp. Field-Programmable Gate Arrays* (FPGA 05), ACM Press, 2005, pp. 127-137.
8. A. DeHon and M.J. Wilson, “Nanowire-Based Sublithographic Programmable Logic Arrays,” *Proc. Int'l Symp. Field-Programmable Gate Arrays* (FPGA 04), ACM Press, 2004, pp. 123-132.
9. V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, 1999.
10. H. Naeimi and A. DeHon, “A Greedy Algorithm for Tolerating Defective Crosspoints in NanoPLA Design,” *Proc. IEEE Int'l Conf. Field-Programmable Technology* (FPT 04), IEEE Press, 2004, pp. 49-56.
11. D. Chen et al., “Performance-Driven Mapping for CPLD Architectures,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 10, Oct. 2003, pp. 1424-1431.

12. J. Lach, W.H. Mangione-Smith, and M. Potkonjak, "Efficiently Supporting Fault-Tolerance in FPGAs," *Proc. Int'l Symp. Field-Programmable Gate Arrays* (FPGA 98), ACM Press, 1998, pp. 105-115.

André DeHon is an assistant professor of computer science at the California Institute of Technology. His research interests include ways to implement computations from physical substrates, including VLSI and molecular electronics, up through architecture, CAD, and programming models. DeHon has an SB, an SM, and a PhD in electrical engineering and computer science from the Massachusetts Institute of Technology. He is a member of the IEEE, the ACM, and Sigma Xi.

Helia Naeimi is a graduate student in the Computer Science Department at the California Institute of Technology. Her research interests include reconfigurable architecture and design patterns, with a special emphasis on reliable systems. Naeimi has a BS in computer engineering from Sharif University of Technology, Tehran, Iran. She is a member of the IEEE.

■ Direct questions and comments about this article to André DeHon, Dept. of CS, 256-80 CALTECH, Pasadena, CA 91125; andre@cs.caltech.edu.

For further information on this or any other computing topic, visit our Digital Library at http://www.computer.org/publications/dlib.

ADVERTISER / PRODUCT INDEX JULY/AUGUST 2005

FUTURE ISSUE: September/October 2005

Networks on Chip

System-on-chip (SoC) design in the forthcoming billion-transistor era will imply the integration of numerous diverse semiconductor IP blocks, and multiprocessor platforms will dominate. As the 2003 International Technology Roadmap for Semiconductors highlights, "Global synchronization becomes prohibitively costly due to process variability and power dissipation, and cross-chip signaling can no longer be achieved in a single clock cycle." Networks on chip (NoCs) are emerging as the design paradigm of choice for the next generation of SoC applications.

Advertising Personnel

Marion Delaney
IEEE Media, Advertising Director
Phone: +1 212 419 7766
Fax: +1 212 419 7589
Email: md.ieeemedia@ieee.org

Marian Anderson
Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown
IEEE Computer Society,
Business Development Manager
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Advertising Sales Representatives

Product Display
John Restchack
Phone: +1 212 419 7578
Fax: +1 212 419 7589
Email: j.restchack@ieee.org

Recruitment Display
Mid Atlantic (recruitment)
Dawn Becker
Phone: +1 732 772 0160
Fax: +1 732 772 0161
Email: db.ieeemedia@ieee.org

New England (recruitment)
Robert Zwick
Phone: +1 212 419 7765
Fax: +1 212 419 7570
Email: r.zwick@ieee.org

Southeast (recruitment)
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynntom@mindspring.com

Midwest/Southwest (recruitment)
Darcy Giovingo
Phone: +1 847 498-4520
Fax: +1 847 498-5911
Email: dg.ieeemedia@ieee.org

Northwest/Southern CA (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Japan (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe (recuirtment)
Hilary Turnbull
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com